A linguagem de programação Erlang

Paulo Ferreira paf(a)dei.isep.ipp.pt

Jornadas Científicas do ISEP

9 de Maio de 2001

Como ensinar programação concorrente e distribuída?

- Alunos sem experiência de trabalho com processos (1º ano de funcionamento da licenciatura).
- Ausência de sistemas paralelos.
- Inexistência de verbas.
- Foco na robustez e elegância e não na "performance absoluta".
- Objectivo: Levar os alunos a pensar de uma forma clara e correcta.

O que é o Erlang?

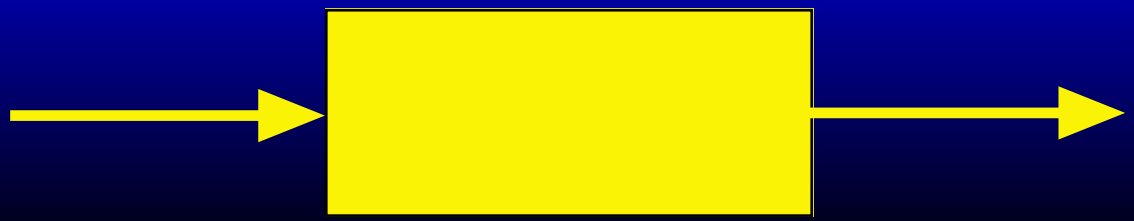
- Uma linguagem funcional, de uso geral, com suporte incorporado para concorrência, distribuição e tolerância a falhas.
- OTP (Open Telecom Platform) é uma plataforma de desenvolvimento de sistemas de telecomunicações.
- É constituída por um sistema "runtime", um conjunto de componentes escritos principalmente em Erlang, e um conjunto de princípios de design de programas.
- Normalmente é usado o conjunto.

O que é uma linguagem funcional?

- Não se mudam os valores das variáveis.
- As funções não possuem “efeitos laterais”.
- Exemplo de efeito lateral: variáveis globais.
- Sem efeitos laterais: puramente funcional.
- Sem efeitos laterais: É mais fácil analizar um programa do ponto de vista matemático.
- Sem efeitos laterais: Cada função pode ser analizada independentemente do ambiente.
- O Erlang não é uma linguagem puramente funcional (I/O e comunicação).

Funções como “caixas negras”

- Para um determinado valor de entrada, a saída é sempre a mesma.
- Podemos “esquecer” o seu interior.
- Temos os “software ICs”, componentes modulares que são independentes entre si.



História do Erlang

- Pesquisa nos laboratórios Ericsson
- Que linguagem usar para desenvolver fácilmente sistemas robustos e fiáveis?
- Experiências com várias linguagens
- Desenvolvimento interno do Erlang
- 1987 -Erlang Primitivo
- 1998 - Lançamento em Open Source

Exemplo de programa

```
-module(mathlib).
-export([factorial/1]).

factorial(1)->1;
factorial(N)->N*factorial(N-1).
```

- Chamada da função: mathlib:factorial(5)

- É uma linguagem declarativa, com influência do Prolog na sintaxe, mas não existe mecanismo de backtracking.
- O facto de ser declarativa confere-lhe uma grande expressividade.
- Não existe declaração do tipo de dados, e a gestão de memória é automática.

Tipos de dados

- Números: Inteiros e Floats
- Àtomos
- Tuplos
- Listas
- Pids (Process Ids)
- Ports (canais de comunicação com programas escritos noutras linguagens)
- Referências (objectos únicos ao nível de vários nós)
- Binários (para manipulação de pacotes ou outros conjuntos de bits)

Mais exemplos

```
area({quadrado,Lado})->Lado*Lado;
area({rectangulo,Lado1,Lado2})->Lado1*Lado2;
area(X)->{nao_sei_fazer_area,X}.
```

- O mecanismo de "pattern matching" ajuda à clareza do programa e podemos "etiquetar"os tuplos com átomos, para que tudo fique mais claro.

```
sort([Pivot|T]) ->
    sort([ X || X <- T, X < Pivot])
    ++ [Pivot]
    ++ sort([ X || X <- T, X >= Pivot]);
sort([]) -> [].
```

Mais um exemplo

```
-define(PROTO1,22).
-define(PROTO2,33).
-define(PROTO3,1256).
%%  Tratar um pacote segundo o protocolo certo

tratar(<<?PROTO1:8,Lixo:24,Chicha/binary>>)->func1(Chicha);
tratar(<<?PROTO2:8,Lixo:32,Chicha/binary>>)->func2(Chicha);
tratar(<<?PROTO3:16,Chicha/binary>>)->func3(Chicha);
tratar(X)->protocolo_nao_reconhecido(X).
```

Processos e mensagens

```
Pid=spawn(m,f,[Arg1,Arg2])

Pid ! Mensagem

receive
    Mensagem1 -> acção1;
    Mensagem2 -> acção2;
    after Timeout -> acção_timeout
end
```

- A comunicação é feita através de mensagens, cujo envio tem sempre sucesso.

Exemplo de um servidor

```
start() -> spawn( m, init, [...]).

init(...) -><initialization>,
    loop(...).
loop(...) ->   receive
           stop -> true;
           Pattern1 -> <actions>, loop(...);
           ...
           PatternN -><actions>, loop(...)
       end.
```

Mais facilidades para processos

- Mensagens de “aviso” quando um processo “ligado” termina.
- Todas as excepções são locais.
- Isto permite a constituição de uma hierarquia de processos para uma maior robustez.
- Processos podem ser registados com um “nome” sendo acedidos através do nome.
- “Upgrade” do código sem parar o processo.

Distribuição

- Arrancar os nós de forma correcta.

```
Pid=spawn(nó,m,f,[Arg1,Arg2])
```

- A partir daí tudo funciona exactamente na mesma.

“Last call optimization”

```
servidor(…)->recebe_mensagem(…),
        trata_mensagem(…),
        servidor(…).
```

- Temos uma função recursiva que se chama a si própria.
- Como existe “LCO” a função corre num espaço constante.
- Isto é verdadeiro para funções que terminam com uma chamada a si próprias, sem cálculos pendentes.

Bibliotecas e interfaces existentes (exemplos)

- Compilador, kernel e biblioteca standard
- Handler para eventos e alarmes, SNMP, medidor de avaliação.
- ASN1, interfaces de baixo nível para C e Java, Servidor Web e cliente Ftp.
- Chamada a objectos COM em windows, SSL e criptografia.
- Base de dados Mnesia (tempo real) e interface ODBC.
- Serviços CORBA e compilador IDL.
- Ferramentas de debugging e monitorização.

Exemplos prático de uso:

- Switch ATM AXD301 10-160 Gbits.
- ANx-DSL nó de acesso ADSL
- Mobility Server
- 1º Demo de GPRS
- Eddieware
- Bluetail - Web Prioritizer + Mail Robustifier (21 meses, 25 empregados, $152M USD)

Características validadas na prática

- Grande número de processos por nó - 4000 no AXD301).
- “Soft RealTime” - Custo das chamadas.
- Sistemas distribuídos- A versão de 40 Gbits do AXD301 tem 72 processadores.
- Comunicação com hardware - Interfaces standard para C e device drivers,
- Sistemas grandes: AXD301 Rel3.2 tem 1M linhas de Erlang.
- Funcionalidade complexa: AXD301 protoc. ITU + ATM Forum.
- Operação contínua: Mobility Server (1994) com 400 produtos.
- Manutenção do software: Upgrades sem parar o sistema.
- Requisitos de qualidade e fiabilidade: GPRS 99.995% de disponibilidade.
- Tolerância a falhas de hardware e software: falhando um dos processadores do GPRS apenas baixa a capacidade do sistema.

Sumário da experiência

- Trauma inicial: “Para quê?”
- “Ah, é parecido com prolog!”
- Passagem da programação imperativa a funcional, pode “bloquear” alunos que encaram o Erlang como mais uma “linguagem normal”.
- Realização de trabalho prático ajuda na compreensão da linguagem.
- No final, reacção positiva por parte dos alunos.
- Ajuda recebida dos docentes envolvidos.
- Trabalho de projecto sobre o Eddie.
- Usado na disciplina de AISC

Trabalho futuro em OC

- Trabalhos práticos mais complexos.
- Uso mais aprofundado das facilidades do OTP.
- Uso e implementação de protocolos de comunicação mais elaborados.
- Interface com outras linguagens.

Preconceitos iniciais dos alunos

- “Se isso não é usado para que é que serve?”
- “Para que é que temos de aprender outra linguagem?”
- “Uma linguagem não chega para todos os tipos de programas?”
- “Se no meu emprego trabalho na linguagem X, de que é que me serve isso?”

Afinal o que é programar?

- Resolver problemas decompondo-os em problemas mais simples.
- Paradigmas da programação fornecem maneiras de estruturar a "fragmentação" dos problemas.
- As linguagens são uma forma de "explicitar" os conceitos que possuímos.
- Se o conceito não existir na linguagem teremos problemas em o expresssar.

Vantagens das Linguagens Funcionais

- Se as funções não possuem efeitos laterais então podem ser analisadas de uma forma independente do contexto.
- Torna-se mais fácil a decomposição do problema/programa em funções.
- Torna-se mais fácil a reutilização de código.
- Torna-se mais fácil (ou possível) a análise formal do programa.
- Torna-se mais fácil a utilização de funções como parâmetros de outras funções.

Porque é que não são mais usadas?

- A indústria não tem tempo para ser criativa, e a academia não é flexivel devido a restrições várias.
- Trauma das teorias matemáticas normalmente associadas às linguagens funcionais.
- Carga semântica extremamente forte de algumas linguagens funcionais.
- Vistas mais como assunto de pesquisa do que como ferramentas práticas.
- Falta de "ferramentas CASE".
- O debate sobre linguagens de programação assume muitas vezes contornos de "fanatismo religioso" onde se defende o que se conhece, contra a incerteza do que não se conhece.

Problemas do mundo académico

- Conservadorismo.
- Psicose de "seguir as linguagens que a indústria quer".
- A indústria na realidade não quer pessoas que saibam a linguagem X.
- A indústria quer pessoas que saibam programar, o que é algo muito diferente.
- A indústria deseja pessoas que digam: "Esta linguagem não presta, existe uma muito melhor, que nos pode fazer ganhar muito mais dinheiro."

Problemas da indústria

- Decisões estratégicas ao nível da tecnologia realizadas com o completo desconhecimento dos problemas em causa.
- Escolha de candidatos a empregos baseada no número de palavras-chave que aparece no curriculum.
- Medo da diferença.

Conclusões

- As linguagens funcionais podem ser ensinadas/aprendidas fácilmente, simplificando a escrita e compreensão de algoritmos avançados de manipulação de dados.
- É recomendável que no curso seja dada uma maior visibilidade e utilização às linguagens funcionais.
- Cada vez mais os programas são constítuídos por um grande número de processos que interagem entre si, e os paradigmas tradicionais de programação têm problemas de adaptação a estes novos modelos computacionais, enquanto que linguagens como o Erlang se adaptam de uma forma notável.