

Extensões ao Erlang

Paulo Ferreira
paf@dei.isep.ipp.pt

Fevereiro de 2006

- Records**
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Records

Records

- Structs em C ou Records em Pascal
- Permitem referenciar os elementos pelo nome
- Os tuplos permitem referenciar pelo número – `Nome=elem(1,Tuplo)`
- Mas se mudamos a ordem dos elementos, retiramos um elemento, ou adicionamos um elemento, temos de alterar o código
- Os records permitem que os seus elementos sejam referenciados pelo nome

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Exemplo

- Exemplo de definição:

- ▲ `-record(pessoa, {nome, telefone, endereço}).`

- Se P for uma variável do tipo pessoa, podemos usar a seguinte sintaxe:

- ▲ `Nome=P#pessoa.nome,`
`Endereço=P#pessoa.endereço,`
`...`

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Definição

Um record é definido da seguinte maneira:

```
-record(NomedoRecord,  
        {Campo1 [=ValorPorDefeito1],  
         Campo2 [=ValorPorDefeito2],  
         . . . ,  
         CampoN [=ValorPorDefeitoN]}).
```

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Definição

Um record é definido da seguinte maneira:

```
-record(NomedoRecord,  
        {Campo1 [=ValorPorDefeito1],  
         Campo2 [=ValorPorDefeito2],  
         ... ,  
         CampoN [=ValorPorDefeitoN]}).
```

- O nome do record e os nomes dos campos devem ser átomos
- Os valores por defeito (opcionais) são usados se ao criar um record não dermos valores aos campos
- Se não existir um valor por defeito o campo assume o valor `undefined`

Definição

Um record é definido da seguinte maneira:

```
-record(NomedoRecord,  
        {Campo1 [=ValorPorDefeito1],  
         Campo2 [=ValorPorDefeito2],  
         . . . ,  
         CampoN [=ValorPorDefeitoN]}).
```

- O nome do record e os nomes dos campos devem ser átomos
- Os valores por defeito (opcionais) são usados se ao criar um record não dermos valores aos campos
- Se não existir um valor por defeito o campo assume o valor `undefined`
- Exemplo: `-record(pessoa, {nome=, tel=[], ender})`.

Detalhes

- Se o record for usado em vários módulos a sua definição deve ser colocada num ficheiro `.hrl` e os módulos que usam essa definição devem usar um

- ▲ `-include(NomeDoFicheiro).`

- Exemplo:

- ▲ `-include("os_meus_records.hrl").`

- A definição de um record deve vir antes do seu uso

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Sintaxe

- Um record é criado da seguinte maneira:

- ▲ `#NomeDoRecord{Campo1=Valor,
... ,
CampoM=ValorM}`.

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Sintaxe

- Um record é criado da seguinte maneira:
 - ▲ `#NomeDoRecord{Campo1=Valor,
... ,
CampoM=ValorM}`.
- Se algum dos campos fôr omitido, é usado o valor por defeito.
- Exemplo:
 - ▲ `P=#pessoa{tel=[2,2,8,3,4,0,5,0,0],nome="José"}`.

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Sintaxe

- Um record é criado da seguinte maneira:
 - ▲ `#NomeDoRecord{Campo1=Valor, ... , CampoM=ValorM}`.
- Se algum dos campos fôr omitido, é usado o valor por defeito.
- Exemplo:
 - ▲ `P=#pessoa{tel=[2,2,8,3,4,0,5,0,0],nome="José"}`.
- Seleccionar apenas um campo:
 - ▲ `Variavel#nome_do_record.campo`
 - ▲ `P#pessoa.tel`

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Sintaxe

- Um record é criado da seguinte maneira:
 - ▲ `#NomeDoRecord{Campo1=Valor, ... , CampoM=ValorM}`.
- Se algum dos campos fôr omitido, é usado o valor por defeito.
- Exemplo:
 - ▲ `P=#pessoa{tel=[2,2,8,3,4,0,5,0,0],nome="José"}`.
- Seleccionar apenas um campo:
 - ▲ `Variavel#nome_do_record.campo`
 - ▲ `P#pessoa.tel`
- Mudar o valor de um campo:
 - ▲ `P2=P1#pessoa{tel=[2,2,8,3,4,0,5,0,1]}`

Testes

- Testar se um record é de um dado tipo:

- ▲ `record(Variável,Tipo).`

- Exemplo:

- ▲ `funcao(P) when record(P,pessoa)->pessoa;
funcao(_)->nao_e_pessoa.`

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Representação interna

- Os records são representados internamente em Erlang como tuplos, em que o primeiro membro do tuplo é o nome do tipo de record

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Representação interna

- Os records são representados internamente em Erlang como tuplos, em que o primeiro membro do tuplo é o nome do tipo de record
- Existe uma função que nos ajuda a trabalhar melhor com records:

`record_info(fields,Record)`

devolve uma lista com os campos

`record_info(size,Record)`

devolve o número de campos

`record_info(fields,pessoa)`

devolve `[nome,tel,ender]`

`record_info(size,pessoa)`

devolve 3

Records

Records

Exemplo

Definição

Detalhes

Sintaxe

Testes

Representação interna

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

Listas

Macros

Bins

Funs

Exemplos 1

```
%% File: person.hrl
%%-----
%% Data Type: person
%% where:
%% name: A string (default is undefined).
%% age: An integer (default is undefined).
%% phone: A list of integers (default is []).
%% dict: A dictionary containing various information
%% about the person.
%% A {Key, Value} list (default is the empty list).
%%-----

-record(person, {name, age, phone = [], dict = []}).
```

Exemplos 2

```
-module(person).
-include("person.hrl").
-compile(export all). % For test purposes only.
%% This creates an instance of a person.
%% Note: The phone number is not supplied so the
%% default value [] will be used.
make_hacker_without_phone(Name, Age) ->
    #person{name = Name, age = Age,
dict = [{computer_knowledge, excellent},
{drinks, coke}]}.
```

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2**
- Exemplos 3
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Exemplos 3

```
%% This demonstrates matching in arguments
print(#person{name = Name, age = Age, phone = Phone,
        dict = Dict}) - >
io:format("Name: ~s, Age: ~w, Phone: ~w ~n"
"Dictionary: ~w.~n", [Name, Age, Phone, Dict]).

%% Demonstrates type testing, selector, updating.
birthday(P) when record(P, person) ->
    P#person{age = P#person.age + 1}.
```

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3**
- Exemplos 4
- Listas
- Macros
- Bins
- Funs

Exemplos 4

```
register_two_hackers() ->
    Hacker1=make_hacker_without_phone("Joe", 29),
    OldHacker=birthday(Hacker1),
% The central register server should have
% an interface function for this.
    central_register_server!
    {register_person, Hacker1},
    central_register_server ! {register_person,
    OldHacker#person{name = "Robert",
    phone = [0,8,3,2,4,5,3,1]}}.
```

- Records
- Records
- Exemplo
- Definição
- Detalhes
- Sintaxe
- Testes
- Representação interna
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4**
- Listas
- Macros
- Bins
- Funs

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Listas

Operações com Listas

- ++ Concatenação de duas Listas

- ▲ L1 ++ L2 ++ L3

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Operações com Listas

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

■ ++ Concatenação de duas Listas

▲ L1 ++ L2 ++ L3

■ -- Subtração de duas Listas

▲ L1 -- L2

▲ Produz uma lista que é o resultado da primeira lista, depois de retiradas as primeiras ocorrências de cada um dos elementos da segunda lista.

Operações com Listas

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs


- ++ Concatenação de duas Listas

- ▲ $L1 ++ L2 ++ L3$

- -- Subtração de duas Listas

- ▲ $L1 -- L2$

- ▲ Produz uma lista que é o resultado da primeira lista, depois de retiradas as primeiras ocorrências de cada um dos elementos da segunda lista.

-  Cuidado com prioridades, usar parênteses como precaução, quando se usa mais do que um tipo de operação!

Compreensão de Listas

- Constituem uma notação sucinta para gerar os elementos de uma lista
- *Set Comprehension* na teoria dos conjuntos de Zermelo-Frankel
- Expressões ZF em Miranda e SASL
- Similares¹ ao `setof` e `findall` em Prolog

¹Parecidos mas não iguais!

Sintaxe

■ Sintaxe:

```
[Expressão || Qualificador1, Qualificador2, ... ]
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Sintaxe

- Sintaxe:
`[Expressão || Qualificador1, Qualificador2, ...]`
- Expressão é uma expressão arbitrária e cada qualificador pode ser um Gerador ou um Filtro

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Sintaxe

- Sintaxe:
`[Expressão || Qualificador1, Qualificador2, ...]`
- Expressão é uma expressão arbitrária e cada qualificador pode ser um Gerador ou um Filtro
- Um Gerador é escrito como `Padrão < -ListaExpr`
- `ListExpr` deve ser uma expressão que dá uma lista de termos
- Um Filtro é um predicado ou uma expressão booleana.
- Um predicado é uma função que devolve `true` ou `false`.

Exemplos 1

```
> [X || X <- [1,2,a,3,4,b,5,6], X > 3].  
[a,4,b,5,6]
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Exemplos 1

```
> [X || X <- [1,2,a,3,4,b,5,6], X > 3].  
[a,4,b,5,6]
```

■ Deve ler-se:

▲ A lista dos X tais que X é retirado da lista [1,2,...] e X é maior que 3

$X <- [1,2,a,3,4,b,5,6]$ é um gerador
 $X > 3$ é um filtro

Exemplos 2

- Podemos adicionar mais filtros:

```
[X || X <- [1,2,a,3,4,b,5,6], integer(X), X>3].  
[4,5,6]
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Exemplos 2

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

- Podemos adicionar mais filtros:

```
[X || X <- [1,2,a,3,4,b,5,6], integer(X), X>3].  
[4,5,6]
```

- Podemos combinar geradores para ter o produto cartesiano de duas listas:

```
> [{X, Y} || X <- [1,2,3], Y <- [a,b]].  
[{1,a},{1,b},{2,a},{2,b},{3,a},{3,b}]
```


Quick Sort

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

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Permutações

```
perms([]) -> [[]];  
perms(L) -> [[H|T] || H <- L, T <- perms(L--[H])].
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Triplos Pitagóricos

- Conjuntos de inteiros tais que: $A^2 + B^2 = C^2$
- A função `pyth(N)` gera a lista dos inteiros que cumprem essa condição para $A + B + C \leq N$.

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Triplos Pitagóricos

- Conjuntos de inteiros tais que: $A^2 + B^2 = C^2$
- A função `pyth(N)` gera a lista dos inteiros que cumprem essa condição para $A + B + C \leq N$.
- Exemplo:

```
pyth(N) -> [ {A,B,C} || A <- lists:seq(1,N),  
               B <- lists:seq(1,N),  
               C <- lists:seq(1,N),  
               A+B+C =< N,  
               A*A+B*B == C*C ].
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Triplos Pitagóricos

- Conjuntos de inteiros tais que: $A^2 + B^2 = C^2$
- A função `pyth(N)` gera a lista dos inteiros que cumprem essa condição para $A + B + C \leq N$.
- Exemplo:

```
pyth(N) -> [ {A,B,C} || A <- lists:seq(1,N),  
                B <- lists:seq(1,N),  
                C <- lists:seq(1,N),  
                A+B+C =< N,  
                A*A+B*B == C*C ].
```

- Mais eficiente:

```
pyth1(N) -> [{A,B,C} || A <- lists:seq(1,N),  
                    B <- lists:seq(1,N-A+1),  
                    C <- lists:seq(1,N-A-B+2),  
                    A+B+C =< N,  
                    A*A+B*B == C*C ].
```

Regras – Variáveis

- Todas as variáveis que ocorrem num gerador são variáveis *novas* (*fresh*)
- Todas as variáveis que estavam definidas antes da compreensão de listas e que são usadas em filtros possuem o valor que tinham antes
- Não se pode exportar o valor de nenhuma variável

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Na prática 1

- Escrever a função `select` que selecciona certos elementos de uma lista de tuplos

```
select(X,L)->[Y | |{X,Y}<-L] .
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Na prática 1

- Escrever a função `select` que selecciona certos elementos de uma lista de tuplos

`select(X,L) -> [Y | |{X,Y}<-L] .`

- Ideia: Fazer uma lista com todos os `Y` que constituem os segundos elementos dos tuplos da lista `L`, onde `X` é o primeiro elemento

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Na prática 1

- Escrever a função `select` que selecciona certos elementos de uma lista de tuplos

```
select(X,L)->[Y || {X,Y}<-L] .
```

- Ideia: Fazer uma lista com todos os `Y` que constituem os segundos elementos dos tuplos da lista `L`, onde `X` é o primeiro elemento

- Ao compilar:

```
./FileName.erl:Line: Warning:  
variable 'X' shadowed in generate
```

- Problema: O `X` no padrão não é o mesmo `X` da cabeça da função.

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Na prática 2

■ Resultado:

```
>select (b, [{a,1},{b,2},{c,3},{b,7}]).  
[1,2,3,7]
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Na prática 2

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

■ Resultado:

```
>select(b, [{a,1},{b,2},{c,3},{b,7}]).  
[1,2,3,7]
```

■ Corrigindo:

```
select(X, L) -> [Y || {X1, Y} <- L, X == X1].
```

- O gerador contém apenas variáveis não instanciadas, e o teste é feito no filtro, e agora tudo funciona:

```
>select(b, [{a,1},{b,2},{c,3},{b,7}]).  
[2,7]
```

Recomendações

- Certas operações de *pattern matching* devem estar nos filtros e não podem ficar nos geradores

- Não devemos escrever:

```
f(...) -> Y = ...  
[ Expressão || PadrãoComY <- Expr, ... ]  
...
```

- Mas devemos escrever antes:

```
f(...) -> Y = ...  
[ Expressão || PadrãoComY1 <- Expr, Y == Y1, ... ]  
...
```

Records

Listas

Operações com Listas

Compreensão de Listas

Sintaxe

Exemplos 1

Exemplos 2

Quick Sort

Permutações

Triplos Pitagóricos

Regras – Variáveis

Na prática 1

Na prática 2

Recomendações

Macros

Bins

Funs

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Macros

Sintaxe

- Os Macros podem ser escritos em Erlang usando a seguinte sintaxe:
`-define(Constante, Valor).`
`-define(fun(Var1, Var2, ..., Var), Substituição).`
- Os macros são expandidos quando se usa a forma `?Nome_do_Macro`
- Se definirmos por exemplo:
`-define(timeout, 200).`
- A expressão `?timeout` será então substituída por 200 sempre que aparecer.

- Records
- Listas
- Macros
- Sintaxe**
- Argumentos
- Pré-definidos
- Condicionais
- Uso de condicionais 1
- Uso de condicionais 2
- Expansão de macros
- Bins
- Funs

Argumentos

- Exemplo do uso de argumentos em macros:
`-define(macro1(X,Y), {a,X,b,Y}).`

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Argumentos

- Exemplo do uso de argumentos em macros:
`-define(macro1(X,Y), {a,X,b,Y}).`
- Podemos usar este macro da seguinte forma:

```
bar(X) ->  
    ?macro1(a,b),  
    ?macro1(X,123).
```

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Argumentos

- Exemplo do uso de argumentos em macros:
`-define(macro1(X,Y), {a,X,b,Y}).`
- Podemos usar este macro da seguinte forma:

```
bar(X) ->  
    ?macro1(a,b),  
    ?macro1(X,123).
```

- Este macro será expandido para:

```
bar(X) ->  
    {a,a,b,b}  
    {a,X,b,123}.
```

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Pré-definidos

Estão pré-definidos os seguintes macros:

- ?MODULE - devolve o nome do módulo
- ?FILE - devolve o nome do ficheiro
- ?LINE - devolve o número da linha corrente
- ?MACHINE - devolve o nome da máquina virtual ('JAM', 'BEAM' ou 'VEE').

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Condicionais

Estão definidas as seguintes directivas condicionais:

- `undef`(Macro) . - remove a definição do Macro
- `ifdef`(Macro) . - faz as linhas seguintes se Macro estiver definido
- `ifndef`(Macro) . - faz as linhas seguintes se Macro não estiver definido
- `else` . - macro de *else*
- `endif` . - macro de *endif*

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Uso de condicionais 1

- Os macros condicionais usam-se normalmente agrupados da seguinte forma:

```
-ifdef(debug).
```

```
-define(...).
```

```
-else.
```

```
-define(...).
```

```
-endif.
```

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Uso de condicionais 2

- Podemos ver isso no seguinte exemplo:

```
-define(debug, true).  
-ifdef(debug).  
-define(trace(Str, X),  
    io:format("Mod:~w line:~w ~p ~p~n", [MODULE, LINE, Str, X])).  
-else.  
-define(trace(X, Y), true).  
-endif.
```

Records

Listas

Macros

Sintaxe

Argumentos

Pré-definidos

Condicionais

Uso de condicionais 1

Uso de condicionais 2

Expansão de macros

Bins

Funs

Uso de condicionais 2

- Podemos ver isso no seguinte exemplo:

```
-define(debug, true).  
-ifdef(debug).  
-define(trace(Str, X),  
  io:format("Mod:~w line:~w ~p ~p~n", [MODULE, LINE, Str, X])).  
-else.  
-define(trace(X, Y), true).  
-endif.
```

- Dadas estas definições a expressão `?trace("X=",X)`, na linha 100 do módulo `foo`, é expandida para:

```
io:format("Mod:~w line:~w ~p ~p~n", [foo,100,"X=", [X]]),
```
- Se removermos o define de `debug`, a expressão será expandida para `true`

Expansão de macros

- Podemos usar o seguinte código:

```
-module(mexpand).  
-export([file/1]).  
file(File) ->  
case epp:parse_file(File ++ ".erl", [], []) of  
  {ok, L} ->  
    {ok, Stream} = file:open(File ++ ".out", write),  
                  lists:foreach(fun(X) ->  
                                io:format(Stream, "~s~n", [erl_pp:form(X)])end, L),  
  file:close(Stream)  
end.
```

- Outra forma de expandir macros é compilar o ficheiro com a opção 'P'
`compile:file(Ficheiro, ['P'])`
- Produz uma listagem com o nome `Ficheiro.P` em que podemos ver o resultado de qualquer expansão de macros

Records

Listas

Macros

Bins

Bins

Exemplos

Especificação de
segmentos

Exemplos 1

Exemplos 2

Exemplos 3

Funs

Bins

Bins

- Um *bin* é uma sequência de bytes de baixo nível
- Exemplo:
Bin=<<E1,E2,...En>>
- Cada elemento especifica um certo segmento do binário
- Um segmento é um conjunto de bits contíguos
- A fronteira dos segmentos não se encontra alinhada ao byte

Records

Listas

Macros

Bins

Bins

Exemplos

Especificação de
segmentos

Exemplos 1

Exemplos 2

Exemplos 3

Funs

Exemplos

```
Bin1 = <<1,17,42>>
```

```
Bin2 = <<"abc">>
```

```
A=1, B=17, C=42, Bin3=<<A,B,C:16>>
```

- Se fizermos:

```
<<D:16,E,F/binary>>=Bin3
```
- dará:
 - ▲ D=273
 - ▲ E=00
 - ▲ F será um binário com o valor 42

Especificação de segmentos

- Cada segmento tem a seguinte especificação:
Valor:Tamanho/Lista_especificação_tipos
 - ▲ O único campo obrigatório é o tamanho
- A lista de especificação dos tipos é uma lista de tipos separada por «:»

Tipo	integer, float ou binary
Sinal	signed ou unsigned
<i>Endianness</i>	big ou little
<i>Unit</i>	tamanho de cada unidade que vai ser multiplicado por Tamanho para dar o tamanho real

Exemplos 1

Records

Listas

Macros

Bins

Bins

Exemplos

Especificação de
segmentos

Exemplos 1

Exemplos 2

Exemplos 3

Funs

- `X:4/little-signed-integer-unit:8`
um elemento com um tamanho de 32 bits, e que contém um inteiro com sinal no formato
- `<<X:1,Y:6>>`
dá erro, um binário tem no total um número certo de bytes
- `<<X:1,Y:6,Z:1>>`
já é aceite pelo compilador
- `<<X+1:8>>`
dá erro (problema do compilador)
- `<<(X+1):8>>`
é a forma correcta
- `<<"hello">>`
é equivalente a `<<$h,$e,$l,$l,$o>>`

Exemplos 2

- Como tirar o resto do binário:
`<<A:6,B:2,Resto/binary>>`
- Atenção que Resto tem de ter um número certo (inteiro) de bytes

Records

Listas

Macros

Bins

Bins

Exemplos

Especificação de
segmentos

Exemplos 1

Exemplos 2

Exemplos 3

Funs

Exemplos 3

```
-define(IP_VERSION, 4).
-define(IP_MIN_HDR_LEN, 5).

DgramSize = size(Dgram),
case Dgram of
  <<?IP_VERSION:4, HLen:4, SrvcType:8, TotLen:16,
  ID:16, Flgs:3, FragOff:13,
  TTL:8, Proto:8, HdrChkSum:16,
  SrcIP:32,
  DestIP:32, RestDgram/binary>>
  when HLen >= 5, 4*HLen =< DgramSize
    -> OptsLen = 4*(HLen - ?IP_MIN_HDR_LEN),
    <<Opts:OptsLen/binary,Data/binary>> = RestDgram,
    ...
end.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Funs

Objectos Funcionais

- Os objectos funcionais são um novo tipo de dados introduzido na versão 4.4 do Erlang.
- Podemos passar funs como argumentos a uma função.
- Podemos escrever funções que devolvem funs.
- As funs permitem encapsular padrões comuns de design em formas funcionais chamadas funções de ordem elevada
- Ficamos com programas mais curtos e mais claros

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Comentários



- Não é normal em linguagens imperativas
- Para um programador «normal» esta é a parte mais «confusa» das linguagens funcionais
- Permite uma grande reutilização do código
- Permite uma grande concisão na escrita
- Exige atenção na leitura dos programas
- Exige atenção para «redundâncias» que não podem ser aproveitadas noutras linguagens

- Records
- Listas
- Macros
- Bins
- Funs
- Objectos Funcionais
- Comentários
- map 1
- map 2
- Filosofia
- foreach 1
- foreach 2
- Vantagens
- Sintaxe
- Definição de funs
- Variáveis e funs
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- splitlist
- foldl
- mapfoldl 1
- mapfoldl 2
- Funs²

map 1

- Se quisermos escrever uma função que duplique o valor de cada um dos elementos de uma lista:

```
double([H|T]) -> [2*H|double(T)];  
double([]) -> []
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

map 1

- Se quisermos escrever uma função que duplique o valor de cada um dos elementos de uma lista:

```
double([H|T]) -> [2*H|double(T)];  
double([]) -> []
```

- Se quisermos uma função que some um a cada um dos elementos de uma lista:

```
add_one([H|T]) -> [H+1|add_one(T)];  
add_one([]) -> [] .
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

map 1

- Se quisermos escrever uma função que duplique o valor de cada um dos elementos de uma lista:

```
double([H|T]) -> [2*H|double(T)];  
double([]) -> []
```

- Se quisermos uma função que some um a cada um dos elementos de uma lista:

```
add_one([H|T]) -> [H+1|add_one(T)];  
add_one([]) -> [] .
```

- Estas funções possuem muito em comum...

map 2

- Estas funções possuem muito em comum, podemos escrever uma função map que exprime o comum nas duas funções:

```
map(F, [H|T]) -> [F(H) | map(F, T)];  
map(F, []) -> [].
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

map 2

- Estas funções possuem muito em comum, podemos escrever uma função map que exprime o comum nas duas funções:

```
map(F, [H|T]) -> [F(H) | map(F, T)];  
map(F, []) -> [].
```

- Podemos então reescrever as funções do seguinte modo:

```
double(L) -> map(fun(X) -> 2*X end, L).  
add_one(L) -> map(fun(X) -> 1 + X end, L).
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

map 2

- Estas funções possuem muito em comum, podemos escrever uma função map que exprime o comum nas duas funções:

```
map(F, [H|T]) -> [F(H) | map(F, T)];  
map(F, []) -> [].
```

- Podemos então reescrever as funções do seguinte modo:

```
double(L) -> map(fun(X) -> 2*X end, L).  
add_one(L) -> map(fun(X) -> 1 + X end, L).
```

- $\text{map}(F, L)$ é uma função que aceita uma função F e uma lista L como argumentos e devolve a nova lista que é obtida aplicando F a cada um dos elementos de L

- Records
- Listas
- Macros
- Bins
- Funs
- Objectos Funcionais
- Comentários
- map 1
- map 2**
- Filosofia
- foreach 1
- foreach 2
- Vantagens
- Sintaxe
- Definição de funs
- Variáveis e funs
- Exemplos 1
- Exemplos 2
- Exemplos 3
- Exemplos 4
- splitlist
- foldl
- mapfoldl 1
- mapfoldl 2
- Funs²

Filosofia

- O processo que nos permite «extrair» o que há de comum em diferentes programas chama-se «abstracção procedimental»
- Serve para escrever diferentes funções:
 1. Escrevendo uma função que representa as partes comuns
 2. Exprimindo as diferenças em termos de funções que são passadas como argumento

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

foreach 1

- Vamos imprimir todos os elementos de uma lista num *stream*:

```
print_list(Stream, [H|T]) ->
    io:format(Stream, "~p~n", [H]),
    print_list(Stream, T);
print_list(Stream, []) ->true.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

foreach 1

- Vamos imprimir todos os elementos de uma lista num *stream*:

```
print_list(Stream, [H|T]) ->
    io:format(Stream, "~p~n", [H]),
    print_list(Stream, T);
print_list(Stream, []) ->true.
```

- Vamos enviar uma mensagem a uma lista de processos:

```
broadcast(Msg, [Pid|Pids]) ->
    Pid ! Msg,
    broadcast(Msg, Pids);
broadcast(_, []) ->true.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

foreach 1

- Vamos imprimir todos os elementos de uma lista num *stream*:

```
print_list(Stream, [H|T]) ->
    io:format(Stream, "~p~n", [H]),
    print_list(Stream, T);
print_list(Stream, []) ->true.
```

- Vamos enviar uma mensagem a uma lista de processos:

```
broadcast(Msg, [Pid|Pids]) ->
    Pid ! Msg,
    broadcast(Msg, Pids);
broadcast(_, []) ->true.
```

- Nos dois casos anteriores a estrutura das funções é similar
- As duas funções usam cada um dos argumentos da lista para fazer «alguma coisa»
- Essa «alguma coisa» deve ser passada como argumento à função que processa a lista

foreach 2

- Com:

```
foreach(F, [H|T]) ->
    F(H),
    foreach(F, T);
foreach(F, []) -> ok.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

foreach 2

- Com:

```
foreach(F, [H|T]) ->
    F(H),
    foreach(F, T);
foreach(F, []) -> ok.
```

- Poderemos então escrever:

```
print_list(S,L)->
    foreach(fun(X) -> io:format(S,"~p~n",[X]) end, L).

broadcast(M,L)->foreach(fun(Pid) -> Pid ! M end, L).
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Vantagens

- O programa fica mais legível (se o soubermos ler)
- O programa fica mais curto
- A intenção do programador fica expressa no código (temos de perceber a intenção em vez das acções)
- As funções que aceitam funs como argumentos são muito mais fáceis de reutilizar

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Sintaxe

- A forma normal de escrever uma fun é:
$$F = \text{fun } (\text{Arg1}, \text{Arg2}, \dots, \text{argN}) \rightarrow \dots \text{end}$$

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Sintaxe

- A forma normal de escrever uma fun é:
`F = fun (Arg1, Arg2, ...argN) -> ...end`
- Se a função já está escrita no mesmo módulo podemos usar:
`F = fun Nomedafuncao/Aridade`

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Sintaxe

- A forma normal de escrever uma fun é:
`F = fun (Arg1, Arg2, ...argN) -> ...end`
- Se a função já está escrita no mesmo módulo podemos usar:
`F = fun NomeDaFuncao/Aridade`
- Se a função já está escrita noutra módulo²:
`F = fun Modulo:NomeDaFuncao/Aridade`

²Pode-se ver ainda `F= {modulo,funcao}` que é a versão primitiva que já não se deve utilizar

Definição de funs

```
-module(fun_test).  
-export([t1/0, t2/0, t3/0, double/1]).  
-import(lists, [map/2]).  
t1() -> map(fun(X) -> 2 * X end, [1,2,3,4,5]).  
t2() -> map(fun double/1, [1,2,3,4,5]).  
t3() -> map(fun ?MODULE:double/1, [1,2,3,4,5]).  
double(X) -> X * 2.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Variáveis e funs

- Todas as variáveis que ocorrem na cabeça de uma fun são variáveis *frescas*
- Variáveis que estejam definidas antes, e que ocorram em chamadas a funções ou *guards* possuem o valor que tinham antes da fun
- Não se pode exportar variáveis de uma fun
- Ver os manuais se houver problemas !

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 1

Exemplos tirados do módulo `lists`

■ `map`

```
map(F, [H|T]) -> [F(H) | map(F, T)];  
map(F, []) -> [].
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 1

Exemplos tirados do módulo `lists`

■ `map`

```
map(F, [H|T]) -> [F(H) | map(F, T)];  
map(F, []) -> [].
```

■ `any`

```
any(Pred, [H|T]) ->  
    case Pred(H) of  
    true -> true;  
    false -> any(Pred, T)  
    end;  
any(Pred, []) -> false.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 2

■ all

```
all(Pred, [H|T]) ->
    case Pred(H) of
        true -> all(Pred, T);
        false -> false
    end;
all(Pred, []) ->true.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 2

- all

```
all(Pred, [H|T]) ->  
    case Pred(H) of  
        true -> all(Pred, T);  
        false -> false  
    end;  
all(Pred, []) ->true.
```

- foreach

```
foreach(F, [H|T]) ->  
    F(H),  
    foreach(F, T);  
foreach(F, []) ->ok.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 3

■ filter

```
filter(F, [H|T]) ->
    case F(H) of
        true -> [H|filter(F, T)];
        false -> filter(F, T)
    end;
filter(F, []) -> [].
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 3

■ filter

```
filter(F, [H|T]) ->
    case F(H) of
        true -> [H|filter(F, T)];
        false -> filter(F, T)
    end;
filter(F, []) -> [].
```

■ takewhile

```
takewhile(Pred, [H|T]) ->
    case Pred(H) of
        true -> [H|takewhile(Pred, T)];
        false -> []
    end;
takewhile(Pred, []) -> [].
```

Exemplos 4

■ `dropwhile`

```
dropwhile(Pred, [H|T]) ->
  case Pred(H) of
    true -> dropwhile(Pred, T);
    false -> [H|T]
  end;
dropwhile(Pred, []) -> [] .
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

Exemplos 4

■ `dropwhile`

```
dropwhile(Pred, [H|T]) ->
  case Pred(H) of
    true -> dropwhile(Pred, T);
    false -> [H|T]
  end;
dropwhile(Pred, []) -> [] .
```

■ `first`

```
first(Pred, [H|T]) ->
  case Pred(H) of
    true -> {true, H};
    false -> first(Pred, T)
  end;
first(Pred, []) -> false.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

splitlist

```
splitlist(Pred, L) ->splitlist(Pred, L, []).  
splitlist(Pred, [H|T], L) ->  
    case Pred(H) of  
        true -> splitlist(Pred, T, [H|L]);  
        false -> {reverse(L), [H|T]}  
    end;  
splitlist(Pred, [], L) ->{reverse(L), []}.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

foldl

```
foldl(F, Accu, [Hd|Tail]) ->
                                foldl(F, F(Hd, Accu), Tail);
foldl(F, Accu, []) -> Accu.
```

- foldl aceita uma função de dois argumentos, uma lista e um acumulador
- a função deve devolver um acumulador que é usado de cada vez que a função é chamada

```
1> L = ["I", "like", "Erlang"].
["I", "like", "Erlang"]
2> lists:foldl(fun(X, Sum) -> length(X) + Sum end, 0, L).
11
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

mapfoldl 1

- Faz map e foldl ao mesmo tempo

```
mapfoldl(F, Accu0, [Hd|Tail]) ->
    {R,Accu1} = F(Hd, Accu0),
    {Rs,Accu2} = mapfoldl(F, Accu1, Tail),
    {[R|Rs], Accu2};
mapfoldl(F, Accu, []) -> {[], Accu}.
```

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²

mapfoldl 2

```
11 > Uppcase = fun(X) when $a = < X,  
                                X = < $z - > X + $A - $a;  
                                (X) - > X  
                                end.
```

```
#Fun < erl eval >
```

```
12 > Uppcase_word =fun(X)->lists:map(Uppcase, X)  
                                end.
```

```
#Fun < erl eval >
```

```
13 > Uppcase_word("Erlang").  
"ERLANG"
```

```
14 > lists:map(Uppcase_word, L).  
["I", "LIKE", "ERLANG"]
```

```
15 > lists:mapfoldl(fun(Word, Sum) - >  
15 > {Uppcase_word(Word), Sum + length(Word)}  
15 > end, 0, L).  
{["I", "LIKE", "ERLANG"], 11}
```

- Funs que devolvem Funs
- Funções de ordem elevada
- Maior expressividade
- Permitem «poupar» ainda mais código
- Exemplos de aplicações:
 - ▲ Listas infinitas
 - ▲ Parsers e gramáticas

Records

Listas

Macros

Bins

Funs

Objectos Funcionais

Comentários

map 1

map 2

Filosofia

foreach 1

foreach 2

Vantagens

Sintaxe

Definição de funs

Variáveis e funs

Exemplos 1

Exemplos 2

Exemplos 3

Exemplos 4

splitlist

foldl

mapfoldl 1

mapfoldl 2

Funs²