Llenguatges de Programació - FIB

Josep de Cid

Gener 2018

Índex

1 Introducció

Davant la necessitat de trobar una forma òptima de tractar concurrència, serveis distribuïts, tolerància a errors tot mantenint un llenguatge d'alt nivell per mantenir la simplicitat i alta productivitat, neix Erlang a finals dels anys 80.

Erlang és un llenguatge de propòsit general, funcional, amb avaluació estricta i tipatge dinàmic, que permet treballar concurrència, CPU multi-nucli, distribució en xarxa i tolerància al error. El seu principal objectiu és l'aplicació simple de les característiques anteriorment descrites en sistemes majoritàriament en temps real o de continua execució.

Sovint també veiem el nom d'erlang acompanyat d'OTP, o directament el llenguatge Erlang/OTP, però que és OTP exactament?

OTP són les sigles de *Open Telecom Platform*, el conjunt de biblioteques de suport estàndard per a aplicacions en Erlang. Conté estructures de dades bàsiques com ara llistes, arbres, diccionaris, conjunts; eines per servidors, on trobem cues de fitxers, sockets, servidors genèrics, tractament del temps; suport a cadenes i events, paquets matemàtics bàsics, sistemes de tractament d'errors i alertes, protocols com HTTP o SSL i un conjunt d'interfícies per integrar llenguatges forans com C/C++, Java o fins i tot més orientats a xarxes i sistemes, com servidors TCP.

En l'actualitat, Erlang s'utilitza en sistemes massius a gran escala d'empreses líders en el sector tecnològic, com és el cas dels sistemes de xat de **Whatsapp** i **Facebook Messenger**, distribuint el servei ofert a milions de persones simultàniament, en diferents servidors; altres grans companyies com **Yahoo!** amb el seu servei de marcadors Delicious, i **Amazon** que l'utilitza en serveis cloud d'AWS implementant SimpleDB; i empreses de telecomunicacions importants com **Motorola** i **Ericsson** que l'utilitzen en nodes GPRS i xarxes mòbils 3G arreu del món.

També podem trobar conegudes aplicacions de codi obert, com **CouchDB**, una base de dades orientada a documents sense fer servir esquema, que ofereix una gran escalabilitat a través de múltiples CPU i servidors, o **RabbitMQ**, implementació del protocol de missatgeria AMQP, estàndard emergent per a sistemes empresarials de missatgeria d'alt rendiment.

2 Origen i context històric

Durant la década dels 80, el software dels productes de la companyia Ericsson (Switches de telecomunicacions, sistemes de ràdio...) s'estaven tornant massa complexos, on C, C++, Pascal, EriPascal, assembler, PLEX i una vintena de llenguatges més s'utilitzaven a sistemes de producció i laboratoris de recerca. Fins al moment només hi havien mecanismes ad hoc per a manteniment, depuració i actualitzacions dels sistemes. Els enginyers d'Ericcson van pensar que hi havia d'haver una forma millor, mantenint un llenguatge d'alt nivell per augmentar la productivitat, que facilités tractar concurrència, distribució en xarxa i recuperació d'errors.

A principis de dècada, entre el 1982 i 1986, apareixen les primeres proves de concepte del llenguatge que seria anomenat Erlang, nom donat pel matemàtic danès pioner en l'estudi de xarxes de telecomunicacions, Agner Krarup Erlang, i també per la coincidència amb l'acrònim "**Er**iccson **lang**uage". Els primers experiments es realitzaren sobre llenguatges ja existents com Lisp, Prolog o Parlog, on finalment s'escolliria **Prolog** com l'intèrpret per les primeres versions d'Erlang, sent substituït posteriorment.

Al 1988 apareix el primer producte, una central telefònica PBX en utilitzar Erlang. Un any després s'experimenta amb una reescriptura del codi de l'interpret Prolog, conseguint així un nou intèrpret força millorat que aconseguí incrementar notablement l'eficiència.

Amb la dècada dels 90, arriben els primers articles d'investigació sobre Erlang, que s'expandeix a empreses com Bell (Bellcore). Al 1992 apareixen els primers ports a sistemes operatius estàndard, **PC** i **Macintosh**, i un any després s'afegeix la distribució en xarxa.

Al 1998, es decideix obrir Erlang a **codi obert** i a dia d'avui, s'utilitza tant en grans empreses com en una gran i creixent activitat de la comunitat open-source.

3 Característiques del llenguatge

Quan parlem d'Erlang ens podem estar referint a dues coses diferents (almenys en el camp dels llenguatges de programació) alhora que similars. El primer és el llenguatge en si, i per altra banda tenim el sistema d'execució (o *runtime*), conegut per les característiques de ser distribuït, tolerant a l'error... entre molts altres, com veurem a continuació.

3.1 Paradigmes

El llenguatge de programació Erlang es caracteritza per ser **immutable**, on l'estat d'un objecte ja creat no es pot modificar; fer ús del **Pattern matching**, consistent en comparar una seqüència de *tokens* fins trobar el primer coincident i executar l'acció pertinent; i per tant, ser un llenguatge de **programació funcional**, caracteritzat per ser tractat tot com a funcions.

La part seqüencial del llenguatge compta també amb sistema d'avaluació ansiosa (**eager evaluation**), **assignació simple** i **tipatge dinàmic**.

Si en canvi, parlem del sistema *runtime*, podem dir que està preparat per a sistemes **distribuïts**, amb necessitat de **tolerància a l'error**, en temps real, alta disponibilitat (mai s'ha d'aturar), i que faciliti fer *hot swapping*, és a dir, poder actualitzar codi a producció sense haver de parar el sistema.

Per tant podem concloure que Erlang és multi-paradigma: llenguatge de propòsit general, concurrent, funcional i amb un sistema *runtime* que compta amb *garbage collector*.

3.2 Compilació/Interpretació

Erlang és un llenguatge situat en la frontera entre els llenguatges compilats i interpretats. El codi, es compila a **bytecode**, per posteriorment ser executat per la màquina virtual d'erlang: **BEAM**, acrònim de "Bogdan/Björn's Erlang Abstract Machine". BEAM carrega el bytecode, el qual és convertit a codi enfilat (programació mitjançant subrutines) en temps de càrrega.

També inclou un compilador natiu per a la majoria de plataformes desenvolupat al projecte HiPe, de la universitat de Uppsala, i suporta interpretació directament des del codi font, mitjançant un AST (Arbre de Sintaxis Abstracta).

3.3 Sistema de tipus

El tipatge d'Erlang és bàsicament dinàmic, la qual cosa significa que la comprovació de tipatge es realitza durant l'execució enlloc de durant la compilació, a l'aigual que en llenguatges interpretats com Ruby o Python; i fortament tipat, el que significa que una variable d'un tipus concret, no es pot utilitzar com si fos d'un altre tipus, excepte si fem una conversió explícita.

Erlang té vuit tipus diferents de dades primitives:

- **Integers:** L'aritmètica d'enters és exacta i només ve limitada per la memòria del sistema, el que es coneix com precisió arbitrària: `-15012018`
- **Floats:** Nombres en coma flotant segons l'especificació IEEE 754 representats amb 64 bits: `19.96`
- **Atoms:** Utilitzats per denotar valors importants com ara constants, per exemple les paraules reservades del llenguatge *and*, *orelse*, *try*, entre d'altres. Els àtoms no són eliminats pel *Garbage Collector* i per tant s'acumulen fins que el sistema està ple, ja sigui per falta de memòria o per assolir el límit, 1048577, i per tant, no s'han de definir dinàmicament. El programador també pot declarar-ne de propis, en forma de cadena alfanumèrica, començant sempre en minúscula: `josepdecid`
- **Pids:** Identificadors de processos d'Erlang creats per la primitiva del llenguatge `spawn(...)`.
- **Funs:** Clausures de funcions de la forma `fun(...) -> ... end`
- **References:** Símbols globals únics, l'única propietat dels quals, és poder ser comparats per igualtat. És creen cridant a `make_ref()`.
- **Binaries:** Seqüència de bytes, per assolir una forma eficient en espai d'emmagatzemar dades i accedir-hi pel canal d'entrada/sortida.
- **Ports:** Ofereixen comunicació amb el món exterior, permeten intercanviar missatges, obeint un protocol. Són creats amb el mètode `open_port(...)`.

Per sobre, englobant les anteriors, en trobem tres de compostes:

- **Tuples:** Contenidors per a un nombre fixat de tipus de dades, ja siguin primitives o compostes, de forma que l'accés a un element es realitza en temps constant. Es representen de la forma $\{D_1, D_2, ..., D_n\}$.
- **Lists:** Contenen un nombre variable de tipus de dades. Expressades amb la sintaxis $[H|T]$, on H, el primer element també anomenat cap, és accessible en temps constant; i T, la cua és la resta. La simplificació de la representació d'una llista és amb la següent sintaxis: $[D_1, D_2, ..., D_n]$.
- **Maps:** Contenidors que poden emmagatzemar un nombre variable d'associacions clau-valor. $\{K_1 => V_1, ..., K_n => V_n\}$.

Aquestes estructures de conjunts també ens ofereixen dues estructures considerades "sucre sintàctic", només per facilitar-ne la programació i tractament. Aquestes són els **Strings**, per representar llistes de caràcters; i els **Records**, una forma d'associar etiquetes als elements d'una tupla per no accedir-hi per posició.

Per defecte, Erlang no ofereix cap forma de definir classes, encara que hi ha mòduls externs que ho permeten.

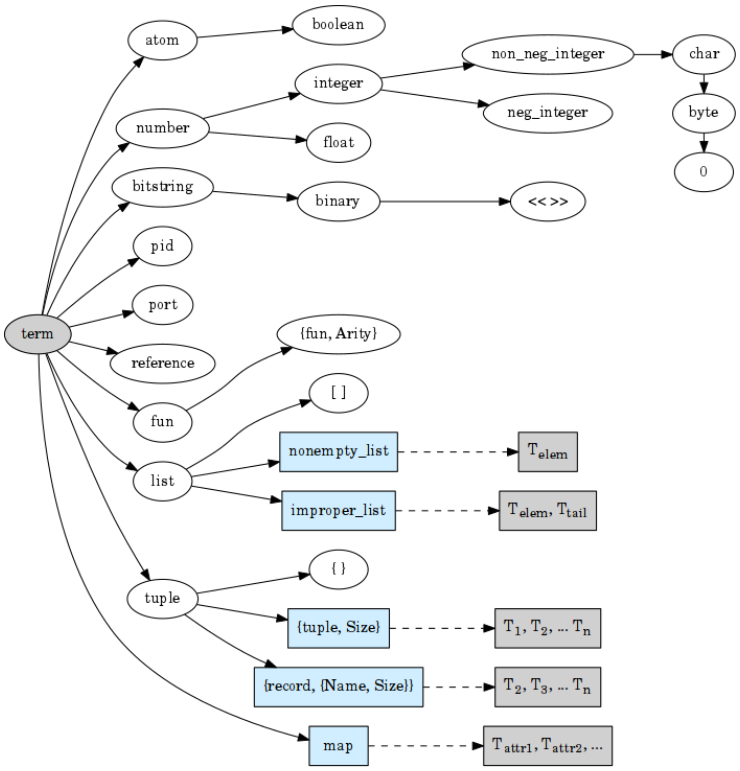

Figure 1: Jerarquia de tipus a Erlang

3.4 Llenguatges similars

El primer llenguatge que ens ve a la ment a l'observar Erlang, és **Prolog**, ja que va ser el primer llenguatge en conformar la base de l'intèrpret, i es va basar en la seva sintaxis, com veurem clarament en els diferents exemples de la següent secció, concretament en la definició de funcions a fitxers per *pattern matching*, l'ús del punt per finalitzar una seqüència, implementació de llistes amb la sintaxis $[Cap|Cua]$... entre molts altres que podrem notar si estem familiaritzats amb l'entorn Prolog. També està inspirat en llenguatges similars, com Smalltalk, LISP, o el seu predecessor a Ericsson, PLEX.

En quan a la seva principal característica, el suport per a concurrència, el llenguatge més similar a Erlang conceptualment és **occam**, llenguatge imperatiu procedimental usat en processos de comunicació seqüencial (CSP).

4 Exemples

A l'igual que prolog amb els seus fitxers carregables *.pl, anomenats bases de dades del llenguatge, amb erlang tenim el mateix sistema amb *.erl, de forma que definint un mòdul, les funcions a exportar i les implementacions d'aquestes, podem carregar-los en una sessió de l'intèrpret.

4.1 Seqüencialitat

Per familiaritzar-nos amb el llenguatge, comencem amb uns exemples simples seqüencials. A part de les implementacions, que no tenen gaire a destacar, veurem que s'ha de definir el nom del mòdul, que serà el que s'utilitzarà per importar-lo, com veurem a continuació. També necessitem exportar les funcions indicant l'aritat, cosa que ens permet definir funcions diferents amb el mateix nom, depenent dels paràmetres, molt útil en algunes situacions.

A continuació tenim un exemple bàsic de base de dades, amb un parell de funcions per tractar llistes: afegir element i ordenar.

```
-module(listLib).
-export([append/2]).
-export([sort/1]).

% Appends two lists
append([],    E) -> E;
append([H|T], E) -> [H|append(T, E)].

% Sorts a list asc
sort([]) -> [];
sort([Pivot | T]) ->
            sort([X || X <- T, X < Pivot]) ++
            [Pivot] ++
            sort([X || X <- T, X >= Pivot]).
```

Posteriorment, entrant a la consola `erl`, només ens cal compilar el fitxer que conté el mòdul a carregar, amb el mètode abreviat **c** (de compilació), i ja podrem fer us de les funcions, posant el nom del mòdul com a prefix. També podem veure un exemple de funció definida directament a l'intèrpret:

```
1> c(listLib).
{ok,listLib}
2> listLib:append([2,0,1], [8]).
[2,0,1,8]
3> listLib:append("Jose", "p").
"Josep"
4> listLib:sort([1,2,1,5,2,3,7,4,-2]).
[-2,1,1,2,2,3,4,5,7]
5> % Adder
5> Adder = fun(X) -> fun(Y) -> X + Y end end.
#Fun<erl_eval.6.99386804>
6> % Bind Fun
6> G = Adder(10).
#Fun<erl_eval.6.99386804>
7> % Fun result
7> G(5).
15
```

4.2 Concurrència

Passem a veure ara un exemple concurrent, per crear, rebre i enviar mitjançant processos, mitjançant els PIDs. En aquest cas veurem l'ús de funcions anònimes, la comunicació entre processos, i la facilitat per tractar events temporals, com un *timeout* en el nostre cas.

```
% Spawn PID
Pid = spawn(fun() -> loop(0) end)

% Send message
Pid ! Message,
...
% Receive message
receive
    Message1 ->
        Actions1;
    Message2 ->
        Actions2;
    ...
    after Time ->
        TimeOutActions
end
```

4.3 Distributivitat

En aquest exemple distribuït veiem com ens connectaríem a un node remot, mitjançant el mètode *connect_node*, del mòdul *net_kernel* amb un paràmetre del tipus *node()*. Posteriorment, creem dos processos identificats amb Pid1 i Pid2, creats al node especificat.

L'assignació a true al connectar el node i comprovar que el procés està viu, serveix per, en cas que la funció no retorni el mateix, l'execució no segueix fins arribar al final de la seqüència, al punt, saltant-se totes les instruccions entre comes fins al primer punt, que només s'avaluen si l'anterior retorna un valor no fals.

```
...
true = net_kernel:connect_node(Node),

Pid1 = spawn(Node, Fun),
Pid2 = spawn(Node, Module, Func, Argv),

true = is_process_alive(Pid1),
...
```

4.4 Control d'errors

Aquí podem veure un exemple de throw/catch, on definint una funció que pot llençar un error, al cridar-la, la posem en un bloc catch, en el que podem diferenciar casos i indicar-ne el comportament amb el que respondrà cadascun.

Veiem que llençar excepcions amb estructures complexes i no només variables simples o missatges d'error.

```
...
case (catch foo(A, B)) of
    {abnormal_case1, Y} ->
        ...
    {'EXIT', Opps} ->
        ...
    Val ->
        ...
end,

...

foo(A, B) ->
    ...
    throw({abnormal_case1, ...})
```

5 Fonts d'informació

La major part del contingut està basat en la documentació dels propis autors o actuals mantenidors del llenguatge i la seva documentació, a partir de la seva pàgina web oficial [1], com la documentació de funcions per als exemples i informació sobre el llenguatge.

Els documents web, han servit principalment, per veure exemples de codi i aprendre'n el funcionament [5].

També s'han consultat altres pàgines com Wikipedia [2], HiPE [3] o WhatIs [4], amb informació variada sobre el llenguatge, com els orígens, evolució, o alguns exemples.

Detalls més específics sobre l'ús del llenguatge i característiques concretes s'han obtingut a través de llibres d'Erlang [6][8][7]

Personalment, a l'estar basat en informació oficial majoritàriament creada per contribuïdors d'Erlang, o documentació àmpliament acceptada per la majoria de la comunitat. Al ser un llenguatge relativament poc conegut no hi han gaires fonts d'informació ni recursos en comparació a llenguatges universalment coneguts, com C++, Python, JS, PHP... i encara menys que puguin competir amb la pàgina oficial.

6 Referències

[1] Erlang Programming language: Documentation,
`https://www.erlang.org/docs`

[2] Erlang (programming language). *Wikipedia*, December 30, 2017.
`https://en.wikipedia.org/wiki/Erlang_(programming_language)`

[3] High-Performance Erlang Project. *HiPE*, January 2, 2018.
`http://www.it.uu.se/research/group/hipe/`

[4] Erlang Programming Language. *WhatIs*, January 3, 2017.
`http://whatis.techtarget.com/definition/Erlang-programming-language`

[5] Erlang Tutorial. *TutorialsPoint*, January 3, 2017.
`https://www.tutorialspoint.com/erlang/`

[6] Hebert, Fred. *Learn You Some Erlang for Great Good!*.
San Francisco: No Starch Press; January 13, 2013.

[7] Armstrong, Joe. *Making reliable distributed systems in the presence of software errors*. Ph.D. Dissertation. The Royal Institute of Technology, Stockholm, Sweden; November 20, 2003.

[8] Cesarini, Francisco; Thomson, Simon. *Erlang programming*.
Sebastopol: O'Reilly Media; June 2009.