Erlang

MESSAGE ORIENTED PROGRAMMING

Alcune risorse

erlang.org e erlang.org/doc.html
- Documentazione, tutorial, esempi, articoli, libri, guide …

learnyousomeerlang.com/content

Introducing Erlang (O'Reily)

Programming Erlang, (The pragmatic bookshelf)
- In biblioteca

Erlang Programming (O'Reilly)
- *http://en.wikibooks.org/wiki/Erlang_Programming*

Designing for Scalability with Erlang/OTP (O'Reilly)

Cenni storici

1982-85 Telco interessate a nuovi strumenti per programmare la loro infrastruttura

1985-87 sperimentazione con Prolog, Lisp, Parlog etc
- Il back-tracking è un problema
- Servono primitive per la concorrenza e la gestione degli errori
- Ok astrarre, ma bisogna gestire dati «low-level» in modo efficiente

1987 Primi prototipi Erlang (VM in prolog)

1987-90 Usato in prototipi (Erlang Vs C)

1991 VM in C e release al pubblico

1992 Uso in produzione

1993 Primitive di distribuzione aggiunte a VM

1996 OTP framework

Erlang/OTP

Linguaggio funzionale

Alto livello

VM

Soft real-time

Concorrenza (actor-model)

Scalabilità

Robustezza

Distribuzione trasparente

Interoperabilità (Drivers, FLI etc)

(Security…hem)

Casi di studio

Servizi di messaggistica
- Facebook
- Ejabber
- RabbitMQ
- GPRS/3G/4G (Ericson, Motorola, T-Mobile)

Storage/DB
- SimpleDB (Amazon)
- CouchDB
- Scalaris

WEB
- MochiWEB
- YAWS

Erlang on XEN

Strumenti

Qualsiasi editor + riga di comando

emake/edoc

typer/dialyzer

IDEs
- Emacs + Erlwere-mode + Distel
- Vim + vimerl
- NetBeans + ErlyBird
- Eclipse + ErlIDE

Elixir (linguaggio, target Erlang VM, embeds Erlang)

Tipi base

(Esempio schell)

Interi

Float

Atomi
- *A la* prolog

Reference

Tuple

Liste

Funzioni

BitString/Binary

Funzioni

E.g.
- `times(X,Y) -> X * Y.`
- ```
  rept([X,X|_]) -> true;
  rept([_|Y]) -> rept(Y);
  rept([]) -> false.
  ```
- `even(I) when I rem 2 == 0 -> true;`
- `even(I) -> false;`

```
name pattern1 when guard1 -> body;
name pattern2 when guard2 -> body;
name patternN when guardN -> body.
```

Match valutato secondo l'ordine

Guardie senza side effects

Apply

Moduli

Le funzioni sono organizzate in moduli

Struttura:
- Attributi
  - -module(name)
  - -export(funzioni esportate)
  - -vsn(version number)
  - Etc.
- Include
  - -inlude("file.hrl").
  - -include_lib("mnesia").
- Dichiarazioni macro
  - -define(MACRO_NAME,SUBSTITUTION).
  - -define(TIMEOUT, 10000).
  - -define(DBG(Pattern, Args), io:format(Pattern, Args)).
- Dichiarazioni record
- Dichiarazioni di funzioni
```

Record

-record(name, {filed1 [= default value1], …}).
- Eg: -record(person,{name,age,flag = true}).

#record-name{field-name = value, …}
- E.g. #person{name = "Mario", age = 37}.

RecordExpression#Record-name.field-name
- E.g. PersonVar#person.age

E.g.
- onBirthday(#person{age = Age} = P) -> P#person{age = Age + 1}.

Controllo di flusso

```
If
    booleanExpr -> body;
    booleanExpr -> body
end
```

```
case Expression of
    pattern [when guard] -> body;
    pattern [when guard] -> body
end
```

Evitare i «catch-all» e la programmazione difensiva

Errori ed eccezioni

```
Try Expression of
        Pattern [when guard] -> body
        …
Catch
        [class:]exceptionPattern [when guard] -> body
end
```

Class:
- Exit
- Error
- Throw

Annotazioni di tipo

Erlang non ha un sistema di tipi statico

Esistono tool per inferire/verificare specifiche di tipo

La direttiva `-type name :: typeExpression` definisce sinonimi di tipo

- `-type hash() :: <<_:160>>`
- `-type non_neg_integer() :: 0..`
- `-type nonempty_string() :: [char(),…]`
- `-type timeout() :: 'infinity' | non_neg_integer()`

La direttiva `-spec [Module:]Function(typeExprs) -> typeExpr` definisce il prototipo di una funzione

Typer

Typer inferisce/verifica annotazioni di tipo

typer <files> e typer -r <dir> visualizzano le annotazioni di tipo
E.g. typer echo.erl

```
%% File: "echo.erl"
%% ----------------
-spec start() -> 'true'.
-spec loop() -> 'ok'.
-spec test(_) -> {'error','timeout'} | {'ok',_}.
```

typer --show-exported considera solo le funzioni esportate
typer --edoc usa il formato edoc per l'otput (@spec…)
typer --annotate e typer –annotate-inc

Dialyzer

Ricerca:

- Errori di tipo
- Violazioni di API
- Codice morto o irraggiungibile
- Violazioni di opacità
- Race conditions
- Deadlocks e violazioni di protocollo (sperimentale)

PLT: analisi di librerie riutilizzabile

- Generare una PLT e.g.
  - dialyzer --build_plt --apps erts kernel stdlib
- Aggiornare una PLT e.g.
  - dialyzer --add_to_plt --apps crypto

Strumenti utili

Dialyzer
- Verifica (ed estrazione) di specifiche di tipo, deadcode detection, …

Typer
- Ispezione e inferenza di specifiche di tipo

Tidier
- Automatic refactoring e pulizia sorgenti

PropEr
- Verifica semi-automatica di proprietà mediante test-unit

CED
- Verifica (testing) concorrente (e.g. deadlocks)

Soter
- Interpretazione astratta

Concorrenza

FRAMMENTO CONCORRENZIALE

Attori

Processi leggeri

Nessuna memoria condivisa

Passaggio di messaggi asincrono

Ogni processo ha una *mailbox* associata
- Ordinata per ricezione
- Non ci sono limiti di dimensione o tipo di messaggi
- Ogni valore può essere spedito
- (I record sono tuple, attenzione)

I messaggi sono ricevuti quando vengono estratti dalla mailbox
- Estrazione mediante pattern
- Secondo l'ordine di ricezione

Receive

```
receive
      pattern [when guard] -> body;
      pattern [when guard] -> body
after timeout -> body
end
```

La clausola after è opzionale; ometterla equivale a `after infinity`

```
clear_mailbox() ->
    receive
        _ ->
            clear_mailbox()
    after 0 ->
        ok
    end.
```


Spawn, Link e Monitor

spawn(Module,Function,List of args)
- Crea un nuovo processo che esegue Module:Function(Args)

exit(Reason) = exit(Pid, Reason)
- Invia il segnale di terminazione {'EXIT', From, Reason}
- Attivare/disattivare la cattura: process_flag(trap_exit, Boolean)
- Reson = kill non può essere catturata
- Reason = normal, Pid non termina

link(Pid or Port)
- Collega due processi di modo che la terminazione di uno si propaghi all'altro
- E.g. supervisor (trap_exit = true) e worker (trap_exit = false)

spawn_link è la versione atomica di spawn and link

Monitor

erlang:monitor(process/node, Pid/Node) -> MonitorRef
- Monitora un processo (anche remoto)
- Il chiamante riceve {'DOWN', MonitorRef, Type, Object, Info} se il runtime non è in grado di contattare il processo/nodo monitorato
- Info = noproc | noconnection

erlang:demonitor(MonitorReference, [flush])
- Interrompe un monitoraggio
- Elimina eventuali messaggi dalla cosa se è fornita l'opzione flush

Live upgrade

Esempio

Falsi miti

I processi hanno costo zero
- Erlang usa processi «leggeri», ma comunque impongono un costo
- Usare processi con «grana troppo fine» è controproducente

I deadlock sono impossibili
- Ricezione bloccante

Scalabilità lineare nel numero di core
- Costi di scheduling, comunicazione

Esercitazione

Mutua esclusione

Algoritmi visti:
- Server centrale,
- anello, voto

Implementare mutex in Erlang
- Ovviamente li trovate nella libreria standard «global»

Non c'è condivisione di memoria

Servizio offerto da un processo mutex:
- start/stop
- wait/notify

Per casa:
- Fallimenti, priorità (starvation!)
- Mutex manager & deadlock detection centralizzata
- Mutua esclusione decentralizzata

Variabili mutabili

Libreria per variabili mutabili (e condivisibili) in Erlang

API:
- Creazione
- Lettura/scrittura
- Distruzione

I processi non condividono memoria

I nomi di variabile possono esser legati al più una volta

Non ricorrere a ETS/DETS/Mnesia o simili

Variabili mutabili(2)

Estendere l'API:
- Lock/Unlock

L'aggiornamento a caldo può dare problemi?

Accesso/condivisione «remoto»
- Accedere a variabili su altri nodi
  - (tip: registrare processi)
- Fallimenti?
- Risoluzione/ricerca?

Per casa:
- Replicazione
- Storage Key/Value
- Persistenza su disco

Event manager

Pubblicazione di eventi

Registrazione dinamica di «event handlers»
- e.g. debug, log, trace, health monitoring
- Handler possono avere uno stato
  - Sono callbacks: lo stato deve essere mantenuto e fornito dal manager
  - Sono processi: si fanno carico di mantenere il proprio stato
  - Soluzione ibrida: callbacks mascherano l'interazione con i processi
- E.g. un handler può instradare eventi ad altri manager

API: add_handler, delete_handler, publish_event (start, stop)

Per casa:
- Aggiungere metadati agli eventi (key->value)
- Policy di distribuzione e.g. all, any, first, round robin etc.
- Spawn Vs callback

Elezione

Studiare/testare algoritmi di elezione con Erlang

Realizzare un gruppo di processi
- Localmente
- Su più nodi

Elezione mediante
- Bully
- Ring
- (soddisfiamo le ipotesi sul modello di sistema?)

Per casa:
- Reti ad-hoc
- P2P/sistemi su larga scala
  - Chiavi speciali
  - Token mobili