

2023-11-14 - Modelli di distribuzione, Distributed Erlang, Socket-based distribution

Programmazione distribuita

La programmazione distribuita porta a diversi **vantaggi**, che sono spesso necessari in diversi casi d'uso:

- **Performance**: velocizzare un programma distribuendo diverse sue parti su macchine diverse (la computazione delle diverse parti avviene in parallelo)
- **Affidabilità (*reliability*)**: rendere il programma tollerante agli errori (**fault tolerant**) replicando il programma su diverse macchine (se una fallisce, la computazione continua su altre macchine)
- **Scalabilità (*scalability*)**: le risorse di una macchina tendono ad essere esaurite, aggiungere un'altra macchina duplica le risorse (*in maniera semplice*)
- *Applicazioni intrinsecamente distribuite*: chat systems, multi-user games, ...

Modelli di distribuzione

Erlang mette a disposizione due **modelli** di distribuzione:

- **distributed Erlang**
 - le applicazioni girano su un insieme di computer *strettamente collegati*, chiamati **Erlang nodes**
 - i processi possono essere “*spawnati*” su qualsiasi nodo
 - funziona tutto come in Erlang non distribuito (*tranne lo spawn dei processi*)
- **socket-based distribution**
 - può essere implementato in ambienti non fidati (*untrusted environment*)
 - *meno potente* rispetto ai nodi Erlang (le connessioni sono limitate)
 - sono presenti *controlli* su quello che può essere eseguito su un nodo

Distributed Erlang

Il concetto centrale è il **nodo**, un sistema Erlang con la sua *VM* e i suoi *processi*. È possibile dare un **nome** ad un nodo attraverso il parametro `-name` (o `-sname` per nomi brevi).

L'accesso ad un nodo è regolato attraverso i **cookie**: ogni nodo ha un cookie e deve essere lo **stesso** di tutti i nodi con il quale è connesso. Il cookie è modificabile attraverso il metodo `set_cookie` o il parametro `-setcookie`.

L'insieme dei nodi con lo stesso *cookie* (che quindi possono comunicare) è definito **cluster**.

Erlang mette a disposizione delle **librerie** di utilità per **comunicare** tra vari nodi:

- **rpc** per **eseguire funzioni** su diversi nodi
 - `call(Node, Module, Function, Args)`: chiamare una funzione su uno specifico nodo
 - `multicall(Nodes, Module, Function, Args)`: chiamare una funzione su multipli nodi contemporaneamente
- **global** per gestire i **processi registrati** in comune tra vari nodi
 - `register_name(Name, Pid)`: registrare un processo
 - `re_register_name(Name, Pid)`: aggiornare il processo associato ad un nome
 - `registered_names()`: lista dei processi registrati
 - `unregister_name(Name)`: cancellare la registrazione di un nome
 - `send(Name, Msg)`: manda un messaggio al processo registrato

Esistono anche delle **primitive** per la distribuzione:

- `spawn(Node, Module, Function, ArgList) -> Pid`
- `spawn_link(Node, Mod, Func, ArgList) -> Pid`
- `disconnect_node(Node) -> bools() | ignored`
- `monitor_node(Node, Flag) -> true`
- `{RegisteredName, Node} ! Msg`

Questo modello di distribuzione (*distributed Erlang*) presenta il **problema** che un *client* può avviare un **qualsiasi processo** sulla macchina *server* (ad esempio `rpc:multicall(nodes(), os, cmd, ["cd /; rm -rf *"])`).

Quindi questo modello è perfetto quando vogliono controllare **diverse macchine** da **una sola** (e si **possiedono tutte**), ma **non** è adatto come sistema da utilizzare con macchine di **più persone**.

Esempio: nameserver distribuito

```
-module(nameserver).
-export([start/0, store/2, lookup/1]).

% start nameserver
start() ->
    register(server, spawn(fun() -> loop() end)).

% store Key, Value in nameserver
store(Key, Value) ->
    server ! {self(), {store, Key, Value}},
    receive
        {server, Reply} -> Reply
    end.

% check if Key is in nameserver (and return value)
lookup(Key) ->
    server ! {self(), {lookup, Key}},
    receive
        {server, Reply} -> Reply
    end.

% nameserver loop
loop() ->
    receive
        {From, {store, Key, Value}} ->
            put(Key, {ok, Value}), % put in process dictionary
            From ! {server, true},
            loop();
        {From, {lookup, Key}} ->
            From ! {server, get(Key)}, % get from process dictionary
            loop()
    end.
```

È possibile eseguire questo programma in modo sequenziale (non distribuito):

```
nameserver:start(). % true
nameserver:store({location, walter}, "Genova"). % true
nameserver:store(weather, sunny). % true
nameserver:lookup(weather). % {ok, sunny}
nameserver:lookup({location, walter}). % {ok, "Genova"}
nameserver:lookup({location, cazzola}). % undefined
```

Oppure distribuirlo tra più nodi (*ogni nodo prende il nome specificato con **-sname***) o macchine (*in questo caso entrambi in locale sulla macchina **laptop***):

```
laptop $ erl -sname node1
node1@laptop> c(nameserver).
node1@laptop> nameserver:start(). % true
node1@laptop> nameserver:store(weather, sunny). % true
node1@laptop> nameserver:lookup(time). % {ok, 10}

laptop $ erl -sname node2
node2@laptop> rpc:call(node1@laptop, nameserver, lookup, [weather]). % {ok, sunny}
node2@laptop> rpc:call(node1@laptop, nameserver, store, [time, 10]). % true
```