Erlang!

Łukasz Cieśnik lukasz.ciesnik@gmail.com

Politechnika Poznańska

2009-10-29

Plan prezentacji

- Geneza języka
- Programowanie sekwencyjne
- Współbieżność w Erlangu
- Rozpraszanie
- Obsługa błędów
- Mnesia

Geneza

Początek w 1986 w firmie Ericsson. Napisany przez Joe Armstronga na potrzeby oprogramowania telekomunikacyjnego:

- *soft real time systems* — wymagany jest niski czas odpowiedzi, ale bez sztywnych ograniczeń czasowych,
- działające nieprzerwanie (*non-stop systems*),
- obsługujące asynchroniczne zdarzenia, najłatwiej modelowane przy pomocy procesów
- duże (złożone) systemy
- skalowalne, wymagające rozpraszania
- odporne na błędy (*robust*)

Cechy

- paradygmat funkcyjny:
  - funkcje (w tym domknięcia) to *first-order objects*,
  - ograniczenie efektów ubocznych,
- zachłanne obliczanie,
- wbudowane silne wsparcie dla współbieżnie wykonywanych procesów komunikacja na zasadzie przesyłania komunikatów,
- rozproszone przetwarzanie,
- obsługa błędów,
- ładowanie kodu w działającym systemie (*hot code loading*) bez naruszania działających procesów,
- komunikacja ze światem zewnętrznym.

Programowanie funkcyjne w Erlangu

W Erlangu same obiekty (*term*) są niemodyfikowalne, ale istnieją efekty uboczne:

- tworzenie procesów
- wysyłanie, odbieranie wiadomości
- obsługa wejścia/wyjścia

Zachłanne wartościowanie i obecność jawnej komunikacji lepiej służy modelowaniu procesów komunikujących się ze światem zewnętrznym i obsługujących asynchronicznych zdarzenia.

Typy danych

- liczby (całkowite, zmiennoprzecinkowe),
- atomy (symbole),
- listy,

```
[1, 2, 3] == [1 | [2 | [3 | []]]]
```

- krotki,

```
{1, 2, 3}
```

- brak osobnego typu łańcuchowego — tekst reprezentowany jest przez listę kodów znaków,
- rekordy (implementowane jako krotki),

```
-record(person, {firstname, lastname}).
#person{firstname="John", lastname="Kowalski"}
    == {person, "John", "Kowalski"}
```

- funkcje (przeciążane ze względu na ilość argumentów),
- identyfikatory procesów.

Składnia

Przykładowy kod:

```
-module(factor).
-export([factor/1]).

factor(N) when N > 0 ->
    factor(N, 2).

factor(N, Div) when N rem Div == 0 ->
    [Div | factor(N div Div, Div)];
factor(N, Div) when Div*Div =< N ->
    factor(N, Div+1);
factor(N, _) when N > 1 -> [N];
factor(_, _) -> [].
```

Procesy

Procesy w ramach maszyny wirtualnej:

- tanie w tworzeniu,
- niski narzut przełączania kontekstu,
- brak ograniczenia na ilość procesów,

Utworzenie nowego procesu:

```
Pid = spawn(Module, Function, Arguments)
```

Identyfikator procesu: `self()`
Wysłanie wiadomości: `Pid ! Message`

Ping: serwer

```
echo() ->
    receive
        {echo_request, From, Seq} ->
            From ! {echo_reply, self(), Seq},
            echo();
        stop ->
            ok
    end.
```

Ping: wysyłanie

```
ping_send(_, Receiver, Stop, Stop, _) ->
    Receiver ! stop;
ping_send(Peer, Receiver, Seq, Stop, Interval) ->
    Peer ! {echo_request, Receiver, Seq},
    receive
    after Interval ->
        ping_send(Peer, Receiver, Seq+1, Stop, Interval)
    end.
```

Ping: odbiór

```
ping(Peer, N, Interval) ->
    Receiver = self(),
    spawn(fun() ->
            ping_send(Peer, Receiver, 0, N, Interval)
        end),
    ping_recv(Peer).

ping_recv(Peer) ->
    receive
        {echo_reply, From, Seq} ->
            io:format("reply from ~w seq=~w\n",
                      [From, Seq]),
            ping_recv(Peer);
        stop ->
            ok
    end.
```

Ping: wywołanie

```
-module(ping).
-export([start/0, echo/0, ping/3]).

start() ->
    Echo_server = spawn(fun echo/0),
    ping(Echo_server, 3, 500),
    Echo_server ! stop.
```

Rozpraszanie

Dzięki komunikacji poprzez wymianę pakietów wsparcie dla rozproszonych systemów jest naturalne.
Można bez większych zmian program działający lokalnie uruchomić na wielu węzłach, ale trzeba określić lokalizację procesów na etapie ich tworzenia — częściowa przezroczystość lokalizacji.

Odporność na awarie

Erlang wspiera tworzenie systemów odpornych na błędy programistyczne czy awarie sprzętu:

- Zapewnia izolację procesów, łatwy podział systemów na niezależne moduły i warstwy.
- Mechanizmy wykrywania błędów, monitorowania procesów.
- Możliwość powtórzenia przetwarzania.
- Możliwość poprawienia kodu w działającym systemie (bez restartowania).

Monitorowanie procesów

Procesy można organizować w graf nieskierowany połączeń ustanawianych przez wywołanie `link(Pid)`.
Proces przy zakończeniu (pomyślnym lub nie) wysyła informację o tym do procesów z nim połączonych. Domyślnym zachowaniem procesów przy dostarczeniu wiadomości o pozytywnym zakończeniu jest jej ignorowanie, a przy błędnym — zakończenie przetwarzania i dalsze rozpropagowanie błędu.
Po wywołaniu `process_flag(trap_exit, true)` proces będzie odbierał informację o identyfikatorach zakończonych procesów oraz przyczynach zakończenia.

Mnesia

Mnesia to obiektowo-relacyjna baza danych.

- Szybki czas dostępu — baza działa w tej samej maszynie wirtualnej co aplikacja.
- Język zapytań zrealizowany na bazie składni Erlanga.
- Wartościami atrybutów mogą być dowolne struktury danych.
- Tranzakcje mogą być rozproszone i zagnieżdżone.
- Operacje krytyczne pod względem czasu wykonania można realizować z obejściem tranzakcyjności.
- Tabele mogą być replikowane w pamięci i/lub na dysku wielu węzłów. Gwarantuje to mniejszy czas dostępu oraz odporność na awarię — można operować na tabeli tak długo jak dostępna jest jedna z replik.
- Zapytania są niezależne od lokalizacji tabeli.

Zapytania 1

```
-record(person, {id, first_name, last_name, sex}).
-record(project, {id, name}).
-record(project_assignment,
        {project_id, person_id, hours}).

list_females() ->
  mnesia:transaction(fun() ->
      Q = qlc:q([{P#person.first_name, P#person.last_name}
          || P <- mnesia:table(person),
          P#person.sex == female]),
      qlc:e(Q)
    end).
```

Zapytania 2

```
-record(person, {id, first_name, last_name, sex}).
-record(project, {id, name}).
-record(project_assignment,
        {project_id, person_id, hours}).

assigned_people(Project) ->
    Q = qlc:q([Person ||
        Person <- mnesia:table(person),
        Assign <- mnesia:table(project_assignment),
        Assign#project_assignment.project_id
            == Project#project.id,
        Assign#project_assignment.person_id
            =:= Person#person.id])
    qlc:e(Q).
```

Linki

- Erlang — oficjalna strona: <http://erlang.org >
- prezentacja: <http://sirius.cs.put.poznan.pl/ inf66226/erlang/ >