Piotr Pałach

Język programowania Erlang

1. Informacje podstawowe

1.1. Opis języka

Erlang został stworzony w laboratorium firmy Ericsson i jest z powodzeniem stosowany w projektach wymagających wysokiej skalowalności i odporności na awarie. Jest to język funkcyjny, dynamiczny, o ścisłym typowaniu, w którym główny nacisk położono na współbieżność oraz możliwości przetwarzania rozproszonego. Erlang był własnościowym językiem programowania używanym przez Ericssona, jednak od roku 1998 kiedy kod źródłowy został otwarty, zyskuje z dnia na dzień na popularności. Z powodzeniem używany jest w komercyjnych produktach na całym świecie, które wykraczają niejednokrotnie poza jego telekomunikacyjne korzenie.

1.2. Nazwa

Nazwa Erlang została nadana na cześć pierwszego naukowca w dziedzinie telekomunikacji: Agnera Krarupa Erlanga (1878-1929), matematyka dunskego, pioniera teorii ruchu telekomunikacyjnego oraz teorii kolejek. Czasami nazwa języka bywa interpretowana jako Ericsson LANGuage.

2. Cechy Erlanga.

2.1. Współbieżność

Jedna z głównych zalet Erlanga. Procesy są podstawowymi elementami w strukturze aplikacji napisanych w Erlangu. Implementacja współbieżnych procesów jest niezależna od systemu operacyjnego i nie używa specyficznych dla danego środowiska mechanizmów jak na przykład wątki. Tworzenie i zarządzanie erlangowymi procesami jest bardzo proste, podczas gdy w innych językach współbieżność dostarcza wielu problemów oraz jest źródłem nie małej ilości różnego rodzaju błędów. Erlangowe procesy są dużo lżejsze od systemowych(ok. 300 bajtów na jeden proces), dzięki czemu operacje tworzenia i niszczenia procesów są relatywnie tanie obliczeniowo i pamięciowo. W pojedynczym erlangowym systemie możliwe jest utworzenie nawet milionów procesów bez znaczącego obniżenia wydajności. W Computer Language Shootout w konkursie na przesyłanie wiadomości pomiędzy tysiącami wątkow (thread-ring) rozwiązanie napisane w Erlangu zajmuje pierwsze miejsce pod

względem wydajności oraz objętości kodu.
Komunikacja pomiędzy procesami w Erlangu(z racji tego, że procesy nie dzielą między sobą wspólnej pamięci) odbywa się poprzez przesyłanie asynchronicznych komunikatów. Z każdym procesem powiązana jest kolejka FIFO, służąca do odbioru komunikatów. Proces odbiorczy decyduje o tym, czy dany komunikat przetworzyć, czy nie. Wszystko to jest Każdy proces ma swoją skrzynkę (ang. mailbox), w postaci kolejki, w której są przechowywane wiadomości wysłane przez inne procesy dopóki nie zostaną odczytane. Odbieranie wiadomości odbywa się przez mechanizm dopasowania wzorca. Po odczytaniu wiadomości proces Erlanga wraca do wykonywania. Do utworzenia wiadomości może zostać użyta dowolna struktura danych(liczby całkowite, zmiennoprzecinkowe, znaki, atomy), a nawet funkcje.

2.2. Rozproszoność

W język wbudowany jest mechanizm rozproszenia. Można tworzyć procesy z dowolnego węzła na dowolnym węźle. Proces może zostać utworzony na zdalnym węźle, a komunikacja z nim jest przezroczysta (tzn. komunikację z zdalnym procesem przeprowadza się w dokładnie takim sam sposób jak z procesem lokalnym).

2.3. Obsługa błędów

Obsługa błędów odbywa się na ogół przez nadzór jednych procesów nad innymi. Kiedy proces się zepsuje, wychodzi i wysyła wiadomość do procesu kontrolnego, który może podjąć odpowiednią akcje - np. restart bądź zakończyć zepsuty proces i uruchomić następny. Takie podejście do programowania zwiększa niezawodność i czyni kod mniej skomplikowanym. Ponadto proces kontrolny i proces nadzorowany nie muszą znajdować się na tej samej fizycznie maszynie, dzięki czemu stosunkowo łatwo daje się programować systemy wysokiej dostępności, w których poszczególne funkcje wykonywane są przez sprawne w danym momencie węzły.

2.4. Gorąca wymiana kodu(Hot code upgrade)

Wiele systemów nie może być zatrzymana w celach wymiany oprogramowania. W Erlangu można wymieniać kod w uruchomionym systemie(w systemie koegzystują przez chwile oba kody), a tym samym instalować poprawki błędów oraz uaktualnienia bez zatrzymywania działania systemu.

2.5. Obsługa systemów wieloprocesorowych

Od wersji R11 środowisko Erlanga potrafi automatycznie wykorzystać potencjał ukryty w procesorach wielordzeniowych (multi-core) oraz systemach wieloprocesorowych. Wcześniej aby wykorzystać taki system należało uruchomić kilka kopii maszyny wirtualnej, co niestety pociągało zwiększenie zużycia pamięci, oraz zwiększało niepotrzebnie koszty przesyłania wiadomości pomiędzy procesami na jednym komputerze. W większości przypadków kod, napisany dla jednego procesora, bez problemu wykorzysta możliwości takiej architektury, i to bez żadnych zmian w programie.

3. Zastosowania języka Erlang

Najpopularniejsze aplikacje, które zostały napisane w erlangu to:
— Mnesia, rozproszona, odporna na błędy baza danych czasu rzeczywistego, rozprowadzana wraz z Erlangiem
— ejabberd, serwer Jabber napisany w Erlangu, który jest uznawany za bardzo stabilny i wysoko skalowalny
— Yet another web server, Yaws, wysoko wydajny serwer stron Web który w wielu testach generowania stron dynamicznych przegania Apacha
— Tsung, wysoko wydajne narzędzie diagnostyczne do przeprowadzania testów wydajności
— oraz wiele innych aplikacji zarówno klienckich jak i również serwerowych
Zastosowania języka erlang pokazują, że ten język nie jest kolejnym mało znaczącym językiem programistycznym, którego ktoś wymyślił z powodu dużo wolnego czasu, ale bardzo dobrze sprawdzoną i stabilną platformą do rozwoju oprogramowania. Obecnie Erlang używany jest przez kilka ogólnoświatowych firm telekomunikacyjnych takich jak: Nortel, T-Mobile i Telia.

4. Typy danych

Erlang jest językiem z dynamicznym typowaniem oraz statycznym pojedynczym przypisaniem. Oznacza to, iż nie określamy jawnie typu zmiennej, jednak wartość zmienne musimy już jawnie określić. Ponadto w przeciwieństwie do wielu języków jak np. C++, czy Java raz przypisana wartość do zmiennej nie może zostać później zmieniona. Nową zmienną tworzy się poprzez jej pierwsze użycie, kiedy zostaje przypisana jej wartość.
W praktyce oznacza to:
`X`= 3
Kiedy ten kod zostanie wykonany, Erlang za pomocą operatora = porównuje lewą stronę z prawą stroną i próbuje znaleźć zwór. Jeśli znajdzie że lewa strona równania nie została jeszcze utworzona, a prawa strona jest wartością, tworzy zmienną i przypisuje mu wartość.
Jeżeli teraz chcielibyśmy przypisać inną wartość do zmiennej x zostanie wyrzucony błąd.
`X`= 10
To samo ograniczenie wpływa na instrukcję popularną w C++ i Javie
`X`+= 5

Wyróżniamy podstawowe typy danych

4.1. Liczba (całkowita lub zmiennoprzecinkowa)

Liczby w Erlangu są konstruowane jak w wielu innych językach, przy pomocy cyfr. Liczby całkowite mają nieograniczony zakres(jedyne ograniczenie - wielkość pamięci).

Przykład:

```
1
-15
123.1233
```

4.2. Atom

Atom jest to literał, którego nazwa zaczyna się z małej litery. Jeżeli atom zaczyna się z dużej litery, lub zawiera spacje musi być ujęty w pojedynczy apostrof '.

Przykład:

```
pierwszy
drugi_atom
'Trzeci atom'
```

4.3. Tuple

Tuple składa się ze stałej liczby elementów otoczonych parą nawiasów klamrowych ({ i }) . Element tupla może się składać z liczby, bądź cyfry. Dowolny elementy tupla może być łatwo dostępny, szczególnie przy użyciu wzorców. Tuple możemy manipulować za pomocą funkcji BIF - Build In Function(funkcje które zostały wbudowane w wirtualną maszynę Erlang).

Przykład:

```
1> T1 = {piotr,23,{luty,29}}.
{piotr,23,{luty,29}}
2> element(1,T1).
piotr
3> element(3,T1).
{luty,29}
4> T2 = setelement(2,T1,25).
{piotr,25,{luty,29}}
5> tuple size(T1).
3
6> tuple size({}.
0
7> {Imie,Lata,Data_urodz} = T1.
{piotr,23,{luty,29}}
8> Imie.
piotr
```

4.4. Lista

Lista w przeciwieństwie do tupla może zawierać zmienną liczbę elementów oddzielonych przecinkami, które otoczone są parą kwadratowych nawiasów [i]. Ciąg znaków otoczonych cudzysłowem również jest listą, a jej elementami są litery (które są liczbami 32-bitowymi). Możemy utworzyć listę z

elementami różnych typów, ze względu na to że Erlang jest językiem dynamicznym, w którym występuje nieścisłe typowanie. Dostęp do elementów listy jest realizowany przy pomocy dostępu do głowy listy (head), oraz ogona (tail). Głowa listy zawiera pierwszy element, natomiast ogon pozostałą część listy(podobieństwo do listy Prologowskiej). Składnia języka ułatwia operacje na listach, a biblioteka standardowa STDLIB zawiera wiele pomocnych funkcji do obsługi list.
Przykład:

```
1>  L1 = [b,1,{d,2 }].
[b,1,{d,2 }].
2> [H|T] = L1.
[b,1,{d,2 }].
3> H.
b
4> T.
[1,{d,2 }].
5> L2 = [d|T].
[d,b,1,{d,2 }].
6> length(L1).
3
7> length([]).
0
```

Listy umożliwiają różnego rodzaju filtoranie danych.

```
1> L = [1,2,3,4,5].
[1,2,3,4,5]
2> [E +1 || E <-  L].
[2,3,4,5,6]
3> [E || E <-  L, E < 3].
[1,2]
4> [E + 1000 || E <-  L, E =/= 1, E < 3 orelse E > 4].
[1002,1005]
6> [E + 1 || E <- [1,"Kitten","Batman",5], is_integer(E) ].
[2,6]
7> [ A || {A,B} <- [{a,b},{c,d},1,tomato,{e,f}] ].
[a,c,e]
```

Trzy pierwsze operacje na listach sprowadzają się do wykonania akcji dla każdego E w liście L gdzie warunek jest spełniony. Ostatnie dwie instrukcje pokazuja że filtrowanie danych na liście, może być wykonywane dla różnych typów elementów.

4.5. Inne typy

— fun() - jest to obiekt funkcyjny, umożliwia stworzenie funkcji anonimowej która będzie w sobie zawierała odwołanie do prawdziwej funkcji
```
1> Fun1 = fun (X) -> X+1 end.
#Fun<erl eval.6.39074546>
```

```
2> Fun1(3).
4
```

— pid procesu - potrzebny do przekazywania komunikatów, albo sprawdzania stanu procesu
— port -służy do komunikacji przy pomocy komunikatów ale z elementami poza językiem Erlang, np. programem C, potokiem UNIX, czy socketem TCP
— rekord - tuple o specjalnej konstrukcji
— boolean - wartość logiczna true lub false

5. Kilka prostych przykładów

5.1. Silnia

Wzór na silnię przedstawia się następująco

```
n! = 1,  if n = 0,
n! = n * (n - 1)!,  if n > 0
```

Zapis silni w erlangu jest bardzo zwięzły :

```
silnia(0) -> 1;
silnia(N) -> silnia(N-1) * N.
```

5.1.1. Forma Klauzulowa

W erlangu do zapisu wyrażenia warunkowego wykorzystano tak zwaną formę klauzulową. Składa się ona z pewnej liczby klauzul opisujących przypadki związane z wyznaczaniem wartości funkcji:

```
funkcja(argumenty1) -> ciało1; % klauzula1
funkcja(argumenty2) -> ciało2; % klauzula2
...
funkcja(argumentyN) -> ciałoN. % klauzulaN
```

Poszczególne klauzule pisane są jedna pod drugą, rozdzielone są znakiem średnika a ostatnia zakończona jest kropką. Kiedy funkcja o takiej postaci jest wykonywana wybierana jest jedna klauzula, pierwsza z góry, która spełnia warunek. Wybór następuje na podstawie związanych z nimi warunków. Liczy się przy tym kolejność definicji klauzul.
W naszym programie mamy tylko dwie przypadki (`N>0`and N=0). Jeśli argumentem wywołania funkcji będzie 0 zostanie ono dopasowane do argumentu pierwszej klauzuli i zwrócona zostanie wartość 1. W przeciwnym przypadku zadziała klauzula

5.1.2. Dopasowanie do wzorca

Podstawowym sposobem dostarczania warunków jest mechanizm dopasowywania do wzorca. Dopasowanie do wzorca polega na sprawdzeniu czy oba typy argumentów do siebie pasują. W rozważanym przypadku, liczby pasują gdy są sobie równe, zmienne zaś pasują do każdej liczby przejmując jednocześnie jej wartość. Wartościowanie logiczne dopasowania jest istotną różnicą

pomiędzy opisywanym a standardowym, imperatywnym przypadkiem gdzie następuje tylko przypisanie do zmiennych. W przypadku silni jeśli argumentem wywołania funkcji będzie 0 zostanie ono dopasowane do argumentu formalnego pierwszej klauzuli i zwrócona zostanie wartość 1. W przeciwnym przypadku zadziała klauzula 2.

5.1.3. Definicja modułu prosteFunk oraz funkcji silnia

Wszystkie funkcje w Erlangu powinny być definiowane w modułach. Moduł prosteFunk.erl zawiera zdefiniowaną silnię wraz z odpowiednimi nagłówkami

```
-module(prosteFunk).
-export([silnia/1]).

silnia(0) -> 1;
silnia(N) when N>0 -> silnia(N-1) * N.
```

Znaczenie Insrukcji:
-module(prosteFunk) - nazwa modułu zgodną z nazwą pliku
-export([silnia/1]) - nazwa funkcji, oraz liczba argumentów(funkcja silnia posiada jeden argument)

5.1.4. Kompilacja I uruchomienie funkcji silnia

```
1> c(prosteFunk).
{ok,prosteFunk}
2> prosteFunk:silnia(25).
15511210043330985984000000
```

5.2. Algorytm Quicksort

5.2.1. Defnicja funkcji Quicksort

```
-module(prosteFunk).
-export([quicksort /1]).
quicksort ([]) -> [];
 quicksort([Pivot|T]) ->
     quicksort ([X||X <- T, X < Pivot]) ++
      [Pivot] ++
     quicksort ([X||X <- T, X >= Pivot]).
```

Powyższa definicja funkcji Quicksort wywołuje rekurencyjnie funkcje quicksort dopóki wszystkie elementy nie zostały posortowane.

Wyrażenie [X||X¡- T, X ¡ Pivot] oznacza: dla każdego X który należy do ogona listy, który nie zawiera już pierwszego elementu, i takich X które są mniejsze od elementu Pivot(element podziału, środkowego) wykonaj rekurencyjnie sortowanie tego zbioru.

Operator ++ ma za zadanie połączyć powstałą po sortowaniu listę

5.2.2. Kompilacja I uruchomienie funkcji quicksort

```
1> prosteFunk:quicksort([1,4,6,8,2]).
[1,2,4,6,8]
2> prosteFunk:quicksort([1,4,6,a,g,8,2]).
[1,2,4,6,8,2,g]
{ok,prosteFunk}
```

5.3. Prosty proces wypisujący wiadomość którą otrzymał

5.3.1. Kilka słów o procesach

Większość równocześnie działających funkcji w Erlangu tworzy się bardzo prosto za pomocą funkcji:
`spawn(Module,Fun,Args)` `spawn(fun() -> loop() end)`
która zwraca identyfikator procesu, używany do komunikacji z nowo utworzonym procesem.

Wyrażenie `Pid ! Wiad` wysyła wiadomość do konkretnego procesu za pomocą identyfikatora procesu - Pid. Wiadomość jest odbierana używając konstrukcji `receive ... end`, która rozpoznaje rodzaj wiadomości otrzymanej za pomocą klazul które zostały zadeklarowane.

5.3.2. Definicja funkcji

```
start() ->
  spawn(fun() -> loop() end).

loop() ->
  receive
    {kwadrat, X}->
      io:format("~nNapisales ~p wart=~p ~n",[kwadrat,X*X]),
     loop();
    {szescian, X}->
      io:format("~nNapisales ~p wart=~p ~n",[szescian,X*X*X]),
      loop();
    Any ->
      io:format("~nDostalem wiadomosc: ~p~n",[Any]),
      loop()
  end.
```

Przykład pokazuje funkcję servera który wykonuje akcję(wypisuje komunikat i liczy wartość w zależności od rodzaju przesłanej wiadomości za pomocą omawianej wcześniej techniki dopasowania wzorca.
Funkcja start() tworzy nowy proces dla funkcji loop(). Funkcja loop() nasłuchuje kanał(czeka) na wiadomość, w zależności jaki format wiadomości

5.3.3. Uruchomienie procesu i wysłanie wiadomości

```
1> Pid = prosteFunk:start().
<0.240.0>
2> Pid ! {kwadrat,10}.
```

```
{kwadrat,10}
Napisales kwadrat wart=100
3> Pid ! {szescian,5}.
{szescian,5}
Napisales szescian wart=125
4> Pid !{"nic nie warta wiadomosc"}.
{"nic nie warta wiadomosc"}
Dostalem wiadomosc: {"nic nie warta wiadomosc"}
```

Uruchamiając funkję start pobieramy Pid(identyfikator tego procesu). Teraz możemy już wysyłać wiadomości do servera za pomocą składni Pid ! wiadomość. Jeżeli nasza wiadomosc zawierała: nazwe kwadrat, i wartośc, to server dopasuje ją do pierwszej możliwości(kwadrat), pomnoży naszą wartość i wyśle odpowiedz.

5.4. Gorąca wymiana kodu

5.4.1. Kilka słów o programie

Program pokazuje możliwość podmiany kodu programu bez zatrzymywania systemu. Język erlang przechowuje w pamięci dwie wersje kodu danej aplikacji. Wersja która ma aktualniejszą datę, jest uruchamiana.

5.4.2. Kod programu

```
-module(wymianaKodu).
-export([start/0,server/0,client/2]).

start( ) -> Pid = spawn(wymianaKodu, server, [])
  , spawn(wymianaKodu, client, [0,Pid]).

server( ) ->
    receive
        Msg ->
            io:format('przed otrzymano: ~p~n', [Msg]),
            wymianaKodu:server( )
    end.

client(N, Pid) ->
    Pid ! N,
    client(N+1, Pid).
```

Za pomocą funkcji start tworzymy proces funkcji server oraz funkcji client która jako argument dostaje Pid procesu servera. Funkcja klient przesyła wiadomość do servera, server wyświetla ją na ekran.

5.4.3. Wyniki uruchomienia programu

```
1> c(wymianaKodu).
{ok,wymianaKodu}
2> wymianaKodu:start().
3> przed otrzymano: 0
```

```
4> przed otrzymano: 1
5> przed otrzymano: 2
6> c(wymianaKodu).
7> po otrzymano: 3
8> po otrzymano: 4
```

Kompilujemy program i uruchamiamy za pomocą funkcji start(). Server wyświetla text wiadomości na podstawie starego kodu. Po edycji(zmianie w tekście wyświetlanym przez serwer słowa "przed" na "po" i skompilowaniu ponownie kodu maszyna wirtualna erlanga automatycznie przeładuje przy następnym wywołaniu funkcji server nowy kod.

5.5. Większy przykład - program czat

Program chat wypisujuje informacje przesłane przez jednego użytkownika każdemu użytkownikowi który aktualnie znajduje się w pokoju, także informacja o dołączeniu nowego użytkownika do pokoju jest wysyłana każdemu obecnemu użytkownikowi pokoju.

6. Wykaz źródeł

— http://erlang.org/
Strona domowa języka Erlang
— http://erlang.org/doc/pdf/
Pełna Documentacja
— http://erlang.org/white_paper.html
Wprowadzenie do Erlanga z przykładami
— http://www.erlang.pl/doku.php
Główna polska strona języka Erlang