Štúdia využitia AOP na implementáciu bezpečnosti v jazyku Erlang

Aspektovo orientovaný vývoj softvéru, FIIT STU, 2015

Bc. Martin Šustek,
Softvérové inžinierstvo,
2. ročník, 3. semester

Abstract

Využitie aspektovo orientovaného programovania je príznačné najmä pre oblasť objektovo-orientovaných jazykov, no implementácie umožňujúce využitie tohto prístupu vznikajú aj pre funkcionálne jazyky. Výnimkou neostal ani Erlang. Erlang je funkcionálny jazyk vytvorený pre vytváranie distribuovaných a škálovateľných aplikácií s vysokou dostupnosťou.

Použitie AOP v Erlangu sa stalo možným vďaka implementácii vytvorenej Alexeiom Krasnopolskim (1). Od nej sa odvíja aj implementácia funkcionality opísanej v tejto správe, ktorá pojednáva o možnostiach riešenia nízkej úrovne zabezpečenia komunikácie procesov a uzlov v rámci zhluku virtuálnych strojov Erlangu. Ďalším prínosom, o ktorý sa riešenie pokúša, je zavedenie štandardov pre štruktúru vymieňaných správ, dodržiavajúcich best-practices zavedené jazykom

1. Introduction

Z povahy jazyka Erlang vyplýva fakt, že programy, v ňom napísané, sú zväčša spúšťané v distribuovanom prostredí (5), na viacerých výpočtových uzloch. Na každom z takýchto uzlov musí fungovať virtuálny stroj Erlangu a tieto stroje musia o sebe navzájom vedieť. Prepojenie virtuálnych strojov prebieha počas inicializácie zhluku (clustra) tak, že si jednotlivé uzly vymenia „security cookie“. Toto je zároveň jediný mechanizmus zabezpečujúci bezpečnosť komunikácie, keďže Erlang bol vytvorený primárne pre beh v rámci intranetovej siete.

Ďalšou oblasťou bez implicitných štandardov je v Erlangu posielanie správ. To je vstavanou vlastnosťou jazyka a poslanie správy zabezpečuje bez mimoriadnych výdajov použitie "bang operatora" - ! . Programátorovi je však ponechaná voľnosť v tom, ako sa rozhodne koncipovať telo správy. Z praxe vyplynuli určité best-practices, ktoré komunita používa. Sú nimi najmä:

- ako správy sa používajú *označené n-tice (tagged tuples)*, čo znamená, že správa je dátového typu tuple a jej prvým prvkom je atom, ktorý označuje jej obsah (napr. { tag, "content string" })
- do správy sa pridáva *Pid odosielateľa*, aby jej prijímateľ vedel, komu má odpovedať
- správa sa označuje *unikátnou referenciou*, aby v prípade synchrónnej komunikácie proces vedel jednoznačne identifikovať, že odpoveď, ktorú obdržal, je odpoveďou na správu, ktorú pôvodne odoslal.

Pre vytvorenie štandardov pre bezpečnosť a štruktúru komunikácie sme sa rozhodli použiť modul ErlAOP. Proces realizácie tejto úlohy je opísaný v ďalších kapitolách.

2. Prerequisites

Pre spustenie výstupných modulov potrebuje mať záujemca nainštalované minimálne nasledujúci softvér:

- Erlang (2)
    - Optional: Erlang/OTP 17
- AOP for Erlang (3)

3. Implementation

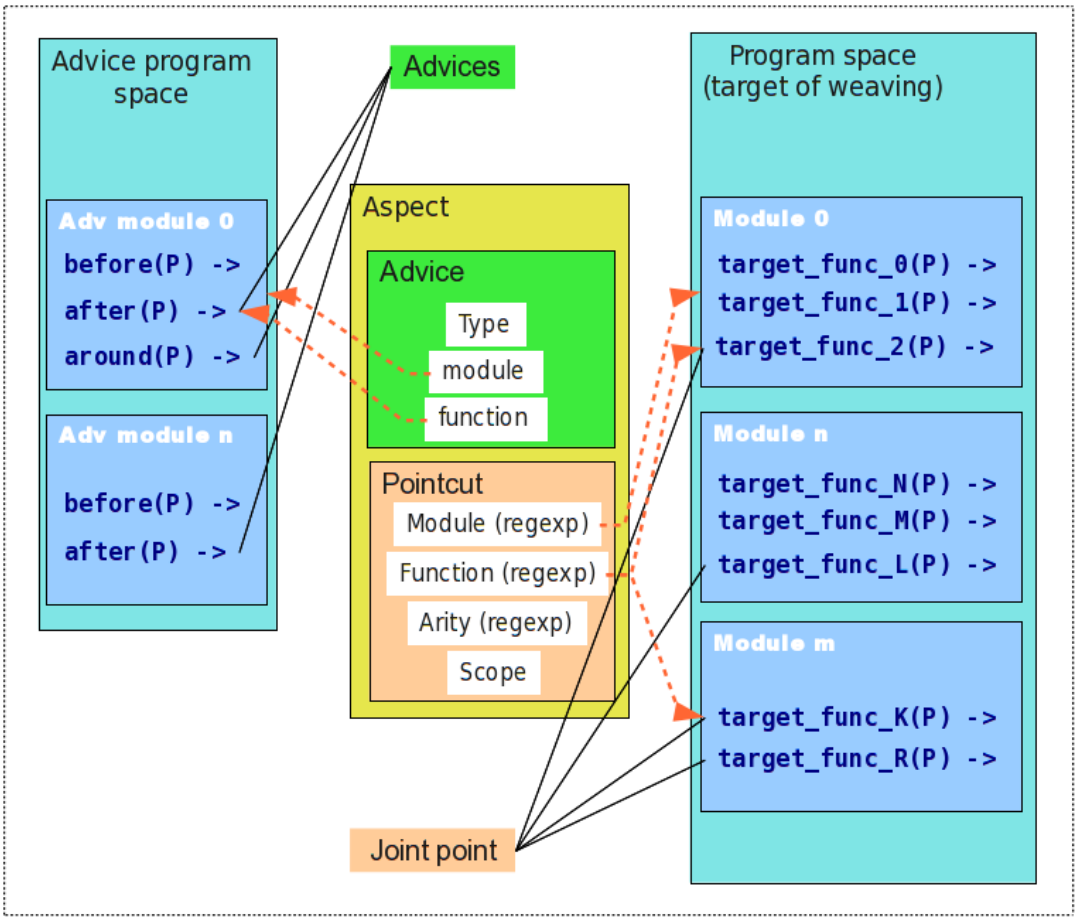
Pomocou spomenutého modulu ErlAOP sme implementovali Around advice, ktoré obaľuje volanie funkcie send_eaop() a má na starosti manažment bezpečnosti komunikácie a konzistentosti správ. Tento proces je opísaný v tejto kapitole.

3.1 AOP for Erlang

ErlaAOP neimplementuje všetky mechanizmy AOP a aj tie, ktoré sú implementované, sú upravené tak, aby korešpondovali s vlastnosťami jayzka. Sú nimi:

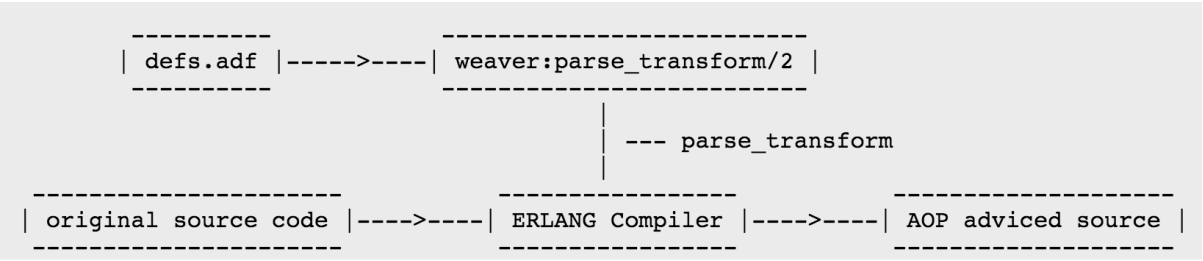
- Join point - môže ním byť iba volanie funkcie,
- Pointcut - regulárny výraz, ktorý zahŕňa jeden alebo viac joinpoints. Používa sa implementácia regexu pre Erlang (PCRE),
- Advice - telo funkcie vykonávanej v Join pointe vyhovujúcom konkrétnemu regulárnemu výrazu. Jedná sa zväčša o dvojicu modul-funkcia,
- Target - funkcia, modul alebo množina modul, pre ktoré chceme uplatniť AOP správanie,
- Aspekt - kombinácia Advice a k nemu prislúchajúcich Pointcuts,
- Proxy - modul funguje vďaka tomu, že pôvodnú funkciu obalí proxy funkciou a pri jej volaní dôjde k zavolaniu proxy a pôvodnej funkcie v poradí podľa povahy Advice,
- Vtkanie - proces párovania aspektov na cieľový zdrojový kód a generovania proxy funkcií na miesto, kde mali stáť pôvodné funkcie.

Opísané mechanizmy a ich prepojenia sú zobrazené na obrázku 1.



Obrázok 1: Umiestnenie AOP konceptov v rámci modulu ErlAOP

Takto definovaná schéma mechanizmov sa po vytvorení špecifickej implementácie premietne do schémy vtkania zobrazenej na obrázku 2.



Obrázok 2: Proces vtkania aspektov do kódu v rámci modulu ErlAOP

3.2 Specific functionallity implementation

Nasledujúca podkapitola opisuje proces definovania špecifických častí pre ErlAOP moduly, ktoré boli spomenuté v predchádzajúcej kapitole a ich vytvorenie je nutné pre správne vtkanie a uplatnenie aspektov. Sú nimi:

- program.erl - cieľový program implementujúci funkciu, ktorej volanie vyústi v uplatnenie aspektu s Around advice
- advices.erl - telo funkcionality Around advice
- defs.adf - súbor definujúci mapovanie cieľových funkcií na aspekty

Cieľový program je tvorený jednou funkciou obaľujúcou odoslanie správy globálne registrovanému procesu. Jej implementáciu demonštruje obrázok 3.

```
1   -module(program).
2   -author("Martin Sustek").
3   -date({2015,11,26}).
4   -version("1.0").
5   -export([send_eaop/3]).
6
7   send_eaop(TargetName, TargetNode, Msg) ->
8       {TargetName, TargetNode} ! Msg
9   .
```

Obrázok 3: Cieľová funkcia obalená aspektom

V module advices.erl je definované jediné advice typu Around. Toto v poradí, v akom sú menované, uplatní nasledujúce funkcie (implementácia zobrazená na obrázku 4):

- pomenovanie uzla
- kontrola sieťovej konektivity
- nastavenie security cookie
- overenie funkčnosti cieľového uzla
- preštylizovanie správy na unifikovaný formát

```
1   -module(advices).
2   -author("Martin Sustek").
3   -date({2015,11,26}).
4   -version("1.0").
5   -export([around_advice/3]).
6
7   around_advice(M, F, Args) ->
8       node_name_check(M, F, Args, node())
9   .
10
11  node_name_check(M, F, Args, nonode@nohost) ->
12      NodeName = list_to_atom(integer_to_list(erlang:phash2({self()}))),
13      io:format("<<< Setting node [NAME] to: ~p~n", [NodeName]),
14      net_kernel:start([NodeName, shortnames]),
15      alive_check(M, F, Args, is_alive());
16  node_name_check(M, F, Args, NodeName) ->
17      io:format("<<< Node [NAME] exists and is: ~p~n", [NodeName]),
18      alive_check(M, F, Args, is_alive())
19  .
20
21  alive_check(M, F, Args, true) ->
22      io:format("<<< {Source} Node is [ALIVE], proceeding to ping.~n"),
23      cookie_check(M, F, Args, erlang:get_cookie());
24  alive_check(_M, _F, _Args, false) ->
25      io:format("<<< {Source} Node is [DOWN] exiting."),
26      {source_down, "Failed to send message, source node is down!~n"}
27  .
28
29  cookie_check(M, F, [TargetName, TargetHost, Msg], nocookie) ->
30      NodeCookie = 'b76rtSEPOirgyu34werbfGHekadf',
31      io:format("<<< Node [COOKIE] not set. Setting to: ~p~n", [NodeCookie]),
32      erlang:set_cookie(node(), NodeCookie),
33      ping_node(M, F, [TargetName, TargetHost, Msg], net_adm:ping(TargetHost));
34  cookie_check(M, F, [TargetName, TargetHost, Msg], Cookie) ->
35      NodeCookie = 'b76rtSEPOirgyu34werbfGHekadf',
36      io:format("<<< Node [COOKIE] set to: ~p. Overwriting with: ~p~n", [Cookie, NodeCookie]),
37      erlang:set_cookie(node(), NodeCookie),
38      ping_node(M, F, [TargetName, TargetHost, Msg], net_adm:ping(TargetHost))
39  .
40
41  ping_node(M, F, Args, pong) ->
42      io:format("<<< {Target} Node is [ALIVE], proceeding to cookie check.~n"),
43      msg_structure_check(M, F, Args);
44  ping_node(_M, _F, _Args, pang) ->
45      io:format("<<< {Target} Node is [DOWN], exiting.~n"),
46      {target_down, "Failed to send message, target node is down!"}
47  .
48
49  msg_structure_check(M, F, [TargetName, TargetHost, { Tag, Ref, Pid, MsgBody }]) when is_atom(Tag), is_reference(Ref), is_pid(Pid) ->
50      io:format("<<< Message is [REFERENCED TUPLE with SENDER Pid].~n"),
51      erlang:apply(M, F, [TargetName, TargetHost, { Tag, Ref, Pid, MsgBody }]);
52  msg_structure_check(M, F, [TargetName, TargetHost, { Tag, Ref, MsgBody }]) when is_atom(Tag), is_reference(Ref) ->
53      io:format("<<< Message is [TAGGED TUPLE with REFERENCE].~n"),
54      erlang:apply(M, F, [TargetName, TargetHost, { Tag, Ref, self(), MsgBody }]);
55  msg_structure_check(M, F, [TargetName, TargetHost, { Tag, MsgBody }]) when is_atom(Tag) ->
56      io:format("<<< Message is [TAGGED TUPLE].~n"),
```

Obrázok 4: Špecifická implementácia Around advice štrukturujúceho komunikáciu

Poslednou časťou implementácie je modul defs.adf, v ktorom sú zapísané Pointcuts a prepája implementácie Advices s cieľovými programami. Príklad z nami vytvorenej implementácie zobrazuje obrázok 5.

```
1  %% defs.adf
2  [Aspect(
3      Advice(around, advices, around_advice),
4      [ Pointcut("program", "send_eaop", 3, public) ]
5  )]
6  .
```

Obrázok 5: Definícia bodového prierezu v ErlAOP

4. Evaluation

Posledným krokom zachyteným v tejto správe je overenie funkčnosti riešenia. Pre tento proces sme si zvolili jeden konkrétny scenár, zachytávajúci všetku funkcionalitu, ktorú riešenie poskytuje.

Tento príklad, ktorý poslúžil na vyhodnotenie predstavuje najhorší možný prípad nezabezpečenej komunikácie, nakoľko:

- zdrojový uzol nie je pomenovaný
- zdrojový uzol pred odoslaním správy neoveril, či je schopný sieťovej komunikácie
- zdrojový uzol nemá priradenú security cookie
- zdrojový uzol pred odoslaním správy neoveril, či je cieľový uzol schopný prijímať správy
- správa nie je štruktúrovaná na žiadnej úrovni, neobsahuje referenciu ani PID odosielateľa

```
➜ xsustekm  erl
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Eshell V6.4  (abort with ^G)
1> c:cd("/Users/martin/Documents/AOVS/xsustekm/ebin").
/Users/martin/Documents/AOVS/xsustekm/ebin
ok
2> c("../src/aop.erl").
{ok,aop}
3> c("../src/fun_proxy.erl").
{ok,fun_proxy}
4> c("../src/weaver.erl").
{ok,weaver}
5> c("../examples/aspects/advices.erl").
{ok,advices}
6> aop:compile(["../examples/target"], ["../examples/aspects"]).
Sources = ["../examples/target/program.erl"]
Config files  = ["../examples/aspects/defs.adf"]
Module name = program is weaved.        Warnings = []
ok
7> nodes().
[]
8> program:send_eaop(target, 'target@Angolinha', "Eaop Test Message").
<<< Setting node [NAME] to: '28268026'
<<< {Source} Node is [ALIVE], proceeding to ping.
<<< Node [COOKIE] set to: 'SYCQDGVWPASYNTIUZEMV'. Overwriting with: b76rtSEPOirgyu34werbfGHekadf
<<< {Target} Node is [ALIVE], proceeding to cookie check.
<<< Message has [NO STRUCTURE].
{eaop_message,72618547,<0.32.0>,"Eaop Test Message"}
(28268026@Angolinha)9> nodes().
[target@Angolinha]
(28268026@Angolinha)10>
```

Obrázok 6: Spustenie na zdrojovom, nepomenovanom uzle bez security cookie

Obrázok 7: Spustenie na cieľovom, pomenovanom uzle s nastavenou security cookie

Aby takto vytvorené riešenie fungovalo v distribuovanom prostredí, moduly pre vtkanie aspektov a samotné aspekty musia byť prítomné na každom z uzlov a pred spustením hocakého programu na uzle, musí byť tento skompilovaný s využitím vtkania aspektov. Pri zachovaní týchto podmienok vieme dosiahnuť redukciu a sprehľadnenie kódu, ktoré demonštruje tabuľka 1.

Tabuľka 1: Dosiahnutá redukcia LOC

Bez využitia AOP	**S využitím AOP**

net_kernel:start(['source', shortnames]),
case is_alive() of
  false ->
  {
  source_down,
  "Failed to send message, source node is down!~n"
  }
  true ->
  erlang:set_cookie(
  node(),
  'b76rtSEPOirgyu34werbfGHekadf'),
  case net_adm:ping('target@Angolinha') of
  pang ->
  {
  target_down,
  "Failed to send message, target node is down!"
  }
  pong ->
  { target, 'target@Angolinha' } ! {
  eaop_msg,
  make_ref(),
  self(),
  "This is final message!"
  }
end.
	send_eaop(
	  target,
	  'target@Angolinha',
	  "This is final message!"
	).

5. Related work

Využitie aspektovo orientovaného programovania v doméne funkcionálnych jazykov je veľmi zriedkavé. Keďže sme túto doménu ešte ďalej zúžili na jazyk Erlang, neexistujú takmer žiadne porovnateľné riešenia. Pokiaľ sa jedná o implementáciu bezpečnosti, neexistuje dokonca žiadne, voľne dostupné, obdobné riešenie.

Nakoľko aj nami predstavené riešenie využíva hotovú implementáciu AOP pre Erlang (3), jediným zostávajúcim súvisiacim riešením je nasledujúci projekt:

- Erlang Weaver (https://github.com/efcasado/weaverl) - rámec pre vtkanie aspektov do kódu Erlangu

Conclusion and Future Work

Výsledkom práce zachytenej v tejto správe je dôkaz vhodnosti využitia aspektovo orientovaného prístupu k vývoju softvéru vo vzdialenej doméne distribuovaných funkcionálnych jazykov. Použité technológie poskytli solídny základ pre ďalšie rozšírenie už tak vyjadrovaním bohatého jazyka.

V závere došlo k unifikácii komunikácie, jej obaleniu bezpečnostnými štandardmi a k zabezpečeniu toho, že jej výsledkom bude vždy jednoznačný výsledok alebo chybová správa. Podarilo sa teda naplniť všetky požiadavky, ktoré boli na riešenie kladené.

Možnosti ďalšieho vývoja riešenia:

Zavedenie centrálneho registra povolených procesov

Pri prvom vykonaní volania obaleného aspektom by došlo k inicializácii vyhradeného procesu, ktorý by vytvoril schému vo vstavanej databáze Mnesia (4). Túto by zdieľali lokálne uzly a vzdialené uzly by ju mohli dopytovať vďaka tomu, že tento uzol by bol globálne pomenovaný a prístupný. Pokiaľ by sa uzol nenachádzal v tomto zozname, komunikácia s ním by nebola umožnená ani v prípade, že by zdieľal s ostatnými uzlami rovnakú security cookie. Takýmto mechanizmom by bolo hocikomu zabránené pristupovať k aktívnym uzlom napriek tomu, že by dokázal získať prístup do intranetovej siete a prelomil by security cookie.

Problémom tohto mechanizmu je fakt, že centrálny register by bol úzkym hrdlom celého systému a mohlo by byť teda ku nemu pristupované iba počas inicializácie uzla. V opačnom prípade by narúšal celý koncept distribuovateľnosti a škálovateľnosti spravovaného systému.

Dopracovanie štandardizácie správ

Riešenie zavádza štandardy pre vymieňané správy podľa praxe jazyka. Správy majú po úprave formát: *{ Označenie, Referencia, Pid odosielateľa, Telo správy }*

Týmto je zabezpečená konzistentnosť komunikácie. Je však potrebné dopracovať ukladanie referencií odosielaných správ v prípade, že programátor pôvodnú správu neoreferencoval. Pri prijatí výsledku jej spracovania by ju vďaka tomu nevedel spoľahlivo identifikovať. Pre dosiahnutie tejto funkcionality by bolo opäť potrebné využiť centrálny register, spomenutý vyššie, do ktorého by sa pod hashom odosielanej správy uložila referencia, ktorou bola označená.

Zdroje:

[1] http://sourceforge.net/u/krasnopolski/profile/
[2] http://www.erlang.org/download.html
[3] http://erlaop.sourceforge.net/
[4] http://www.erlang.org/doc/man/mnesia.html
[5] http://erlang.org/doc/reference_manual/distributed.html