Actor
- A Concurrency Model

조영일

목차

- 소개
  - Concurrency Model
  - Actor Model
- Erlang Grammar
  - variable
  - function
- Actor in Erlang
  - echo server
  - temperature converter
- Real world implementations
  - Some examples
- 정리
  - 장단점

Background

- 오늘날의 컴퓨팅 환경
  - CPU clock은 더 이상 높아지지 않음
  - 대신 core의 수가 급격한 속도로 늘어나고 있음
- Multi-thread programming
  - MT-safe하지 않은 기능들이 존재함
    - 레거시 시스템 라이브러리 뿐만 아니라 대부분의 애플리케이션 서버들이 동시성을 고려하지 않고 작성됨
  - 동기화해야 하는 자원이 존재하기 마련임
  - 로직 오류가 발생하거나 동시성이 떨어지게 되어 있음

Concurrency Model

- 프로세스나 쓰레드를 어떻게 동시에 잘 실행시킬까를 고민해서 만든 방식
- “Shared-state Model”
  - “synchronized”
    - critical section or lock
  - 문제점
    - multi-thread programming이 어렵고 확장성이 떨어짐
    - locking은 전반적인 효율성을 떨어뜨림

7 Concurrency Models

- Thread & Locks
- Functional Programming
  - mutable state를 제거. 쓰레드 안전 보장. 병렬처리 용이성
- Clojure Way - separating identity and state
  - 명령형과 함수형의 결합. atomic/persistent(multiversion) 자료구조. identity와 state 분리
- Actor
  - 지리적 분산, 장애 허용. 탄력성
- CSP(Communicating Sequential Process)
  - Actor와 비슷하지만 채널을 이용. 비동기 go block
- Data Parallelism
  - GPU를 이용한 병렬처리
- Lambda Architecture
  - Map&Reduce, Streaming. Batch & Realtime

Actor Model

- A concurrency model
  - no shared state — without lock
  - lightweight process — == thread
  - async message passing
  - mailbox — == message queue
    - pattern matching

Erlang

- Functional, dynamic typing language
- History
  - in 1986 at Ericsson
  - for nonstop system
  - open sourced in 1998
- Actors are part of the language

Basic Grammar

- Variable
  - not assignment but matching & side effect
    - `Value = 4.`
    - `Value.`
    - `Value = 6.  ** exception error`
  - Tuple
    - `*` Literal in lowercase
    - `Stooges = {larry, curly, moe}.`
    - `{S1, S2, S3} = {larry, curly, moe}.`
  - List
    - `List = [1, 2, 3].`
    - `[First, Rest] = List.`

Basic Grammar

- Function
  - definition
    - `Square = fun(X) -> X*X end.`
  - call
    - `Square(5).`
  - compiling
    - `-module(mymath).`
    - `-export([square/1, fib/1]).`
    - `fib(0) -> 0;`
    - `fib(1) -> 1;`
    - `fib(N) when N>1 -> fib(N-1) + fib(N-2).`
    - `c(mymath.erl).`
    - `mymath.fib(7).`

Actor in Erlang

- Process spawning & Message Passing

```
-module(echo).
-export([start/0]).
loop() ->
  receive { Sender, Num } ->
     Sender ! Num, loop()
  end.
start() -> spawn(fun loop/0).

c(echo).
Pid = echo:start().
Pid ! { self(), 42 }.
receive Value -> Value end.
```

Actor in Erlang

- Idiom
  - Actor는 보통 tail recursion의 형태로 만들어짐
  - 상태(state)
    - 함수 파라미터로 전달되고 (필요하면) 재귀 호출에서 변경됨
    - * 저장되지 않고 흘러다님

Actor in Erlang

- More complicated example
  - `-module(temperature).`
  - `-export([temperatureConverter/0]).`
  - `temperatureConverter() ->`
    - `receive`
      - `{toF, C} -> io:format("~p C is ~p F~n", [C, 32+C*9/5]), temperatureConverter();`
      - `{toC, F} -> io:format("~p F is ~p C~n", [F, (F-32)*5/9]), temperatureConverter();`
      - `{stop} -> io:format("Stopping~n");`
      - `Other -> io:format("Unknown: ~p~n", [Other]), temperatureConverter()`
    - `end.`

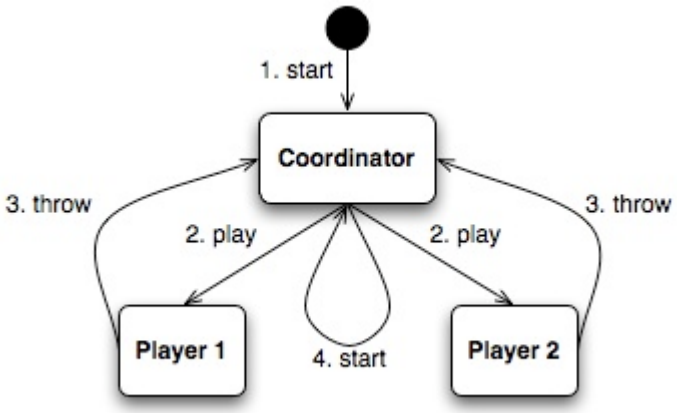
  - `c(temperature).`
  - `Pid = spawn(fun temperature:temperatureConverter/0).`
  - `Pid ! { toC, 32 }.`
  - `Pid ! { toF, 100 }.`
  - `Pid ! { stop }.`

Real World Implementations

- Erlang
  - -> ejabberd (Jabber/XMPP)
  - -> RabbitMQ (AMQP)
  - -> Facebook Chat
  - -> CouchDB
- Scala
  - in standard library (deprecated in 2.10)
  - Vert.x, Akka, Reactors.IO
- Java
  - Vert.x, Akka, Reactors.IO, Orbit, Quasar, Actor, Jetlang, JActor, S4, Korus, ActorFoundry, Peernetic, Ateji PX
- Python
  - Pulsar, Thespian, Pykka, PARLEY

Example

- 가위바위보
  - Coodinator
    - Start라는 메시지를 받아서
    - Player1과 2를 thread로 만들고
    - play라는 메시지를 전달
  - Player 1,2
    - throw라는 메시지를 전달



Scala example

```
class Player() extends Actor {
    def act() {
        // message handling loop
    }
}

abstract case class Move
case object Rock extends Move
case object Paper extends Move
case object Scissors extends Move

// message
case object Start
case class Play(sender:Actor)
case class Throw(player:Actor, move:Move)
```

Scala example

```
class Coordinator() extends Actor {
  def act() {
    val player1 = new Player().start
    val player2 = new Player().start
    loop {
      react {
        case Start => {
          player1 ! Play(self)
          player2 ! Play(self)
        }
        case Throw(playerA, throwA) => {
          react {
            case Throw(playerB, throwB) => {
              announce(playerA, throwA, playerB, throwB)
              self ! Start
            }
          }
        }
…
```

Kilim example

쓰레드와 같은 개념

메시지 채널

```
public class Player extends Task {
  private static final Random RANDOM = new Random();
  private final String name;
  private final Mailbox<PlayMessage> inbox;
  public Player(String name, Mailbox<PlayMessage> inbox) {
      this.name = name;    this.inbox = inbox;
  }
  public void execute() throws Pausable {
      while(true) {
          PlayMessage message = inbox.get();
          message.getResponseMailbox().putnb(
              new ThrowMessage(name, randomMove()));
    }
  }
  private Move randomMove() {
      return Move.values()[RANDOM.nextInt(Move.values().length)];
  }
}
```

ActorFoundry example

```
public class Player extends Actor {
    // etc
    @message public void play(ActorName coordinator) {
        send(coordinator, "playerThrows", name, randomMove());
    }
}
```

정리

- Actorl Model의 장점
  - shared-state가 아니므로 concurrency level을 극대화할 수 있음
  - Abstraction
    - 복잡한 application programming이 필요없음
  - 기존 방식의 문제점을 해결
    - race condition
    - deadlock
    - starvation
    - live locks
  - massive multi-core 환경에 적합함

정리

- Actor Model의 단점
  - trasaction이 필요한 곳에서는 부족함
  - async하므로 global consensus를 얻기 어려움
  - 문제를 잘게 쪼개는 것이 늘 쉽진 않음
  - 상속이나 구조에 대한 직접적인 개념이 존재하지 않음
  - 분산환경에서는 메시지 통신이 많이 필요함
  - 필연적으로 동적 타이핑이라서 정적 타이핑의 장점을 얻을 수 없음
    - 정적 분석을 통한 최적화
    - 메모리 요구사항 파악
  - 메시지 순서 때문에 Stack 구조와 맞지 않음

References

- Understanding actor concurrency, Part 1: Actors in Erlang - JavaWorld
- Understanding actor concurrency, Part 2: Actors on the JVM - JavaWorld
- Erlang Programming Language
- Erlang's actor model
- The Actor Model - Towards Better Concurrency
- Why has the actor model not succeeded?
- http://logonjava.blogspot.kr/2010/09/concurrent-processing-actor-model.html

- Thank You!