Инженерный вестник Дона, №2 (2025)
ivdon.ru/ru/magazine/archive/n2y2025/9858

Акторная модель в языке программирования Elixir: основы и применение

В.И. Шиян, А.М. Кузнецов, П.Н. Литвиненко, А.А. Полтавская, М.А. Прохоров, К.А. Степуленко, Д.С. Токарева

Кубанский государственный университет, Краснодар

Аннотация: В статье рассматривается акторная модель, реализованная в языке программирования Elixir, который является наследником языка Erlang. Акторная модель представляет собой подход к параллельному программированию, где независимые объекты, называемые акторами, взаимодействуют друг с другом посредством асинхронных сообщений. В статье подробно описаны основные концепции Elixir, такие как сопоставление с образцом, неизменяемость данных, типы и коллекции, а также механизмы работы с акторами. Особое внимание уделено практическим аспектам создания и управления акторами, их взаимодействию и поддержанию состояния. Статья будет полезна исследователям и разработчикам, интересующимся параллельным программированием и функциональными языками.

Ключевые слова: акторная модель, elixir, параллельное программирование, сопоставление с образцом, неизменяемость данных, процессы, сообщения, почтовый ящик, состояние, рекурсия, асинхронность, распределённые системы, функциональное программирование, отказоустойчивость, масштабируемость.

Введение

Современные вычислительные системы требуют эффективных подходов к параллельному программированию для обработки больших объёмов данных и выполнения сложных задач. Одним из таких подходов является акторная модель, которая была впервые предложена в языке Erlang и успешно применяется в телекоммуникационной отрасли [1]. Elixir, как наследник Erlang, сохраняет все преимущества акторной модели, добавляя современные инструменты и синтаксис, что делает его привлекательным для разработки распределённых и отказоустойчивых систем [2].

Цель данной статьи – рассмотреть основные концепции акторной модели в контексте языка Elixir, изучить механизмы работы с акторами и их взаимодействие, а также продемонстрировать практические примеры использования данной модели для решения задач параллельного программирования.

Основы языка Elixir

Сопоставление с образцом. Одной из ключевых особенностей Elixir является сопоставление с образцом, которое используется для работы с данными и управления потоком выполнения программы. В отличие от традиционных языков программирования, где оператор присваивания просто присваивает значение переменной, в Elixir оператор = выполняет сопоставление с образцом [3]. Это означает, что левая часть выражения должна соответствовать правой части. Пример сопоставления с образцом показан на рис. 1.

```
1. x = 1  # x принимает значение 1
2. 1 = x  # успешное сопоставление, так как x равно 1
3. 2 = x  # ошибка, так как x не равно 2
```

Рис. 1. – Пример сопоставления с образцом в Elixir

Сопоставление с образцом также используется для работы со списками и кортежами. Пример работы со списками и кортежами представлен на рис. 2.

```
1. list = [1, 2, [3, 4, 5]]
2. [a, b, c] = list  # a = 1, b = 2, c = [3, 4, 5]
```

Рис. 2. – Пример работы со списками и кортежами в Elixir

Неизменяемость данных. В Elixir все данные неизменяемы. Это означает, что после создания объекта его нельзя изменить [4]. Вместо этого создаётся новый объект, который является результатом преобразования исходного. Например, при работе со списками. Пример работы с неизменяемыми данными показан на рис. 3.

```
1. list = [1, 2, 3]
2. new_list = [0 | list]  # новый список [0, 1, 2, 3]
```

Рис. 3. – Пример работы с неизменяемыми данными в Elixir

Неизменяемость данных упрощает управление состоянием в параллельных системах, так как исключает возможность возникновения состояний гонки.

Типы и коллекции. Elixir поддерживает широкий спектр типов данных, включая целые числа, числа с плавающей запятой, атомы, строки, кортежи, списки и словари [5]. Кортежи используются для упорядоченных коллекций значений, а списки представляют собой связанные структуры данных. Словари позволяют хранить пары ключ-значение. Пример использования типов и коллекций в Elixir представлен на рис. 4.

```
1. tuple = {1, :ok, "hello"}
2. list = [1, 2, 3]
```

Рис. 4. – Пример использования типов и коллекций в Elixir

Акторная модель в Elixir

Основные концепции. Акторная модель представляет собой подход к параллельному программированию, где независимые объекты, называемые акторами, взаимодействуют друг с другом посредством асинхронных сообщений [6]. Каждый актор инкапсулирует своё состояние и обрабатывает сообщения последовательно, что исключает возможность возникновения состояний гонки.

В Elixir акторы реализуются с помощью процессов, которые являются легковесными и могут создаваться в больших количествах [7]. Процессы в Elixir не являются процессами операционной системы, а представляют собой виртуальные процессы, управляемые виртуальной машиной Erlang BEAM.

Создание и управление акторами. Акторы создаются с помощью функции spawn или spawn_link [8]. Функция spawn принимает функцию и возвращает идентификатор процесса PID, который используется для отправки сообщений актору. Пример создания актора показан на рис. 5.

```
1. actor = spawn(fn ->
2.   receive do
3.     {:hello} -> IO.puts("HI!")
4.   end
5. end)
```

Рис. 5. – Пример создания актора в Elixir

Актор будет ожидать сообщение {:hello} и выведет «HI!» при его получении.

Сообщения и почтовые ящики. Сообщения в Elixir асинхронны и помещаются в почтовый ящик актора [9]. Актор обрабатывает сообщения последовательно, извлекая их из почтового ящика и выполняя соответствующие действия. Пример обработки сообщений актором представлен на рис. 6.

```elixir
defmodule Talker do
  def loop do
    receive do
      {:greet, name} -> IO.puts("Hello #{name}")
      {:praise, name} -> IO.puts("#{name}, you're amazing!")
      {:celebrate, name, age} -> IO.puts("HB #{name}. #{age} years old!")
    end
    loop
  end
end

pid = spawn(&Talker.loop/0)
send(pid, {:greet, "Ken"})
send(pid, {:praise, "Lilja"})
send(pid, {:celebrate, "Miles", 42})
```

Рис. 6. – Пример обработки сообщений актором в Elixir

Поддержание состояния. Акторы могут поддерживать состояние с помощью рекурсии и сопоставления с образцом [10]. Например, актор-счётчик может хранить текущее значение счётчика и обновлять его при получении сообщения. Пример поддержания состояния актора показан на рис. 7.

```elixir
defmodule Counter do
  def loop(count) do
    receive do
      {:next} ->
        IO.puts("Current count: #{count}")
        loop(count + 1)
    end
  end
end

counter = spawn(Counter, :loop, [1])
send(counter, {:next})  # Current count: 1
send(counter, {:next})  # Current count: 2
```

Рис. 7. – Пример поддержания состояния актора в Elixir

Практические аспекты работы с акторами

Двунаправленная связь. В некоторых случаях необходимо дождаться ответа от актора. Для этого используется механизм ссылок и уникальных идентификаторов. Пример двунаправленной связи между акторами представлен на рис. 8.

```
defmodule Counter do
  def loop(count) do
    receive do
      {:next, sender, ref} ->
        send(sender, {:ok, ref, count})
        loop(count + 1)
    end
  end
end

def next(counter) do
  ref = make_ref()
  send(counter, {:next, self(), ref})
  receive do
    {:ok, ^ref, count} -> count
  end
end
```

Рис. 8. – Пример двунаправленной связи между акторами в Elixir

Параллельное выполнение задач. Акторы могут использоваться для параллельного выполнения задач. Например, функция Parallel.map позволяет выполнить операцию map над коллекцией параллельно. Пример параллельного выполнения задач показан на рис. 9.

```
defmodule Parallel do
  def map(collection, fun) do
    parent = self()
    processes = Enum.map(collection, fn(e) ->
      spawn_link(fn() -> send(parent, {self(), fun.(e)}) end)
    end)
    Enum.map(processes, fn(pid) ->
      receive do
        {^pid, result} -> result
      end
    end)
  end
end
```

Рис. 9. – Пример параллельного выполнения задач в Elixir

Заключение

Акторная модель в языке Elixir предоставляет мощный инструмент для разработки параллельных и распределённых систем. Благодаря легковесным

процессам, асинхронным сообщениям и неизменяемости данных, Elixir позволяет создавать отказоустойчивые и масштабируемые приложения. В статье были рассмотрены основные концепции языка Elixir, а также практические аспекты работы с акторами, что делает её полезной для исследователей и разработчиков, интересующихся параллельным программированием.

Литература

1. Armstrong J. A History of Erlang // Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages. San Diego, California: ACM, 2007. С. 6-1—6-26. doi: 10.1145/1238844.1238850.

2. Сенлорен С., Эйзенберг Д. Введение в Elixir. М.: ДМК-Пресс, 2017. 262 с. ISBN 978-5-97060-518-9.

3. Чезарини Ф., Томпсон С. Программирование в Erlang. М.: ДМК Пресс, 2012. 488 с. ISBN 978-5-94074-617-1.

4. Хеберт Ф. Изучай Erlang во имя добра! М.: ДМК Пресс, 2015. 686 с. ISBN 978-5-97060-086-3.

5. Чезарини Ф., Виноски С. Проектирование масштабируемых систем в Erlang/OTP. М.: ДМК Пресс, 2017. 486 с. ISBN 978-5-97060-212-6.

6. Васильев Дмитрий. Знакомьтесь, Erlang // Системный администратор. 2009. № 8. С. 12-15. ISSN 1813-5579.

7. Крил Пол Функциональное программирование — друг параллелизма // Открытые системы. 2010. № 8. С. 45-50.

8. Начала работы с Erlang // RSDN Magazine. 2006. № 3. URL: rsdn.org/magazine/2006/03/erlang.

9. Armstrong Joe Erlang // Communications of the ACM. 2010. Т. 53, № 9. С. 68-75. doi: 10.1145/1810891.1810910.

10. Aronis S., Papaspyrou N., Roukounaki K., Sagonas K., Tsiouris Y., Venetis I.E. A Scalability Benchmark Suite for Erlang/OTP // Proceedings of the

Eleventh ACM SIGPLAN Workshop on Erlang Workshop. Copenhagen, Denmark: ACM, 2012. C. 33-42. doi: 10.1145/2364489.2364495.

References

1. Armstrong J. Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages. San Diego, California: ACM, 2007. pp. 6-1—6-26. doi: 10.1145/1238844.1238850.

2. Senloren S., Eisenberg D. Vvedenie v Elixir [Introducing Elixir]. Moskva: DMK-Press, 2017. 262 p. ISBN 978-5-97060-518-9.

3. Cesarini F., Thompson S. Programmirovanie v Erlang [Erlang Programming]. Moskva: DMK Press, 2012. 488 p. ISBN 978-5-94074-617-1.

4. Hébert F. Izuchay Erlang vo imya dobra! [Learn You Some Erlang for Great Good!]. Moskva: DMK Press, 2015. 686 p. ISBN 978-5-97060-086-3.

5. Cesarini F., Vinoski S. Proektirovanie masshtabiruemykh sistem v Erlang/OTP [Designing for Scalability with Erlang/OTP]. Moskva: DMK Press, 2017. 486 p. ISBN 978-5-97060-212-6.

6. Vasiliev Dmitry Sistemnyj administrator. 2009. № 8. pp. 12-15. ISSN 1813-5579.

7. Kril Paul Otkrytye sistemy. 2010. № 8. pp. 45-50.

8. RSDN Magazine. 2006. № 3. URL: rsdn.org/magazine/2006/03/erlang.

9. Armstrong Joe Communications of the ACM. 2010. Vol. 53, no. 9. pp. 68-75. doi: 10.1145/1810891.1810910.

10. Aronis S., Papaspyrou N., Roukounaki K., Sagonas K., Tsiouris Y., Venetis I.E. Proceedings of the Eleventh ACM SIGPLAN Workshop on Erlang Workshop. Copenhagen, Denmark: ACM, 2012. pp. 33-42. doi: 10.1145/2364489.2364495.

Дата поступления: 29.12.2024

Дата публикации: 2.02.2025