本書は、小社刊の以下の刊行物をもとに、大幅に加筆と修正を行い書籍化したものです。

・『WEB+DB PRESS』Vol.127 特集2「作って学ぶPhoenix――Elixirによる高速なWeb開発！」

本書で利用しているソフトウェアは、執筆時点の最新である次のバージョンで動作確認を行っています。

・Elixir 1.15.7-otp-26
・Erlang/OTP 26.1.2
・Phoenix 1.7.10
・Ecto 3.11.1
・Phoenix LiveView 0.20.1
・Nx 0.6.4
・Axon 0.6.0
・Nerves 1.10.4

環境や時期により、手順・画面・動作結果などが異なる可能性があります。

はじめに

本書は、プログラミング言語Elixirについて、基本的な文法や機能についてひととおり解説するとともに、Webアプリケーションはもとより、機械学習やIoTといった領域における実践的な応用についても解説します。そのため、Elixirのことをあまりよく知らない方々にはもちろんのこと、すでにElixirに馴染みのある方々も含む幅広い読者の皆様にお楽しみいただける書籍になっていると自負します。

プログラミング言語 Elixir

本書が紹介するプログラミング言語Elixir[注1]は、José Valim（ジョゼ・ヴァリム）氏を中心とするコミュニティによって活発に開発されている関数型言語です。2011年に開発が開始され、2012年に0.5.0として公開された[注2]後、2014年にバージョン1.0がリリースされた[注3]比較的若い言語です。本書の執筆時点では、バージョン1.15.7が最新の安定版としてリリースされています[注4]。

Elixirは、低遅延で高い可用性を要求される分散システムの構築と運用を目的とするErlang VM（Virtual Machine、仮想マシン）上で動作する言語です。その文法はRubyから大きく影響を受けており、多くのWeb開発者にとって馴染みやすいでしょう[注5]。

Elixir の評判

プログラミング言語としてのElixirそのものについては、本書の第1章から第6章までで解説します。ここでは、Elixirを利用している人々の観点から、現状を概観してみましょう。

■ 百花繚乱のプログラミング言語

ご存じのとおり、世の中にはたくさんのプログラミング言語が存在します。Webアプリケーション開発においては、RubyやPHP、Javaといった言語が広く使われてきました。昨今では、Webフロントエンドのみならずバックエンド

注1　https://elixir-lang.org/

注2　https://elixir-lang.org/blog/2012/05/25/elixir-v0-5-0-released/

注3　https://elixir-lang.org/blog/2014/09/18/elixir-v1-0-0-released/

注4　本書で説明する内容も、バージョン 1.15.7 を前提にしています。

注5　作者の José Valim 氏は、Elixir を作るうえで影響を受けた言語のトップ３は Erlang、Ruby、Clojure であると述べています（https://cognitect.com/cognicast/120）。Elixir は、堅牢な Erlang VM の上で動く、Ruby のように親しみやすい、JVM（*Java Virtual Machine*、Java 仮想マシン）に対する Clojure のような言語であると言えるでしょう。

の開発においても、TypeScriptの躍進が目立つところです。また、機械学習・AI領域においては、関連ライブラリやエコシステムが充実しているPythonが広く使われています。さらに、アプリケーションやシステムの複雑化に伴い静的型付けの意義があらためて見直され、GoやRustといった言語の人気がますます高まっています。

■ 書籍に見るEilxirの人気度

プログラミング言語の人気度をはかるのは難しいですが、どれほど多くの書籍が出版されているかを指標の一つとしてとらえることができるでしょう。TypeScript、Python、Go、Rustについて、ここ数年、とりわけ多くの書籍が刊行されているように思われます。Elixirについてはどうでしょうか？ 2020年以降、日本語の書籍としては『プログラミング Elixir（第2版）』[注6]、『Elixir実践ガイド』[注7]が刊行されました。刊行点数としてはけっして多いわけではありませんが、着実に裾野を広げつつあることは確かでしょう。本書が、そうした流れを後押しすることを、筆者一同願ってやみません。

■ 国外におけるElixirの状況

国外の状況はどうでしょうか？ 前述の『プログラミング Elixir（第2版）』の著者Dave Thomas氏が運営するPragmatic Bookshelf[注8]を中心に、Elixirという名前を冠する書籍が多数刊行されています[注9]。また、書籍ではありませんが、プログラミングに関する質問・回答を通じてナレッジを共有するWebサイトStack Overflow[注10]が2022年に開発者に対して実施した調査では、Elixirが「最も愛されている（most loved）」言語として2位にランクイン[注11]したことが話題となりました。Elixirが、国外の開発者の間で一定の地歩を占めているのは確かであるようです。

■ 国内の盛り上がりを支えるコミュニティ

国内においてはまだまだメジャーな言語とは言いがたいElixirですが、コミュニティの盛り上がりはすでにメジャー級であると言えます。その証拠に、

注6　Dave Thomas著、笹田耕一・鳥井雪訳『プログラミングElixir（第2版）』オーム社、2020年

注7　黒田努著『Elixir実践ガイド』インプレス、2021年

注8　https://pragprog.com/

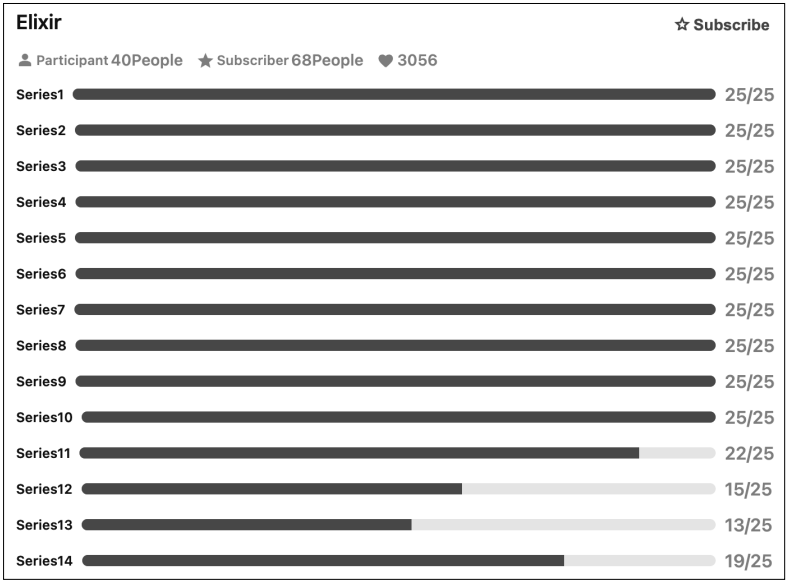
注9　Pragmatic BookshelfのWebサイトから「Elixir」で検索してみてください。https://pragprog.com/search/?q=Elixir

注10　https://ja.stackoverflow.com/

注11　https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted

本書の執筆陣も多く参加したElixir Advent Calendar 2023[注12]では、他言語と比較しても圧倒的なボリュームの記事数が投稿されました（**図1**）。

● 図1　Qiita Advent Calendar 2023のプログラミング言語カテゴリ[注13]におけるElixirに関する記事の投稿数



そうした精力的な活動を支えているのが、国内に30ほど存在するとされるElixirコミュニティです[注14]。冒頭で述べたとおり、Elixir自体がコミュニティによって開発されている言語であるのと同様に、利用者もたくさんのコミュニティを通じてつながり、活発に情報交換しています。読者の皆様にも、ぜひご参加することをお勧めします。

■ Elixirのポテンシャル

なぜElixirがそんなにも熱狂的な盛り上がりを見せているのでしょうか？ その一つとして、Elixirやその基盤であるErlang/OTPの持つ大きなポテンシャルが、人々を強く惹きつけているからという理由が挙げられるでしょう。

注12　https://qiita.com/advent-calendar/2023/elixir

注13　https://qiita.com/advent-calendar/2023/categories/programming_languages

注14　どんなコミュニティがあるかについては、「Elixir コミュニティの歩き方～国内オンライン編～」(https://speakerdeck.com/elijo/elixirkomiyunitei-falsebu-kifang-guo-nei-onrainbian)をご覧ください。

■ Elixir の利用事例

ElixirやErlang/OTPの近年における大規模な利用事例としては、国内ではゲーム業界における事例が目立つようです[注15]。また、チャットプラットフォームのDiscordでも、Elixirが大規模に利用されているようです[注16]。大量のコネクションとトラフィックを扱う必要のあるシステムを構築するのに、Erlang VMが向いているからでしょう。また、本書でも詳しく解説するWebアプリケーションフレームワークPhoenix Framework[注17]が早くから登場したことにより、Webアプリケーション領域においても利用が進んでいます。

■ 広がりを見せる適用領域

深層学習の登場以降、機械学習・AIの著しい発展が世の中を騒がせ続けています。2021年2月にリリースされたNx[注18]を皮切りに、それらの領域におけるElixirの適用可能性が切り開かれてきました。Nxは、機械学習に必須となる高速な数値計算のためのライブラリです。今では、ニューラルネットワークを簡潔に表現できるAxon[注19]や、事前学習済みモデルを簡単に利用できるBumblebee[注20]といったライブラリもあります。また、Elixirのコードをインタラクティブに記述・実行できるLivebook[注21]も活発に開発されています。

主にサーバ上で動作する上述のような領域以外でも、Internet of Things（IoT）でのElixirの利用も進んでいます。ElixirによるIoTデバイス向けのプラットフォームのNerves[注22]を用いると、Raspberry PiシリーズやBeagleBoneシリーズによって堅牢かつ高機能なIoTデバイスを開発できます。また、Elixir Desktop[注23]やLiveView Native[注24]のように、PCやモバイル端末上で動作する

注15 ・「ロマサガRSにおけるElixirサーバー開発実践～生産性を上げてゲームの面白さに注力～」 https://speakerdeck.com/elixirfest/romasakars-niokeru-elixir-sahakai-fa-shi-jian-sheng-chan-xing-woshang-ketekemufalsemian-bai-sanizhu-li
・「Erlang/OTPとejabberdを活用したNintendo Switch(TM)向けプッシュ通知システム「NPNS」の開発事例」 https://speakerdeck.com/elixirfest/otp-to-ejabberd-wohuo-yong-sita-nintendo-switch-tm-xiang-ke-hutusiyutong-zhi-sisutemu-npns-false-kai-fa-shi-li

注16 「Real time communication at scale with Elixir at Discord」 https://elixir-lang.org/blog/2020/10/08/real-time-communication-at-scale-with-elixir-at-discord/

注17 https://www.phoenixframework.org/

注18 https://github.com/elixir-nx/nx

注19 https://github.com/elixir-nx/axon

注20 https://github.com/elixir-nx/bumblebee

注21 https://livebook.dev/

注22 https://nerves-project.org/

注23 https://github.com/elixir-desktop/desktop

注24 https://native.live/

アプリケーションをElixirによって開発できるようになってきています。さらには、すでに触れたElixirのネットワークやWebアプリケーションにおける強みを活かして、デバイスからクラウドまでを一気通貫にElixirを用いて開発する取り組みも進んでいます[注25]。

本書の構成

Elixirの多岐にわたる魅力をできるだけ幅広く紹介するために、本書の構成もこだわり抜きました。

まず、第1章から第6章にかけて、Elixirを楽しむためには最低限これだけは知っておいてほしいという内容にしぼって、簡潔に文法や機能を紹介します。次に、第7章から第11章にかけて、ElixirでWebアプリケーションを開発してみることでさっそくElixirに「実践入門」していきます。その過程を通じて、ElixirによるWebアプリケーション開発に欠かせない、Webアプリケーションフレームワークの Phoenix Framework（と、目玉機能の LiveView）、RDBMS（Relational DataBase Management System）を扱うライブラリの Ectoの使い方を学びます。さらに、第12章から第19章にかけて、機械学習・AI領域およびIoT領域において、具体的なアプリケーションを作っていきます。

このプロセスを通じて、熱狂的な盛り上がりを生み出しているElixirの大きなポテンシャルを、きっと感じていただけることと思います。読者の皆様がElixirの持つ広大な可能性に触れる一助に本書がなることを、筆者一同願っています。

2024年1月
筆者を代表して　栗林 健太郎

●サポートページ

本書のサンプルコードは、下記サポートサイトからダウンロードできます。正誤情報なども掲載します。

https://gihyo.jp/book/2024/978-4-297-14014-4/support

注25　たとえば、「Embedded and cloud Elixir for grid-management at Sparkmeter」（https://elixir-lang.org/blog/2023/03/09/embedded-and-cloud-elixir-at-sparkmeter/）のような事例があります。

謝　辞

本書の執筆にあたって、森正和さん（@piacere_ex）、衣川亮太さん（@pojiro3）（括弧内はXアカウント名）にレビューをご担当いただきました。単なる用語や解説のレビューにとどまらず、読者にとってわかりにくいところや改善案などを忌憚なくコメントいただき、本書をより良い内容にすることができました。ありがとうございます。また、本書の編集は稲尾尚徳さん、池田大樹さんに担当いただきました。7名という筆者の原稿や意見をまとめていくのは非常に大変だったと思います。根気よく付き合っていただきまして、誠にありがとうございます。ただし、本書に記載した内容について誤りがある場合、その文責は執筆を担当した筆者らにあります。

やはり本書は、プログラミング言語Elixirに関わる多くの開発者たちがいてこそ完成しました。本書で取り上げるフレームワークやライブラリのそれぞれの代表者として、Elixirの作者であるJosé Valim氏、PhoenixのChris McCord氏、NxとAxonのSean Moriarity氏、NervesのFrank Hunleth氏に深く感謝申し上げます。彼らの愛と情熱に溢れたフレンドリーなリーダーシップがあったからこそ、比較的若いにもかかわらず爆発的に成長を遂げているElixirの現在のエコシステムがあります。そして、Elixirはやはりプログラミング言語Erlangなくしてはこの世に生まれることはなかったでしょう。ErlangおよびErlang VMの共同設計者であり長らく開発を牽引していたJoe Armstrong氏は、残念ながら2019年にこの世を去っています。この場をお借りして同氏への哀悼の意を示すとともに、すばらしいプログラミング言語を遺してくれたことに深く御礼申し上げます。

国内の盛り上がりを支えるコミュニティを牽引するそれぞれのオーガナイザー、そしてそれぞれのイベントを盛り上げてくれている参加者の皆さまにも感謝いたします。第19章で取り上げているIoTアプリケーションのサンプルは、NervesJPで実施したハンズオンイベントの内容をおおいに参考にしています。この教材を作成されたのは、著者らのほかに、衣川亮太さん（@pojiro3）、三宅泰宏さん（@etcinitd）、菊池豊さん（@kikuyuta）、西内一馬さん（@nishiuchikazuma）、大崎充博さん（@rigutter）の協力によります。ありがとうございます。最後に、Elixirの成長に貢献している世界中のすべてのアルケミスト[注1]に深く感謝申し上げます。

注1　Elixirとは、錬金術で用いられる不老不死をもたらしてくれる霊薬を意味します。そのため、プログラミング言語Elixirの使い手はアルケミスト（錬金術師）と呼ばれることがあります。

執筆者と担当章

章	タイトル	著者名
はじめに		栗林 健太郎
第 1 章	Elixir 小史	大原 常徳
第 2 章	Elixir の基礎	大聖寺谷 一樹
第 3 章	基本的な型とパターンマッチ	大聖寺谷 一樹
第 4 章	モジュール	山内 修
第 5 章	Mix を使った Elixir プロジェクトの開発	山内 修、大原 常徳（一部）
第 6 章	並行プログラミング	大原 常徳
第 7 章	Phoenix の概要	齋藤 和也
第 8 章	Ecto によるデータベース操作	山内 修
第 9 章	phx.gen.auth による認証	齋藤 和也
第 10 章	LiveView によるフロントエンドの開発	齋藤 和也
第 11 章	実践的な Web アプリケーションの開発	齋藤 和也
第 12 章	行列演算ライブラリ Nx の概要	隆藤 唯章
第 13 章	Axon の概要と機械学習システム開発の進め方	隆藤 唯章
第 14 章	機械学習向けのライブラリ	隆藤 唯章
第 15 章	実践的な Axon アプリケーションの開発	隆藤 唯章
第 16 章	Nerves の概要	高瀬 英希
第 17 章	Nerves での開発の進め方	高瀬 英希
第 18 章	Elixir Circuits によるモジュールの制御	高瀬 英希
第 19 章	実践的な IoT アプリケーションの開発	高瀬 英希
あとがき		栗林 健太郎

目次■ Elixir 実践入門――基本文法、Web 開発、機械学習、IoT

第 14 章

機械学習向けのライブラリ 289

第 15 章

実践的な Axon アプリケーションの開発 315

第 16 章

Nerves の概要 329

第 17 章

Nerves での開発の進め方 339

第 18 章

Elixir Circuits によるモジュールの制御 355

第 1 章

Elixir 小史

本書では、プログラミング言語 Elixir について、幅広い領域における利用法を解説します。すなわち、Elixir の言語としての基礎から始まり、そのエコシステムが提供する独特の Web アプリケーション開発スタイルに加えて、IoT や機械学習開発についても扱います。

Elixir は、なぜそのような幅広い領域において用いられる言語となったのでしょうか。本章では、プログラミング言語 Elixir の特徴を簡単に紹介したあと、以下の内容について解説し、Elixir が持つポテンシャルについての理解を目指します。

- **Elixir の実行基盤である Erlang/OTP[注1] の歴史と特徴**
- **Erlang/OTP を基盤に Elixir が登場した背景**
- **Elixir の誕生から現在までの変遷**

Elixir 言語の特徴

Elixir は、José Valim 氏によって開発されているプログラミング言語です。最大の特徴は、プログラミング言語 Erlang の VM（*Virtual Machine*、仮想マシン）上で動作し、Erlang のモジュールを直接利用できることです。Erlang VM 上で動作することから、Elixir は以下のような特徴を備えています。

- 耐障害性
- 高可用性
- 分散システム

Erlang は、後述する OTP というさまざまな分散アプリケーションの実装パターンを抽象化したフレームワークを持っています。Erlang はプログラミング言語であり、OTP は Erlang のフレームワークですが、これらは密接に関連していて、一緒に利用されるのが一般的です。そのため Erlang/OTP という表記がよく用いられます。本章では、プログラミング言語の文脈では「Erlang」、プログラミング環境の文脈では「Erlang/OTP」、言語の処理系の文脈では「Erlang VM」という表現を用います。

注 1　https://www.erlang.org/

したがってElixirは次のErlang/OTPによるプログラミングのパラダイムを受け継いでいます。

- 関数型
- アクターモデル
- プロセス間のメッセージパッシングによる並行処理

本節ではこれらの特徴的な言語機能について解説していきます。

モダンな関数型言語

Elixirは比較的新しい言語で、以下のような現代的なプログラミング言語の要素を備えています。

- ポリモフィズム
- マクロ／メタプログラミング
- パターンマッチ
- テスティング
- ビルドツール
- ライブラリエコシステム

また、プログラミング言語において、生成、代入、引数や戻り値としての受け渡しといった操作を制限なしに行える対象を第一級オブジェクトと言います。Elixirは関数型言語でありElixirの関数は第一級オブジェクトですが、Erlang VM上で生成されるプロセス上で動作することから、ほかの関数型言語とは少し毛色が異なります。

詳細はのちの章で解説しますが、Elixirはこれらの特徴や機能を持つことから、小さなスクリプトから大規模システムのサーバまで無理なく記述できるのです。

アクターモデルによる並行処理と耐障害性

並行処理に対するアプローチは、プログラミング言語によってさまざまです。マルチスレッド、ソフトウェアトランザクショナルメモリ、並行論理プログラミングなど多くの手法があります。Elixirは、並行処理の実現に

アクターモデルを採用しています。

Elixir（の基盤になるErlang/OTP）はそれぞれ独立して動作するプロセス上で関数が動作し、複数のプロセス間でメッセージをやりとりして処理が実行されます。プロセスは受信したメッセージによって次の動作を決定し、これらの一連の振る舞いは逐次ではなく並行的、非同期的に実行されます。

またElixirは耐障害性が高い言語として知られています。この耐障害性は次の2つの特徴によるところが大きいです。

- Supervisorによるプロセス監視
- Erlang VMの堅牢性

OTPの中のSupervisorというプロセスの監視パターンを利用することで、プロセスがクラッシュした場合でもプロセスを適切に再起動させることができます。

また、Erlang VMは非常に堅牢で、安定してプログラムを実行できます。筆者は長年プロダクション環境でElixir/Erlangのサービスを運用していますが、Erlang VMがクラッシュするトラブルに遭遇したことはほとんどありません。

大規模システムでの利用

前節でOTPは分散アプリケーションの実装パターンを集めたフレームワークであると解説しました。このOTPを利用することで、複雑な分散処理を少量のコードで実装することが可能となります。Elixirが大規模で複雑な分散処理を実装するのに適している理由は、このOTPを利用できるところにあります。

Erlang/OTP ── Elixir の実行基盤

前節ではElixirはErlang VM上で動作するプログラミング言語と解説しました。本節ではErlang/OTPの特徴を解説していきます。

Erlang とはどんな言語か

Erlang はエリクソン社が開発した並行処理を簡潔に記述できるプログラミング言語です。もともとは電話交換機のように高い信頼性が要求されるプログラムのために設計されたことから、以下の特徴を持っています。

- 耐障害性
- 分散処理
- 無停止稼働

無停止稼働の特徴により、稼働中のシステムを停止することなくプログラムを変更するホットコードスワップを行うことができます。

これらの機能により、Erlang は高い信頼性が求められるシステムを無停止で稼働させることができるのです。

さらに先述のとおり、Erlang は以下のプログラミングのパラダイムを持っています。

- 関数型
- アクターモデル
- プロセス間のメッセージパッシングによる並行処理

Erlang は処理をモジュール、関数、引数の組み合わせで記述します。なお、モジュール（Module）、関数（Function）、引数（Argument）の頭文字をとって、これらの組み合わせを MFA と呼びます。Erlang VM 上に、これらの関数を処理するプロセスが生成され、複数のプロセスがそれぞれ独立して動作します。

Erlang が得意なこと・苦手なこと

Erlang は耐障害性、無停止稼働の特徴を持っていることから、「落ちない」システムを記述するのに優れています。たとえば広告サーバや常時稼働を前提としたミドルウェアなどの、高い SLA（*Service Level Agreement*、サービス水準合意）を求められるシステムを記述することが得意です。また、大量のプロセスを手早く起動できるため、並列処理も得意とされています。

その一方、Erlang は C や C++ に比べると処理速度は速くありません。リストやマップといったデータ構造、浮動小数点数のデータ型を Erlang は持っ

ていますが、CやC++に比較するとオーバーヘッドが大きく高速に処理を行うことはできません。したがって、数値計算や画像処理などを行うのには向いていません。

なお、現在のElixirは、後述する行列演算ライブラリであるNxと、そのバックエンドとしてネイティブな計算処理を賄うライブラリを利用することで、数値計算や画像処理に向かない弱点を克服しています。これらの詳細は第12章～第15章で解説します。

関数型言語としてのErlang

Erlangは関数型言語です。関数を第一級オブジェクトとして扱えるので、関数自身を引数として別の関数に渡したり、値として扱ったりできます。また、Erlangはアクターモデルによる処理を実現できるよう設計されており、関数はアクターであるプロセス上で実行されます。これがほかの関数型言語と大きく異なる点です。

ほかのプログラミング言語とErlangのパラダイムを比較する際、関数型言語かどうかよりも、アクターモデルベースかどうかの違いのほうが大きいと筆者は考えています。

コンパイラ言語としてのErlang

Erlangはコンパイル言語です。Erlangのソースコードであるerlファイルがコンパイルされると、モジュールごとにbeamファイルが生成されます。beamファイルはErlang VMのマシン語に変換されたバイナリで、Erlang VM上で実行されます。また、コンパイル時にデバッグ情報を付与することでbeamファイルにデバッグ情報を含めることができます。なお、Erlangのコンパイルや逆コンパイル、逆アセンブルなどの機能は、Erlangにバンドルされた標準ライブラリの関数として提供されます。

プロセスとアクターモデル

本項ではErlangのプロセスとアクターモデルについて解説します。

一般的なプログラミング言語は、プログラムのエントリポイントであるmain関数から処理が始まり、分岐や関数呼び出しを繰り返しながら処理を進めていきます。一方Erlangのプログラムは、プログラム起動時に生成さ

れるプロセスが別のプロセスを生成し、さらに別のプロセスを生成するといった形で、多くのプロセスがメッセージのやりとりをしながら処理を進めていきます。

このように一般的なプログラミング言語のプログラムの実行パラダイムは1次元的なのに対して、Erlangのプログラムの実行パラダイムは多次元的だと言えるでしょう。

Erlang におけるプロセス

紛らわしいことに、ErlangのプロセスとOSのプロセスはまったくの別物です。Erlangの「プロセス」は、プロセスごとの識別ID（プロセスID、PID）を持ち、内部に受信したメッセージを保持するキュー（メッセージボックス）を持ちます。プロセスを生成するのに必要なメモリはデフォルトで数百ワード、起動にかかる時間は数マイクロ秒と非常に小さく、大量のプロセスを極めて短い時間で生成できます。これらのプロセスはそれぞれ並列に動作し、同時に別の計算や処理を行うことができます。

アクターモデルによるプログラムデザイン

複数のプロセス間では以下のことが可能です。

- プロセスにメッセージを送信する
- プロセスからメッセージを受信する
- プロセスが別のプロセスを生成する
- プロセスが別のプロセスをリンクする
- プロセスが別のプロセスをモニタする

メッセージの送受信やプロセスの生成は簡単にイメージできますが、「リンク」または「モニタ」はイメージがつかみにくいかもしれません。

あるプロセスが別のプロセスと「リンク」されると、プロセスどうしが互いを監視するようになります。これによって、一方のプロセスが死んだとき、もう一方のプロセスに対して終了メッセージを送信できます。

「リンク」がプロセス間の相互監視だったのに対して、一方通行の監視を行うのが「モニタ」です。「リンク」と「モニタ」には相互に監視し合うのか、片方だけを監視するのかという違いがあります。あるプロセスが別のプロセスを「モニタ」しているケースを考えてみましょう。監視先のプロセ

スが死んだとき、監視元のプロセスに対して終了メッセージが送信されます。一方、監視元のプロセスが死んだとき、監視先のプロセスに対して終了メッセージは送信されません。つまり、「モニタ」は監視先のプロセスだけを監視するしくみと言えるでしょう[注2]。

プロセスはこれらの機能を使って監視し合い、より複雑な並行処理を実現します。

OTP による Erlang の拡張

前項では、Erlangのプロセスによるプログラムデザインについて解説しました。複数のプロセスが協調し合いながら処理を進めることができるので、並行処理を簡単に記述できます。しかし、複雑な並行処理を実装する場合には多くのことを考慮する必要があります。本項で解説するOTPを利用することで、この複雑さをラップしてシンプルに複雑な並行処理を実装できるようになります。

OTP ── Open Telecom Platform

OTPはOpen Telecom Platformの略で、さまざまな並列処理を抽象化したフレームワーク・ライブラリ群です。また、プログラミング言語Erlangについて言及する際、「Erlang」ではなく「Erlang/OTP」と一緒に記載されることが多いです。Erlangのプログラミングにおいて、OTPは非常に重要な位置を占めています。

OTPで実現可能な並列処理のパターンの詳細は第6章で解説しますが、これらのパターンを組み合わせることで、いわゆる「落ちない」システムを簡単に構築できます。

ビヘイビアによる設計の抽象化

OTPでは、並列処理の定番のパターンを「ビヘイビア」として形式化しています。ビヘイビアは次の2つのモジュールから構成されます。

- ビヘイビアモジュール
  OTPがライブラリとして提供。gen_xxxという名前で定義

注2 詳しくは第6章の「プロセスの監視」（122ページ）で解説します。

• コールバックモジュール
　ユーザーが定義

ビヘイビアモジュールはOTPが提供するライブラリとして実装されており、プログラマーはこれらのビヘイビアをコールバックモジュール内で宣言します。

クライアント／サーバの処理を提供するgen_serverビヘイビアを例に見てみましょう。なお以降のソースコードはErlangのソースコードです。シンタックスが少し特徴的ですが、「こういうものなのだな」と思って読み進めてください。

最初に、コールバックモジュール内でどのビヘイビアの規約に従うかを宣言します。

モジュールとビヘイビアの宣言
```
-module(ch1).
-behaviour(gen_server). %% gen_serverビヘイビアの宣言
```

次に、必要なコールバック関数をエクスポートします。

コールバック関数のエクスポート
```
-export([start_link/0]).
-export([alloc/0, free/1]).
-export([init/1, handle_call/3, handle_cast/2]).
```

最後に、それぞれエクスポートするコールバック関数を実装します。

コールバック関数の実装
```
start_link() ->
  gen_server:start_link({local, ch1}, ch1, [], []).

alloc() ->
  gen_server:call(ch1, alloc).

free(Ch) ->
  gen_server:cast(ch1, {free, Ch}).

init(_Args) ->
  {ok, channels()}.

handle_call(alloc, _From, Chs) ->
  {Ch, Chs2} = alloc(Chs),
  {reply, Ch, Chs2}.

handle_cast({free, Ch}, Chs) ->
  Chs2 = free(Ch, Chs),
  {noreply, Chs2}.
```

このコールバックモジュールをコンパイルして実行すると、gen_serverの振る舞いをするプログラムが実行可能になります。代表的なビヘイビアとして以下のビヘイビアがよく利用されます。

- GenServer
- Supervisor

これらのビヘイビアは第6章で解説します。

Elixir の誕生

ElixirのGitHubリポジトリの一番古いコミットは2011年1月9日で、約3年半の開発期間のあと、バージョン1.0が2014年9月にリリースされました。本書の執筆時点ではElixirの最新バージョンは1.15.0になっています。

Elixir が作られた背景

Elixirの開発が始まった2011年は、並行処理の重要性が高くなっていた時期でした。当時よく利用されていたプログラミング言語はオブジェクト指向言語が主流で、並行処理をこれらの言語で実現するには、スレッド間の状態を管理する必要がありました。そのためにロックやミューテックスといった複雑なしくみが必要という問題がありました。

そこで作者のJoséはこれらの課題解決のアプローチとして関数型言語に注目しました。オブジェクトの状態を管理するのではなく、関数を実行して状態が変更された新しいオブジェクトを新たに生成することで、シンプルに並行処理を表現できると考えたのです。

そして、さまざまな関数型言語を研究した結果、Erlang/OTPが採用されました。しかしErlang/OTPではUnicodeの対応がなかったり、メタプログラミングができなかったりなどの制約が多く、のちにElixirとなる新しいプログラミング言語の開発を開始するに至りました。

並行処理へのアプローチ

ErlangにOTPがバンドルされているように、ElixirにもOTP相当の並行処理のパターンを抽象化したビヘイビアモジュールが標準ライブラリとし

てバンドルされています。これらのビヘイビアモジュールを使うことで、Elixir は Erlang/OTP 同様に並行処理をコールバック関数の実装だけで実現できます。

BEAM 言語としての Elixir

コンパイル時に Erlang VM の実行バイナリである beam ファイルを生成する言語を BEAM 言語と呼びます。Elixir も BEAM 言語の一つです。

BEAM 言語の種類は多く、Elixir 以外では**表 1.1** のような言語があります。

● 表1.1 主要なBEAM言語

BEAM 言語	公式サイト
LFE	https://lfe.io/
Efene	http://efene.org/
luerl	https://github.com/rvirding/luerl
clojerl	https://www.clojerl.org/
purerl	https://purerl.fun/
Gleam	https://gleam.run/

Elixir の特徴的なシンタックス

Elixir が作られた背景にあるように、Elixir はメタプログラミングが可能です。また Ruby に近い見た目のシンタックスが特徴的で、Erlang/OTP に比べて読みやすいプログラミング言語です。

本章では詳細な説明は割愛しますが、Elixir のシンタックスで特徴的なものに以下があります。

- パイプ演算子（|>）
- パターンマッチ
- メタプログラミング

Elixir と Erlang/OTP との関係

Elixir は BEAM 言語の一種ですので、beam ファイルをコンパイル後に出力して、Erlang VM 上で動作します。また、Elixir のコンパイラなどのコアモジュールは、Erlang/OTP で実装されています。ですので、Elixir 本体は Erlang/OTP を使って実装されていると言えるでしょう。

Elixir のプログラム実行

Elixir のソースコード（ex ファイル）をコンパイルすると、beam ファイルが生成されます。これらの beam ファイルは、Erlang でコンパイルした beam ファイルと同じ形式で、Erlang VM 上で同様に実行可能です。そのため、Elixir から Erlang/OTP のモジュールや関数をオーバーヘッドなしで利用できます。つまり、Erlang/OTP のライブラリを Elixir は再利用できるのです。

OTP から見た Elixir アプリケーション

Elixir にはコンパイラや REPL（*Read-Eval-Print Loop*）のインタフェースといったプログラムがバンドルされています。これらのプログラムは Erlang/OTP を使って実装されています。Erlang/OTP で実装されているので、当然 OTP（の並行処理パターン）を使って Erlang VM 上で動作します。実際のところ、Elixir コンパイラや REPL は Supervisor ビヘイビアや GenServer ビヘイビアなどのさまざまな OTP ビヘイビアを使って実装されています。

Elixir の発展

Elixir がリリースされてから 2023 年で 11 年になります。リリース当初はライブラリが少なく適用領域も限られていましたが、近年では IoT や機械学習、SPA（*Single Page Application*）やスマートフォンネイティブアプリ開発などの領域でも採用されるようになってきました。

Elixir のエコシステム

Elixir のライブラリは、Hex と呼ばれるパッケージマネージャで管理されています。ライブラリを探す際は hex.pm[注3] を参照するとよいでしょう。hex.pm では Elixir のほかに Erlang/OTP で利用できるライブラリもホスティングされています。

また、Elixir には Mix と呼ばれるビルドツールがバンドルされているので、Mix を使うことで簡単に hex.pm からライブラリをインストールできます。

注 3　https://hex.pm/

Elixir のコミュニティ

Elixirのコミュニティは全世界に数多くあります。GitHubで運用されているElixirの開発リポジトリのWikiには、主要なミートアップ[注4]やカンファレンス[注5]の一覧が掲載されています。

また、日本でも数多くのミートアップやカンファレンスが開催されています。2019年のコロナパンデミック以降はオンラインでの開催が主流になっていますが、近年ではオフラインイベントが復活してきています。

Elixir の持つポテンシャル

Elixirは並行処理を記述しやすい言語として開発が始まりましたが、近年ではさまざまな分野で利用されるようになってきました。

ネットワークアプリケーション

Elixirの基盤となっているErlang/OTPは、信頼性の高いシステムを構築する言語として開発されました。信頼性が求められるシステムは多くありますが、その中でも複数のサーバ上で動作するネットワークアプリケーションは高い可用性が求められる分散システムですので、Elixirと相性が良いシステムと言えるでしょう。

ネットワークアプリケーションの構造

ネットワークアプリケーションは数多くありますが、その構造としては主に以下の特徴があります。

- ネットワーク上の分散された複数のノード上で動作する
- サーバ／クライアントの構造を持つ

また、ネットワークを通して利用されるため、大量のコネクションやデータを扱える必要があります。

注4 https://github.com/elixir-lang/elixir/wiki/Meetups
注5 https://github.com/elixir-lang/elixir/wiki/Conferences

Elixir によるネットワークアプリケーション実装のメリット

ネットワークアプリケーションである Web サーバを考えてみましょう。Web サーバに求められる要件として以下のようなものがあり、これらはまさに Erlang/OTP が得意とするものです。

- 高負荷に強い
- 高い可用性
- 大量のコネクションをさばくことができる

したがって、Elixir を使うことで、Erlang/OTP の特徴を活かしたネットワークアプリケーションを簡単に実装できます。

Elixir で実装されたネットワークアプリケーション

Elixir や Erlang/OTP でよく利用される Web サーバとして Cowboy[注6] があります。Cowboy は Erlang/OTP で実装された Web サーバですが、Elixir でも利用でき、Elixir の Web アプリケーションフレームワークである Phoenix でも採用されています。

そのほかに、Elixir で実装された Web サーバとして Bandit[注7] があります。Cowboy と比較して性能が高く、注目度が高い Web サーバです。

Web アプリケーション

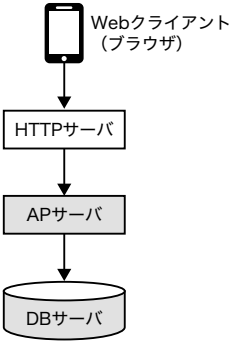
Elixir で Web アプリケーションを開発する場合、Phoenix を利用することがほとんどです。Phoenix は Rails のようなフルスタック Web アプリケーションフレームワークで、Elixir のキラーライブラリと言えるでしょう。

Web アプリケーションの構造

一般的な Web アプリケーションの構造は**図 1.1** のようになります。

注 6　https://github.com/ninenines/cowboy
注 7　https://github.com/mtrudel/bandit

● **図1.1 Webアプリケーションの構造**

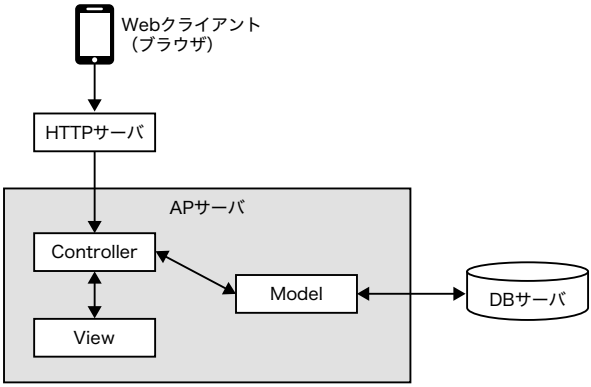


Webクライアントはブラウザで、HTTPサーバはHTTPリクエストを受け取り、APサーバはHTTPサーバからのリクエストを処理し、DBサーバはデータの永続化を行います。

Webアプリケーションフレームワーク

Webアプリケーションフレームワークとは、その名のとおりWebアプリケーションを開発するためのフレームワークです。PhoenixはMVCモデルを採用しており、**図1.2**のような構造を持ちます。

● **図1.2 MVC**



MVCはModel、View、Controllerの頭文字を取ったもので、Webアプリケーションの構造を表しています。Controllerはリクエストを受け取り、

Model と View を呼び出します。Model はデータの永続化を行い、View は HTML などのレスポンスを生成して Controller に返却します。

また、WebSocket を利用したリアルタイム通信を Phoenix Channels という機能で簡単に実装できます。そのほかに LiveView という機能を利用することで、JavaScript を一切書かずにリッチなフロントエンドを実現できるなど、さまざまな機能が用意されています。

Phoenix による Web アプリケーション開発

Phoenix を使うことで、高性能な Web アプリケーションを手早く開発できます。また、Phoenix Channels により、WebSocket を利用したリアルタイム通信を驚くほど簡単に実装できます。

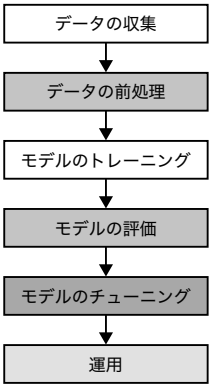
機械学習

第4次AIブームといわれる今日ですが、その基礎技術として機械学習は重要な位置を占めます。機械学習の処理を実装する際によく利用されるプログラミング言語は Python ですが、Elixir でも機械学習の処理を実装できます。

機械学習アプリケーションの構造

機械学習アプリケーションの要素は**図 1.3** のようになります。

● 図1.3 機械学習アプリケーションの要素



「データの収集」でデータ収集システムを構築し、「データの前処理」で収集データを機械学習のモデルに適した形式に変換します。「モデルのトレーニング」では前処理済みのデータを使った機械学習のモデル作成とトレーニング、「モデルの評価」で作成したモデルの性能評価、「モデルのチューニング」でテスト結果をもとにモデルを改善します。最後の「適用」でモデルを実際に利用する運用を行います。

これらの要素の共通点は、データの処理が中心になることです。つまり、Elixirで機械学習を実装する場合、データの処理を効率的に行うことができるライブラリが必要になります。

Elixir による機械学習

Nxは、Elixirの作者でもあるJoséによって開発されている機械学習ライブラリです。多次元の配列を効率的に扱うことが可能で、CPU/GPU/TPUのバックエンドで計算を実行できます。Elixirで機械学習を行う場合、Nxは必須のライブラリと言えるでしょう。

また、LivebookというJupyter Notebookのようなノートブック環境でElixirを実行できます。LivebookはElixirのコードを実行しながらその結果を表示でき、機械学習の処理を実装する際に非常に便利です。Elixirで機械学習を実装する場合、Livebookも必須のツールと言えるでしょう。

Nx のエコシステム

Nxの周辺には、機械学習の処理を効率的に実装するためのライブラリが多数存在します。Nxのエコシステムを利用することで、Elixirで機械学習を実装する際の生産性を高めることができます。

GitHubのNxのページには、Nxの関連プロジェクトの一覧[注8]が掲載されています。これらの中でも特に注目すべきプロジェクトとしてAxon[注9]があります。AxonはNxを利用したニューラルネットワークライブラリで、数値関数の関数APIや、高レベルなモデル作成API、トレーニングAPIを提供しています。

NxとAxonについては第12〜15章で詳しく解説します。

注8 https://github.com/elixir-nx

注9 https://github.com/elixir-nx/axon

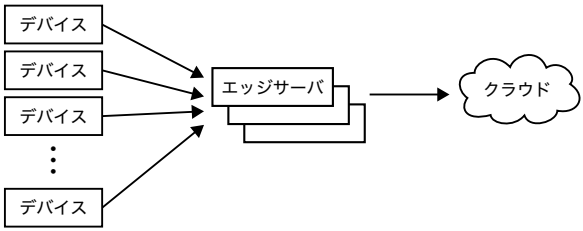
IoT デバイス

IoT は Internet of Things の略で、モノがインターネットを経由して通信することを指します。IoT デバイスとは、モノがインターネットを経由して通信するためのデバイスです。Elixir は、IoT デバイスの開発にも利用できます。

IoT アプリケーションの構造

IoT アプリケーションの要素は**図 1.4** のようになります。

● 図1.4 IoTアプリケーションの要素



デバイスはセンサからデータを収集し、エッジサーバに送信します。エッジサーバは、デバイスから受信したデータをクラウドに送信します。クラウド上で、エッジサーバから受信したデータの処理を行います。

IoT と Elixir の親和性

先ほど述べたように、IoT アプリケーションはデバイス、エッジサーバ、クラウドの 3 つの要素から構成されます。デバイスは後述する Nerves を、エッジサーバは Phoenix を、クラウドのデータ処理は Nx を利用することで、IoT アプリケーションのほとんどを Elixir で実装できます。

また、IoT アプリケーションは大量のデバイスから受信したデータを同時かつリアルタイムに処理し、エラーが発生してもシステム全体が停止することなくエラーを回復できる必要があります。これらの要件は Elixir の強みである並行性と耐障害性によって実現できます。

このように IoT と Elixir は親和性が高く、Elixir を使うことで IoT アプリケーションを効率良く実装できます。

Nervesのエコシステム

ElixirのWebアプリケーションフレームワークとしてPhoenixがあるように、ElixirのIoTデバイス向けプラットフォームとしてNervesがあります。Nervesの構成要素は以下です。

- Erlang VMが起動できる最小構成のLinuxブートローダ
- Elixirの実行環境
- ビルド管理やファームウェア更新を行うためのコマンドラインツール

Nervesを使って実装したIoTデバイス上で動作するアプリケーションは、Elixirのプロジェクトとして管理されるので、Nerves周辺の機能はElixirのHexパッケージとして組み込むことができます。

たとえば、IoTデバイスでLEDを点灯させたい場合を考えてみます。LEDの操作APIを提供するnerves_leds[注10]パッケージをNervesのプロジェクトに組み込むことで、IoTデバイス上でLEDを点灯させることができるのです。

Nervesについては第16章〜第19章で詳しく解説します。

まとめ

本章ではプログラミング言語Elixirの成り立ちと言語的な特徴について解説しました。また、Webアプリケーション、機械学習、IoT開発におけるElixirの利用状況について紹介しました。

さまざまな分野でElixirが利用されるElixirの汎用性の高さを感じていただけたでしょうか。

それでは、次章からさっそくElixirのプログラミングについて解説していきましょう。

注10　https://github.com/nerves-project/nerves_leds