目次 Contents

この本について

まずは、この本を手にとっていただき、ありがとうございます。この本はPHPをベースに、同期処理や非同期処理の流れについて、今まであまり触れる機会がなかった方向けに執筆いたしました。

世に出回っている文献や書籍には、並列処理のことを非同期処理と言っているものがあります。細かいところだとマルチプロセスをマルチスレッドと同一であると誤解したり、並列処理が非同期処理と同一であると誤認したりしているような文献が多くあります。

また、並行処理と並列処理の区別をスクリプト言語ベースで解説しているものがなかなか見つかりません。Go言語やTypeScriptなどのプログラミング言語は、書きぶりはスクリプト言語そのものであるのに非同期処理が難なく扱えます。一方、本書で取り上げるPHPはスクリプト言語の中でも、そもそも非同期処理が苦手だとされてきました。というよりも、ユースケースとして必要な場面がなかったといっても過言ではありません。

この本は、初学者に向けて、並行処理と並列処理、マルチプロセスとマルチスレッド、同期処理と非同期処理を解説し、PHPによるマルチプロセス／マルチスレッドの実装、同期処理、非同期処理の実装、また、それらの知識を身につける上で欠かせない周辺知識を取り上げます。そして非同期処理を容易に扱うことができるPHPのSwooleというライブラリを利用して、プロダクション環境で非同期処理サーバー／アプリケーションを実装する方法を実践的に解説します。

本書は、お手に取っていただいた読者のみなさんが、読み終えてPHPを用いてマルチプロセス、マルチスレッド、同期処理から非同期処理まで幅広く理解し使えるようになることを目指しています。

かつて、レンタルサーバーなど共有のサーバーでPHPを使いつつ高負荷に強いパフォーマンスを求める場合はPHP以外の言語を並行して使っていました。PHPに思い入れのある人は、PHPのexec関数、system、proc_openなどのマルチプロセスを用いて非同期処理を実現していました。また、拡張機能を入れられる環境ではpthreadsといったライブラリなどを利用していました。しかし、どれもプロダクションで扱うには、メンテナンスコストがかかったり、リーダブルなコードではなくなったり、セグメンテーション違反が突如起きたりなど、導入することによるデメリットが多くありました。実際にこれらをプロダクションに導入している実績がある会社は（筆者の肌感ではありますが）そう多くないかと思います。

ですが最近の数年の間に、SwooleというPHPの拡張機能が世の中に広まりました。Swooleが画期的だったのは、非同期処理を扱う際に、「execやsystem関数でプロセスを生成しプロセスを監視する」ような仕組みが不要で、proc_openを用いた際の複雑なコードを書く必要もなく、goという関数を呼び出すだけで非同期処理が実装できるというところでした。また、本来実装の際に苦労するような

ACIDを担保した実装が可能であるところや、HTTPリクエストやWebSocketを受け付ける仕組みがビルトインで用意されているところ、使用するにあたって非常に少ない学習量で済むというところも、Swooleを扱う上での大きなメリットです。

しかし、このように優れた拡張機能を使うにしても、文献や書籍がなかなか見つからず、また見つかっても解説が英語か中国語だけであるといった事情があります。英語が苦手な人やどちらの言語もわからない人は、そもそもハードルが高いと感じてしまうのではないでしょうか。

本書は、初学者の方にとってSwooleの学習ハードルが高いという印象を払拭し、PHPでも非同期処理が手軽にできること、初学者でも簡単に取り組めるということを読者のみなさんに伝えたい思いがあり、執筆いたしました。

PHPとSwooleを取り巻く状況について

この「はじめに」の章を執筆したタイミングはPHP 8.2がリリースされた頃です。しかし、本書の内容自体は、PHP 7.x系が主流だったタイミングに書かれており、執筆するタイミングが異なっています。

また、本書執筆中にSwooleとOpenSwooleのコントリビューターが分かれ両者それぞれのリポジトリにコミットが始まっている状態になり、バージョンや扱える機能も大きく差異が出てきています。この分かれたタイミングというのが、Swooleのバージョンが4.7.1のときであり、このときはSwooleもOpenSwooleも機能に大きな差分はありませんでした。さらに、Swoole 4.7.1はPHP 8.x系に対応しておらず、PHP 7.x系だけでの動作を保証しています。

本書では、SwooleとOpenSwooleのどちらも読者のみなさんの判断で扱える形にするために、PHP 7.4系かつSwooleのバージョンを4.7.1として解説しています。もし、あなたの環境がPHP 8.x系で、Swooleを使いたいと思うのであれば、Swooleを使用するか、OpenSwooleを使用するかの選択をしなければなりません。本書では、両者のどちらかを読者のあなたが選べるようにするために汎用的な解説に留めています。

PHP 8系を使う前提であれば、本書に出てくるコマンドや引数の一部をPHP 8.xに読み替えて読み進めることで、導入することももちろん可能です。たとえば本書で触れている`./configure --enable-maintainer-zts`というコマンドと引数は、PHP 8.xであれば`./configure --enable-zts`と読み替える必要がありますし、PHPをインストールする際の（PHP 7.4としてインストールしている）`yum install -y --enablerepo=remi-php74 php php-devel php-pear php-openssl`というコマンドを、PHP 8.x系では`yum install -y --enablerepo=remi-php82 php php-devel php-pear php-openssl`と読み替える必要があります。

3.1 マルチタスク、マルチプロセスとは

マルチタスクとは、複数のプロセスを切り替えて実行できるようにする処理の総称です。Unix系OSではマルチプロセスと呼ぶことが一般的です。

さて、そもそもプロセスとは何でしょうか。プロセスとはオペレーティングシステムによって一定のメモリ空間が提供され実行してるプログラムのことです。プロセスは、実行中のプログラムが、与えられたメモリ空間に対してデータを保存したり、保存したプログラムの実行単位、つまりスレッドで処理させたりできるようになるのです。メモリ空間はいわば、位置情報（一般的にアドレスと言われています）と紐付けたいデータをマッピングし、保持し、読み出し／書き出しができるようにするための領域のことです。

マルチプロセスはシングルプロセスに比べ、デッドプロセス（ゾンビプロセス）を作らないように配慮したり、プロセス間においてお互いの同期を取らなければならなかったりする必要があり、開発にかかるコストが高くなります。その一方で、たとえ処理可能なプロセッサが1つだけであっても大幅なパフォーマンスの向上が期待できるのです。

3.2 マルチスレッドとは

近年ではシングルタスクのコンピュータは減りつつあり、私たちが使っているコンピュータは大半がマルチプロセスで動作可能です。

先ほども述べたとおり、プロセスはオペレーティングシステムからメモリ空間を割り当てられます。複数のプロセスがある場合、それぞれのプロセスにはメモリ空間が割り当てられており、個々が独立して処理を行っていきます。そのプロセスの中の実行単位の1つがスレッドと呼ばれるものになります。

スレッドは属するプロセスのメモリを共有のリソースとしてアクセスできます。原則的にスレッドそのものはメモリ空間を持っていません。したがって、一般的にマルチプロセスよりもプロセスを生成する時間的コストが低くなります。また、プロセスはカーネルで処理の切り替えが行われるのに対して、スレッドはカーネルではなくユーザーランドで処理の切り替えが行われるので、マルチプロセスと比較して処理の切り替え、つまりコンテキストスイッチが高速であるのが主な特徴です。ただし、マルチスレッドである場合、メモリ空間が共有リソースとなるので、別途メモリに対する安全性を保証する必要があります。

「メモリに対する安全性」とは何でしょうか。たとえばWebサーバで複数のリクエスト（つまり接続）が行われた場合、Webサーバが持っているメモリのデータは独立していなければなりません。なぜな

ら、WebサーバがPOST[注1]で受け取ったデータが共有メモリに置かれ、メモリのデータが独立していないと仮定したとき、メモリ空間上に存在するPOSTのデータを誤って別のリクエストが参照したり、書き換えてしまったりする事故が起こり得ます。プログラムのバグがあった場合、悪意のある方法でメモリ空間のデータにアクセスされてしまう場合もあります。これが数個のリクエストを受け付ける程度のWebサーバであれば大きな問題になりませんが、数千、数万アクセスを受け付けるようなWebサーバであれば、1つのプロセスが専有するメモリ空間が肥大化し、ほかのプロセスの処理の妨げになることもあるでしょう。

したがって、マルチスレッドが必ずしも有効であるとは限りません。共有のメモリ空間でお互いに干渉させないためにも、場合によってはマルチプロセスのほうが合理的なケースもあります。

3.3 PHPと並列処理の歴史

かつてPHPは並列処理が苦手とされていました。ただし実装できなかったわけではなく、Webアプリケーションに特化した言語という位置付けであり、そもそも並列処理におけるドキュメントが一般的な言語と比べると少なかったのです[注2]。さらにそれ以前のごく初期はテンプレートエンジンであるという位置付けでもありました。しかし並列処理を行おうとする試みはいくつかあり、筆者もその試みを行う1人でした。ここからはそうした経験に基づき、マルチプロセスやマルチスレッドをPHPで実装する上でかつてデファクトスタンダードであった手法を紹介します。

3.4 PHPでマルチプロセスを実装する

popen/proc_openを利用する

PHPにはプロセスを起動するためのpopenやproc_openといった関数が標準で備わっています。

popenは、公式ドキュメントでは「*popen — プロセスへのファイルポインタをオープンする*」と記載されています[注3]。またproc_openは公式ドキュメントでは「*proc_open — コマンドを実行し、入出力*

注1 POST は Web サーバーへ値を送るための役割を持つ、HTTP リクエストの 1 つです。データを更新したい場合、更新するデータを JSON で送ったり、Key-Value 形式で送ったりといったユースケースがあります。なお、GET という HTTP リクエストもありますが、これは参照が主な役割であり、データの受信だけを目的としています。

注2 最近だと、Go 言語であれば言語仕様でチャネルを備えていたり、Rust や Java では公式のマニュアルにスレッドの取り扱い方が書かれています。C 言語も同様にマルチスレッドについてのドキュメントがいくつかあります。

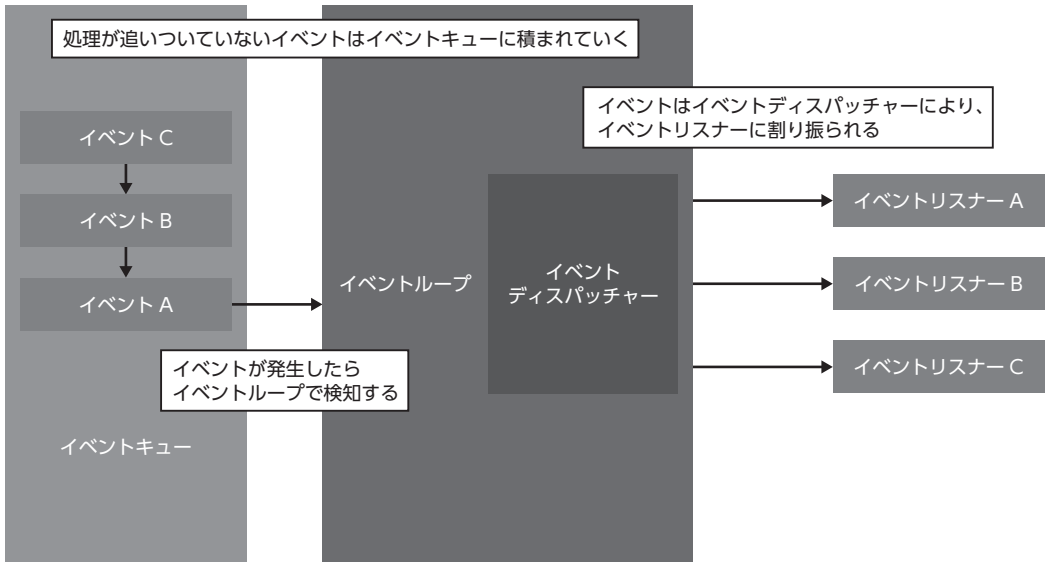
注3 https://www.php.net/manual/ja/function.popen.php

4.1 イベント駆動型プログラミングとは

さて、前章で非同期処理の実装手法としてマルチプロセスとマルチスレッドについて解説しましたが、ほかにも「イベント駆動型プログラミング」で実装する方法があります。前章で紹介したparallel\Eventsもイベント駆動型ではありますが読者の中には、イベント駆動型プログラミングという言葉にそもそも馴染みのない人もいるかもしれません。イベント駆動とは、何らかの操作や処理をプログラムの実行単位（イベント）とするプログラムを指します。

イベント駆動の仕組みを説明します（**図4-1**）。イベントが実行された際、イベントを送信する側の処理であるイベントディスパッチャーという機能によって、イベントリスナー[注1]というイベントを受け取る側の処理に送出され、そしてイベントリスナーに指定されているコールバック関数などの処理が実行されるというものです。一般的に知られている例としては、Node.jsがイベント駆動と言えます。

▼**図4-1**　非同期処理の仕組み



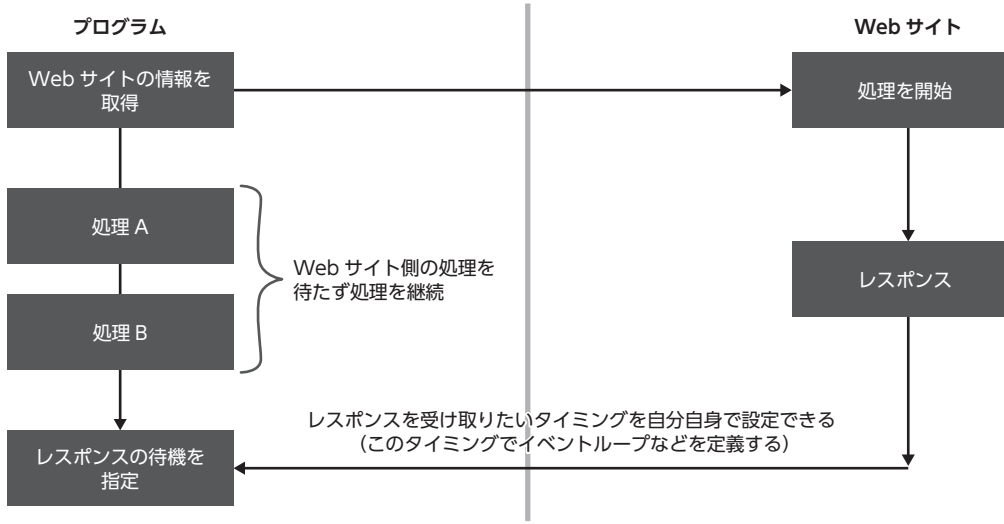
イベント駆動の重要な概念としては、「ノンブロッキングI/O」や「イベントループ」が挙げられます。

ノンブロッキングI/Oとはネットワーク接続やファイルの読み書きの際にすぐに応答がなくても処理元のプログラム側が処理を継続できるようなI/Oを指します（**図4-2**）。逆に処理を継続させず処理が完了するまで待機するI/Oを**ブロッキングI/O**と言います（**図4-3**）。イベント駆動プログラミングでも、さらに非同期処理でもノンブロッキングI/Oはよく耳にする概念です。たとえば、ブロッキン

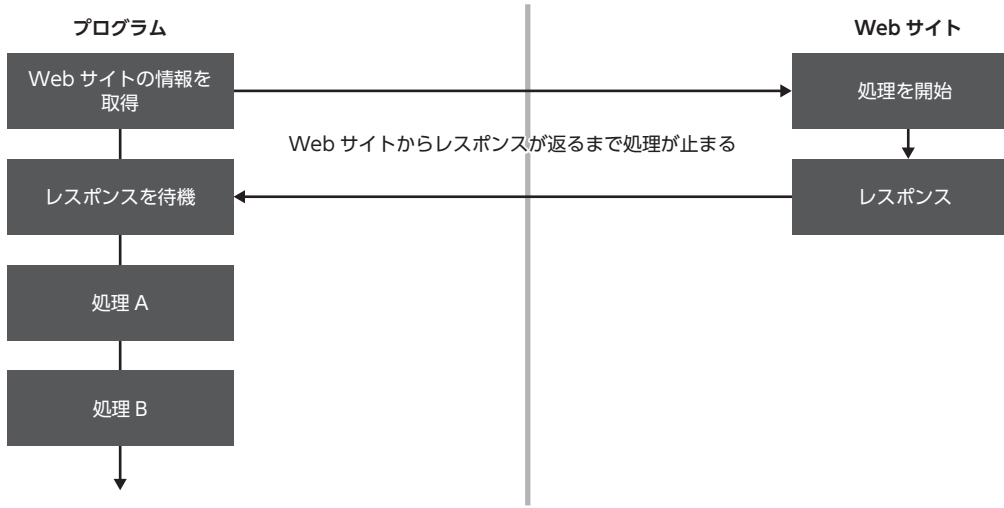
注1　イベントハンドラーと呼ばれることもあります。

グI/Oで、処理に時間がかかるI/O処理の完了まで待機させてしまうと、ほかのタスクが滞ってしまいます。しかし、ノンブロッキングI/Oで処理することで、そうしたタスクの実行が滞ってしまう課題を解決できます。ユースケースとしては、外部のWebサイトのデータを（API経由などで）非同期に取得する、データベースの値を非同期で取得する／書き込む、などがあります。

▼図4-2 ノンブロッキングI/Oのイメージ



▼図4-3 ブロッキングI/Oのイメージ



6.1 サーバーと RFC

HTTPサーバーやWebSocketサーバーを実装したことはあるでしょうか。「なんだか難しそう」「作る機会がなかった」「PHPはそういうものを作るための言語じゃない」などさまざまな理由で作ったことのない人もいるでしょう。サーバーの実装は難しそうに見えますが、第5章でも少し触れたとおり、自分で楽しむために作るものであれば、実はさほど難しくありません。とくにHTTPサーバーはRFC[注1]（Request for Comments：主に技術仕様を標準化している文書）に則って構築するだけで実装できてしまいます。WebSocketサーバーももちろんRFCが公開されています。

本章では、RFCに記述されている仕様を完全に再現するのではなく、いくつか要となる箇所だけを抜粋してHTTPサーバーとWebSocketサーバーを実装してみます。興味のある人は、RFCの実装やHTTPサーバーの仕様を詳細に言及している書籍[注2]を参考に、より実用的なHTTPサーバーを構築してみてください。

なお、たとえばサーバー上のファイルに不正アクセスするディレクトリトラバーサル攻撃や期待しない値を送られてきたケース、DDoS攻撃といったセキュリティ上のリスクを回避する方法はページ数の都合上、省略します。本章で例示しているプログラムはプロダクション環境で使うものではなくSwooleやHTTPの理解を深めるための使用にとどめておいてください。プロダクション環境で使用したい場合は、セキュリティリスクを考慮したプログラムを書くためにさまざまな対策をする必要があります。

6.2 HTTPサーバーをコルーチンで実装する

HTTPの概要

HTTPサーバーを構築する前に、HTTPの仕組みを重要な部分にしぼって紹介します。全体的な仕組みは先ほど紹介したRFCにより詳細に記載されているので、そちらを確認してください。

注 1　RFC は英語で公開されていますが、有志によって日本語版も公開されています。
HTTP/1.1 関連の RFC：
- RFC 7230　https://triple-underscore.github.io/RFC7230-ja.html
- RFC 7231　https://triple-underscore.github.io/RFC7231-ja.html

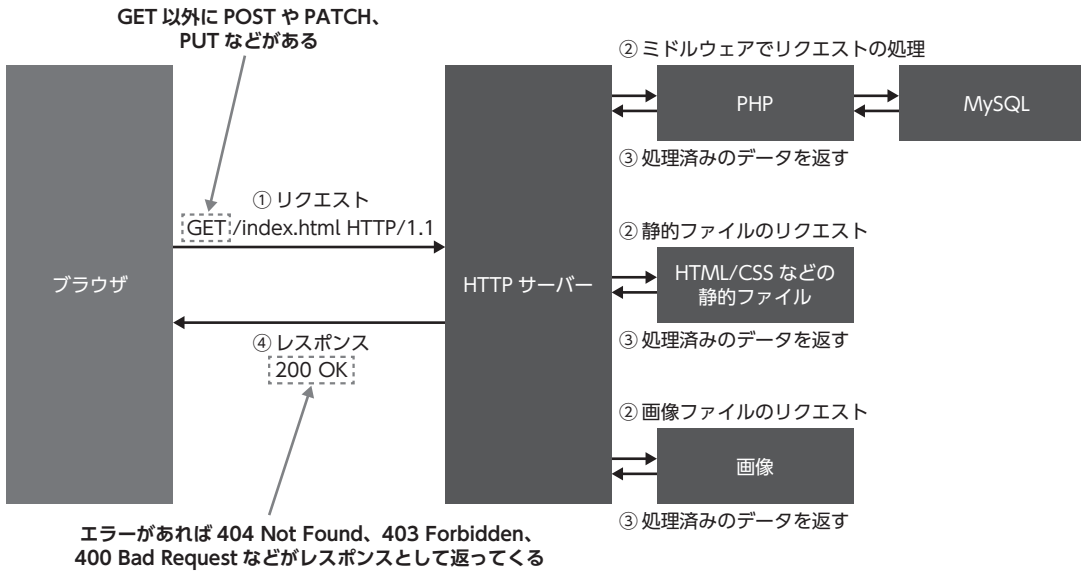
WebSocket 関連の RFC：
- RFC 5455：https://triple-underscore.github.io/RFC6455-ja.html

注 2　筆者のおすすめは次の書籍です。
渋川 よしき 著『Real World HTTP』第 2 版、オライリー・ジャパン、2020 年

HTTPはHyper Text Transfer Protocol（ハイパーテキストトランスファープロトコル）、すなわちバイナリを含むテキストデータを送受信するための規格の1つです。HTTPは、一般的なWebサイトのHTMLをレスポンスとして返す、APIの結果としてJSONを返す、画像をアップロードする、といったさまざまな用途で使われます。代表的なHTTPサーバー（ソフトウェア）としてはApache HTTP Serverやnginxなどが挙げられます。

HTTPは非常に単純な仕組みで「リクエスト」と「レスポンス」のやりとりをします（**図6-1**）。ここでは、PHPとMySQLを含めた通信をもとに図示しています。

▼図6-1 HTTPの通信の仕組み



HTTP通信は一般的にURL（Uniform Resource Locator）が利用されます。URLはたとえばhttp://example.com/path/to[注3]と表現されます。このうち、httpという部分が規格にあたります。なお、http以外にもtcpやunix、https、ftpなどの規格があります。

HTTPサーバーはブラウザやクライアント（curlなど）からのリクエストを受け取って適切なアプリケーションへ処理をルーティングし、そこからレスポンスを受け取ってブラウザやクライアントにデータを返すのが一般的です。取得したいデータはリクエストURI（先ほどの例では/path/toの部分）で処理を分けていきます。

注3 アンダーラインは任意の値を取る箇所に引いています。/path/to はファイルまでの任意のパスを、example.com は任意のドメイン（gihyo.jp など）を指します。