



	この本について .....	iii
	PHPとSwooleを取り巻く状況について .....	iv
	本書の実行環境について .....	v
<b>第 1 章</b>	<b>並行処理／並列処理の概念</b>	<b>1</b>
	1.1 並行処理とは .....	2
	コンピュータとタスク調整の関係 .....	2
	並行処理の流れ .....	2
	1.2 並列処理とは .....	4
	並列処理の流れ .....	4
	1.3 並行処理／並列処理の注意点 .....	6
	メモリへの同時書き込み .....	6
	メモリへの同時書き込みの対策 .....	8
<b>第 2 章</b>	<b>同期処理／非同期処理の概念</b>	<b>9</b>
	2.1 同期処理とは .....	10
	同期処理の流れ .....	10
	2.2 排他制御とは .....	11
	アクセスカウンターを単純に実装しようとする と不整合が発生してしまう .....	11
	楽観ロック .....	13
	悲観ロック .....	14
	flockを使わずに悲観ロックを実装する .....	15
	flockを使って悲観ロックを実装する .....	16
	2.3 非同期処理とは .....	18
	非同期処理の流れ .....	18
	2.4 第2章のまとめ：PHPで非同期処理を 実装するために .....	19

第 3 章	PHPでマルチプロセス/ マルチスレッドを実装する	21
3.1	マルチタスク、マルチプロセスとは	22
3.2	マルチスレッドとは	22
3.3	PHPと並列処理の歴史	23
3.4	PHPでマルチプロセスを実装する	23
	popen/proc_openを利用する	23
	pcntlを利用する	27
3.5	PHPでマルチスレッドを実装する	32
	pthreads—かつてのデファクトスタンダード	33
	parallelを利用する	33
	parallelの実行環境を構築する	33
	Hello Worldを試してみる	34
	マルチスレッドになっているかsleepで確かめる	35
	Functional APIを利用する	36
	引数を渡す	37
	runメソッドの第2引数に渡す	37
	useを使う	38
	タスクの実行結果を取得する	38
	排他制御を行う	39
	チャンネルで値を送受信する	39
	parallel\Syncで値を共有する	42
	parallelでresource型を扱う	44
	Eventsでタスクやチャンネルをウォッチする	47
	parallel\Events\Eventの定義と各要素	51
	Eventのユースケース	52
	parallelの制約	55
3.6	第3章のまとめ	56
第 4 章	PHPの拡張機能「Swoole」入門	57
4.1	イベント駆動型プログラミングとは	58

4.2	Swooleとは	61
	Swooleの周辺状況	62
	Swooleのインストール	62
	peclでインストールする	62
	OpenSSLを利用する場合のインストール方法	63
4.3	Swooleの基本的な使い方	65
	コルーチンで記述する	65
	<b>Column</b> ビット演算	67
	イベントループで記述する	75
	Swoole\WebSocket\Server::onのオプション	76
	オプションの設定例	83
	カーネルパラメータのチューニング	83
	コルーチンとイベントループの使い分けについて	84
4.4	第4章のまとめ	84
第5章	<b>Swooleで非同期処理を実装する</b>	<b>85</b>
5.1	第5章の概要	86
5.2	ビルトイン（組み込み）の非同期処理を利用する	86
	Swoole\Timerでタイマー処理を実装する	86
	HTTPサーバーを実装	89
	クエリ文字列を取得する	90
	POSTの値を取得する	94
	ファイルをアップロードする	97
	WebSocketサーバーを実装する	100
	startを指定する場合	101
	handshakeを指定する場合	102
	messageを指定する場合	103
	closeを指定する場合	105
	openを指定する場合	107
	ファイルの変更を監視する	108
	複数のサーバーを実装する	110
5.3	コルーチンを利用する	116
	Swooleでコルーチンを記述する方法	116

コルーチンで外部のデータを取得する .....	117
file_get_contents で取得する .....	117
guzzle を利用する .....	119
コルーチン間でデータの受け渡しを安全に行う .....	121
Swoole\Coroutine\Channel を利用する .....	122
Swoole\Atomic を利用する .....	125
書き込みのロックを実装する .....	127
PHP のファイルでロックを実装する .....	128
Swoole\Atomic でロックを実装する .....	129
Redis で非同期処理をする .....	131
<b>Column</b> predis で非同期処理を利用する .....	134
MySQL で非同期処理をする .....	136
5.4 第5章のまとめ .....	141

## 第6章 Swoole で HTTP サーバー / WebSocket サーバーを構築する 143

6.1 サーバーと RFC .....	144
6.2 HTTP サーバーをコルーチンで実装する .....	144
HTTP の概要 .....	144
<b>Column</b> HTTP リクエストメソッド .....	146
HTTP のリクエスト / レスポンスの詳細 .....	146
HTTP サーバーの実装イメージ .....	147
HTTP サーバーを実装する .....	148
クライアントの接続を待機する .....	149
接続してきたクライアントをコルーチンで処理する .....	150
リクエストヘッダーとリクエストボディを読み込む .....	153
クライアントに書き込む .....	158
6.3 WebSocket サーバーをコルーチンで実装する .....	165
<b>Column</b> Ajax と Comet、ポーリング .....	166
WebSocket のハンドシェイクを実装する .....	166
WebSocket サーバーにデータを 送受信できるようにする .....	169
6.4 第6章のまとめ .....	184

第7章	Swooleを利用して リアルタイムチャットサービスを作る	185
7.1	チャットサービスの開発に挑戦	186
7.2	チャットサービスの要件	186
7.3	HTTPサーバーを実装する	188
7.4	開発における注意と準備	190
7.5	ページを実装する	191
	<b>Column</b> PSR-7とコーディングルール	192
	ルーティング処理を実装する	205
	エラーハンドリングを行う	206
	<b>Column</b> Swooleにおけるエラー	209
	インデックスページを変更する	210
	ほかのページに遷移するリンクを設置する	210
	ユーザー登録画面を実装する	212
	送信された値を保存するように設定する	214
	データベースへの接続処理をラッパーするクラスを実装する	216
	ログイン処理を実装する	221
	<b>Column</b> セッション管理のあれこれ	223
	ユーザーの詳細表示画面を実装する	229
	チャットルームを実装する	232
	一覧を表示する	232
	ルーム作成用画面を追加する	235
	チャットルームの詳細画面と削除ボタンを実装する	239
	チャットルームの見た目をカスタマイズする	239
	ルームの削除ボタンを作成する	242
7.6	WebSocketサーバーを実装する	246
7.7	第7章のまとめ：Swooleのさらなる活用に向けて	251
	参考文献	252
	索引	253

## この本について

まずは、この本を手にとっただき、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、本来実装の際に苦勞するような

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ことも、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、また見つけ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のでしょうか。

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、初学者でも簡単に組み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ります。

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を使用するかを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<sup>注1</sup>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<sup>注2</sup>。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<sup>注3</sup>。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ン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。

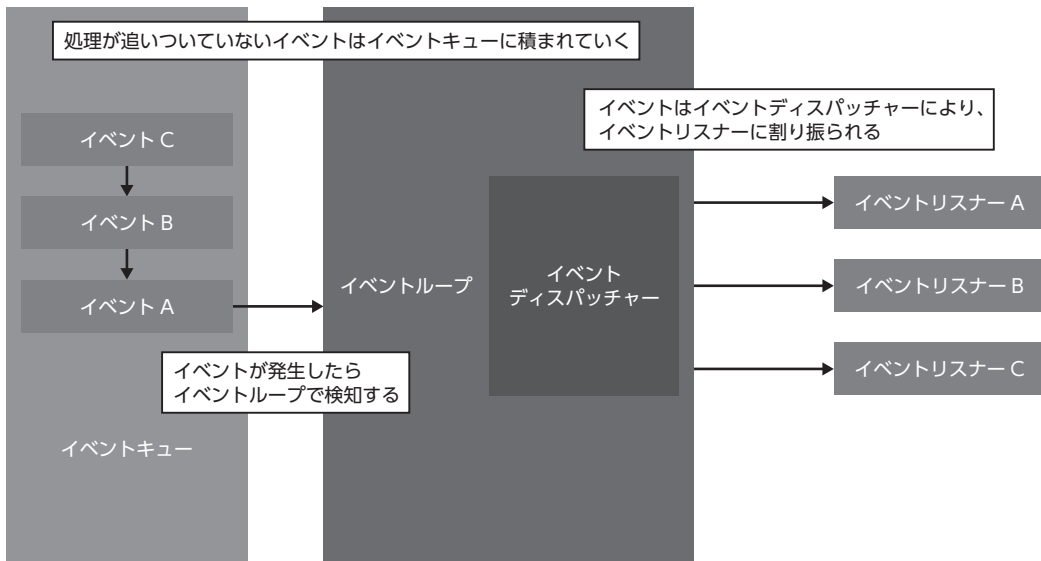
注3 <https://www.php.net/manual/ja/function.popen.php>

## 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<sup>注1</sup>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



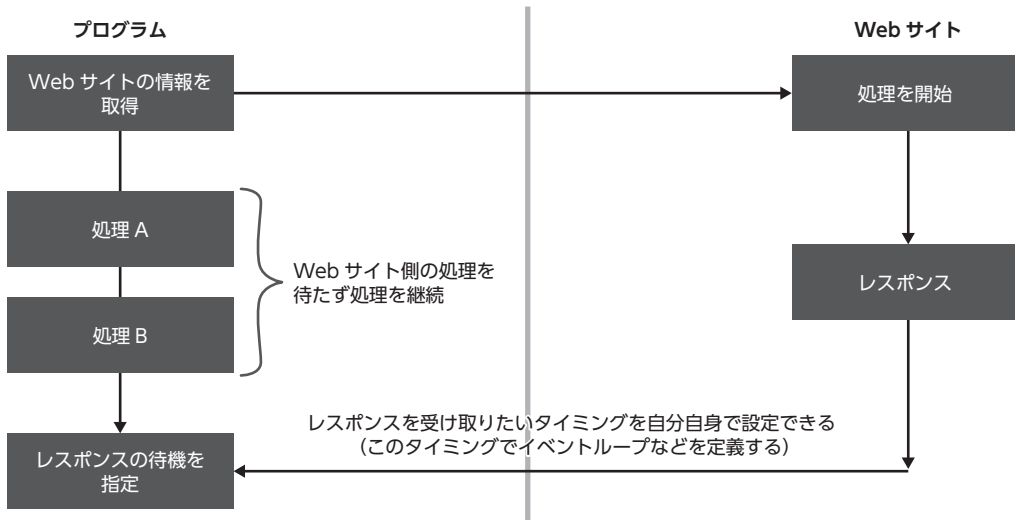
イベント駆動の重要な概念としては、「ノンブロッキングI/O」や「イベントループ」が挙げられます。

**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<sup>注1</sup> (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<sup>注2</sup>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いたい場合は、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>

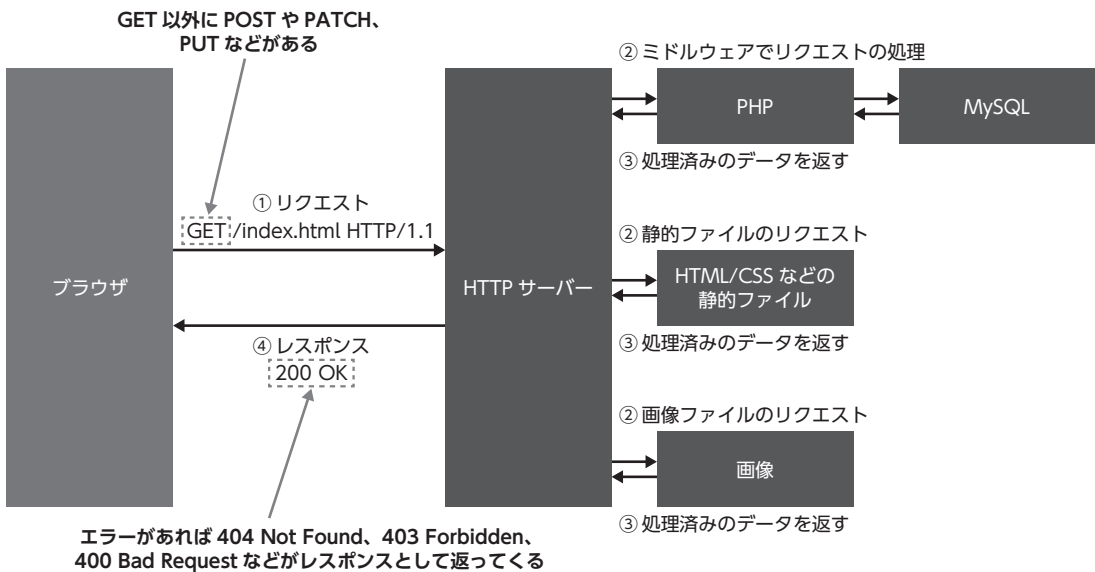
**注2** 筆者のおすすめは次の書籍です。

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に使われます。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。