特典A
アーキテクチャの選定

書籍本編ではReact（Next.js）とDjango（Python）とMySQLのアーキテクチャの組み合わせでフルスタックWeb開発の開発方法を説明しました。本特典では、そうした「アーキテクチャの組み合わせ」を決定するまでの、アーキテクチャの選定手法・プロセスを解説します。

A-1 アーキテクチャの選定方法と視点

A-1-1 アプリケーションの性質とは?

本節では、多数あるアーキテクチャから、そもそも「どのように適切なアーキテクチャを選定するか」を説明します。同時に本書で選定しているアーキテクチャの特徴、さらには他のアーキテクチャと比較した場合のメリット、デメリットなどを説明します。

本書では以下のアーキテクチャを選定しています。

- フロントエンド領域 …… React18、Next.js 13
- サーバーサイド領域 …… Django4、Python3.11
- モデル層 …… MySQL8

アーキテクチャの選定にはいくつかの「ポイント」があります。実際のプロジェクトはビジネスや、アプリケーションの性質によって異なります。皆さんのプロジェクトの要件に合わせて、選定ポイントをカスタマイズしてください。

どのようなアーキテクチャを選んでも、結局は「使い方次第」です。アーキテクチャとは「施工方法」に近いものです。鉄筋でつくるか、木造か。2X4なのか、ログハウスなのか……など、選ぶのは「どんなものを作り、何に使いたいのか」という機能要件と、耐震性や法規制などの非機能要件で決まっていきます。

それでは、それぞれの選定ポイントを見ていきましょう。

①機能要件適合性

アーキテクチャ選定の際、最も重要なのは機能要件です。例えば、作りたいソフトウェアがWebなのか、デバイスドライバなのかでは、選定されるアーキテクチャがまったく異なります。同じWebでも金融機関の決済サービスなのか、単なる知人との情報共有に使う掲示板なのかで、選定の基準は変わります。また、この機能要件の適合性は、ものづくりの際の「開発コスト」にも連動していきます。機能要件が適していると、実現しようとしている要望に見合うAPIなどが充実していれば開発する要件（工数）は少なく済みます。

例えば、Pythonは統計や数学的解析などの機能が充実していることで知られ、AIやビッグデータ解析に使われます。一方、型に曖昧さがあることから、金融機関の決済など「絶対に落ちてはいけないサービス」には向かないと評する声もあります。

Web画面のレンダリングや、HTTP通信の処理、データベースの接続など、基本的な機能の実現については、主要な言語やフレームワークはカバーしています。そのため、アーキテクチャ比較の観点では、次に挙げるような「特徴的な要件」に視点を当てていきましょう。

- グラフや地図などの描画機能
- 帳票などPDFの生成機能
- スマートフォンのSPA（Single Page aplication）などの機能
- 利用が決まっているツールやミドルウェアのサポート機能

②非機能要件適合性

アーキテクチャ選定では、非機能要件の観点も重要です。

ミッションクリティカル性

ミッションクリティカル性とは「社会インフラの制御」や「リアルタイムの金融決済」など、正確性や安定性が必要なシステムに求められる要素のことです。本書では以下のポイントに注目しています。

- **セキュリティ堅牢性**

典型的なセキュリティリスクに対する対策などが行われているか、という観点です。例えば、SQLインジェクションといった攻撃に対して、SQLで使用される予約語や特殊記号を予めブロックする仕組みが用意されているかどうかを確認します。また、テキスト入力フィールドにスクリプトを挿入し、それが実行される（クロスサイト・スクリプティング）ような事態を未然に防ぐための入力チェック機能がフレームワークに含まれているかも検討のポイントです。

不正なフォーム送信を防ぐためにトークンを発行して画面の送受信を制御する機能など、多くのフレームワークはこれらのセキュリティ機能を事前に提供しています。

- **型安全性**

PHPなどのスクリプトタイプのインタプリタな言語では、型の厳密さが不確かなことがあります。型の厳密さが不確かであることには、長所と短所があります。型が厳密でない場合、簡単な会員サイトなどを構築する際には、考慮すべきことや手間が少なくて済みます。しかし、金銭の取引などのミッションクリティカルなソフトウェアにおいては、「項目に2バイト文字で金額が入っていたため、正確な計算ができなかった」といった問題を避けなければなりません。このような「型の厳密さ」は、システムが求める「正確さ」に強く依存します。

- **可用性**

言語やプラットフォームによって、プログラムの動作する仕組みが違います。例えば、PHPの場合、アクセス時にプロセスを生成し、スクリプトをそのプロセス内のスクリプトエンジンで解析し、随時コンパイルして実行します。

この方式は、Webなどの大量アクセスがある場合、無限にプロセスを生成し、最終的にはパフォーマンスの低下からシステムの機能停止につながる可能性があります。また、リソースの廃棄もアクセスの管理に依存しており、適切に処理されない場合、リソースの無駄遣いやシステムのダウンなどが発生するかもしれません。

一方で、JVM（Java Virtual Machine）はプロセス管理やオブジェクトの廃棄を管理する仕組みであるガベージコレクションを提供しています。そのため、リソースの管理やプロセスの廃棄について、スクリプト言語よりも適切に処理されるといえます。

さらに、C++などのように、プロセスやメモリなどの制御をプログラムから直接管理できる言語の場合、直接的にプロセスやスレッドの管理が可能です。これは開発の工数を増やすかもしれませんが、自身で制御できるため、高いパフォーマンスや確実な動作を実現できます。

要求パフォーマンス

パフォーマンスも重要なアーキテクチャの決定要素です。一概にパフォーマンスといっても、数億件もの大量のデータを扱えるのか、スマートウォッチのような性能の低い環境でも十分に動作するのか、動画の配信のような巨大データを持ち回るのが得意なのか、など様々な観点があります。

パフォーマンスの要求は、アーキテクチャの特性や機能要件によって影響を受けます。例として、チケットの販売サイトでは、何万人ものユーザーからのリクエストを同時に処理する能力が優先されます。

また、一般的に汎用性の高い言語は仮想マシンなどを用いて、ハードウェアやOSに対して疎結合に設計されています。その結果、パフォーマンスの観点でC++などのようなハードウェアやOSに密結合なアーキテクチャより劣る傾向があります。

③汎用性

Kotlinのように、1つのソースコードでiOSでもAndoroidでも動作するようなアーキテクチャは汎用性が高いと言われます。仮想マシン（VM）を介するJavaも同じ発想で様々なハードウェアやOS上で同じソースで動作出来ることを設計思想としています。

VMの中でも最も主流なのはJVM（Java Virchal machine）です。JVMは主要なOS（Windows、linux/unix、iOS、andoroid、など）がサポートしています。

またアプリケーションサーバーなどのJVMでの利用を前提とした実行基盤も多く、OSやアプリケーションサーバーなどの導入によって費用をかけることなく環境が手に入ります。また、JVMはslacaやkotlinなどの他に50近くもの様々な言語の実行が可能で、汎用性が非常に高い言語といえます。

一方で、よりハードウェアに近い機械制御のためのプログラミングや、デバイスドライバやガジェットのコントロールなどにはハードウェアに合わせた言語を使用する必要があります。

④利用コスト

アーキテクチャやミドルウェア、フレームワークの中には有償なものや無償のものが数多く存在します。Webアーキテクチャにおいては、オープンソースソフトウェア（OSS）が多く提供されており、多くのサービスは無償のツールだけを使用して開発できます。ただし、帳票出力（PDFなど）などの一般的な機能でも、開発には手間がかかる場合があります。このような場合、有償のAPIなどを購入することで、開発コストを削減できることがよくあります。

一方で、高いミッションクリティカル性を持つシステムにおいては、OSSだけでは十分でないことがしばしばあります。リアルタイムでの外部からのサポートが期待できないため、有償のソフトウェアやサービスが必要とされます。

また、近年では内製化のトレンドが広がり、IT開発の専門家ではないスタッフがソフトウェア開発に参加することが増えています。有償ツールを使用する場合、リアルタイムで専門家からサポートを受けることで、エンジニアの高い専門性がチーム内に不足していても、開発を進めることができます。

プロジェクトのリスク低減や開発コスト削減の観点から、ツール、環境、プログラミング言語に一定の費用を充てることが、プロジェクトの総コストや保守コストを削減する助けになることがあります。

⑤開発・保守コスト

開発コストは、さまざまな要因から構成されます。一般的に、特定の機能に合ったプログラミング言語を選択した場合、言語やフレームワークの特性だけで開発生産性に大きな違いはないといわれています。例えば、Python、Java、PHPで同様の機能を持つWebシステムを開発する場合、各言語の固有のAPIを使用しない限り、大きな違いは生じないと一般的に考えられています。

開発言語自体の生産性の違いよりも、情報量の多さ、使用可能なツール、経験豊富なエンジニアの入手可能性、開発・運用組織のスキル、習熟度などが大きな影響を与えます。これらの要素がわずかに生産性に影響し、最終的に開発コストに大きな違いをもたらすことがあります。

押さえておきたいのは、最も大きな開発コストの要因はエンジニアの教育コストであるということです。また、保守、運用、拡張を行うエンジニアの教育コストも、広い意味での開発コストに含まれます。つまり、「将来かかる費用」もアーキテクチャの選択に大きな影響を与えます。

要するに、開発コストは単なるプログラミング言語やフレームワークの選択だけでなく、情報、ツール、エンジニアのスキル、教育コスト、そして将来の保守と運用に関連する多くの要因から成り立っています。そのため、開発プロジェクトを計画する際には、これらの要因を総合的に考慮することが重要です。

⑥流行り

「流行り」は、実は意外と重要な要素です。流行りとはさらに細分化すると以下のような要素になります。

- **開発者数**
  技術認知度は知見を持つエンジニアの数。場合によってはチームメンバーの集めやすさや単価など、コストにも影響します。

- **情報量**
  個人のブログやトラブルシュート記事など日本語での情報量です。

- **プロダクトサポート**
  機能の追加のみならず「致命的なセキュリティインシデントなどの対応が定期的になされているか」があります。有償プロダクトなら対応は、費用の中に入っていてしかるべきですが、OSSのフレームワークなどは「有志の貢献」が前提です。流行っている状態であれば、コミュニティの参加者も多く、活動が活発で迅速な対応が期待できます。

⑦ビジネス的な理由

ユーザーやクライアントから寄せられるニーズに関する要素です。具体的には「有償プロダクトの機能を使いたい」「既に導入済みの既存サービスと連携したい」といったニーズが挙げられます。社内で既に保守要員がいるなどの理由で、ユーザーから要望が寄せられ、アーキテクチャが限定される場合があります。あるいは、クライアントへ向けた事前のプレゼンテーションなどのプリセールス段階で、プラットフォームやアーキテクチャの要件が決まっていることもあります。

Purchase benefits

A-2 アーキテクチャ選定の実践

ここまで、アーキテクチャの特性の違いが、スキル特性の違いに紐づき、チーム構成や開発プロセスなど、フルスタック開発の要素全般に影響を与えることがわかりました。

この節では実際に事例を用いて「アーキテクチャ選定」を行っていきます。実プロジェクトでは最初に要件や要望を聞き「実現する方法」としてアーキテクチャを決定します。フルスタックエンジニアとしてシステム開発を行う際には、全体的なアーキテクチャ構成を「通し」で考えることになります。

なお、アーキテクチャが決まったら、アーキテクチャ構成図を最初に書くことをおすすめします。チームで開発する際に、最初に「全体像のイメージ」を掴むのに最適なためです。

それでは、前述の比較ポイントを実際に並べて、アーキテクチャを検討してみましょう。ちなみに、各観点の評価は執筆時点（2023年11月）の情報を元にした、筆者の主観によるものです。

A-2-1 フロントエンドアーキテクチャの比較

本書ではサンプルアプリケーションを取り扱うため、機能要件の適合性については説明を割愛します（明確な要件がないためです）。

非機能要件適合性（ミッションクリティカル性）

まずは、フロントエンドにおける非機能要件を見ていきましょう。

セキュリティ堅牢性

クロスサイトスクリプティング（XSS）などWeb特有のセキュリティの問題が複数存在しています。XSSに関しては今回比較している3つのフレームワークは全てリクエスト文字列に対するエスケープが実装されており、対策されています。

型安全性

データの型などの動作が厳密に作られていることはシステムの堅牢さと比例します。その代わり、設計や製造も厳密な型を考慮して行う必要が生まれ、学習コストを押し上げることにもなります。TypeScriptで作られているAngularは堅牢といわれています。また、Next.js、Nuxt.jsはTypeScriptを選択すれば堅牢といえます。本書ではTypeScriptを選択しています。なお、下記の比較表ではNext.js、Nuxt.jsは「TypeScriptを選択していない」状態で比較しています。

可用性

本書で比較しているフロントエンドアーキテクチャは、いずれもブラウザ上で動作します。そのため、この3つのアーキテクチャの相手には稼働確実性に差はありません。

表A-2-1 3つのアーキテクチャの比較表

	React（Next.js）	Vue（Nuxt.js）	Angular
セキュリティ堅牢性	○普通	○普通	○普通
型安全性	×低い	×低い	○高い
可用性	○普通	○普通	○普通

非機能要件適合性（要求パフォーマンス）

パフォーマンスにもいくつかの観点があります。

モジュールの大きさ

ライブラリや定義体などのファイルサイズが大きい、数が多いなどの理由で初期のロード時に時間がかかることがあります。

画面の描画のはやさ

画面を描画する際に差分だけを更新するなど仕様の違い。あるいはDOMのふるまいなどによって描画する速度も変わります。

HTMLDOM とは Document Object Model の略で、HTMLのテキストデータを解釈してブラウザ上に表現する役割持ちます。

使用するメモリの大きさ

フロントエンドアーキテクチャは表示しているデバイス上で動作しています。その使用メモリは見ているパソコンのメモリを使用する形で消費します。パソコンであれば、さほど使用量を機にする必要はありませんがスマートフォン上の限られたメモリではしばしば、ユーザーにストレスを与えます。あるいは複雑で大きな画面を作ると影響が増大します。

表A-2-2　3つのアーキテクチャの比較表

	React（Next.js）	Vue（Nuxt.js）	Angular
モジュールの大きさ	○普通	○普通	×大きい
画面の描画	○普通	×遅い	○普通
使用メモリの大きさ	○普通	×大きい	○普通

評価表では、ベンチマークが3つの平均値から20%以上、大きい、遅いものを×としています。なお、執筆時点の比較結果であり、フレームワークやAPIのバージョンアップで描画の仕方やメモリの使い方などの仕様が代わり大きく結果が変わる可能性があります。

また、「作り方」によってもパフォーマンスは大きく変わります。例えば数千件のデータを全部取ってきて画面上で絞り込み表示すれば重くなりますし。利用に適した数十件だけを取りまわしていれば処理は軽くて済みます。よってフレームワークの重さやライブラリの大きさだけでパフォーマンスが決まるわけでもありません。

汎用性

フロントエンドにおける「汎用性」とは、対応できるブラウザの種類や、デプロイできるサーバー環境の比較です。執筆時点で3つのフレームワークには、いずれも大きな差はありません。

開発・保守コスト

本書では実プロジェクトでは無いため「機能要件」がなく見積もりを行うことができません。そのため開発保守コストの比較は出来ないため本書では「学習コスト」として説明していきます。

本書で比較している3つのアーキテクチャには日本語のものを含めて公式ドキュメントやテックブログ、コミュニティの活動が一定以上あり、概ね学習に必要な情報は手に入ります。

Reactは、JavaScriptの知識があれば比較的簡単に学習できます。Reactのコンポーネントベースのアーキテクチャは、複雑なUIを効率的に管理できるため、開発者にとって理解しやすいです。

Vueは、Reactよりも学習コストが低いといわれています。Vueは、Reactと同様にコンポーネントベースのアーキテクチャを採用しています。Vueは、Reactよりもやや情報量が少ないと言われています。

Angularは、ReactやVueよりも学習コストが高いといわれています。これは、フルスタックのフレームワークであるため、学習することが多く、初心者にとってはハードルが高いことがあります。TypeScriptの知識をすでに持っている場合は容易に学べます。

流行り

本書で比較している3つのアーキテクチャは、いずれも「メジャー」といえるでしょう。そのため、必要十分な情報量・開発者数・サポートがありますが、さらに深堀してみましょう。

開発者数

React、Vue、Angularの開発者数を比較する際の参考値として、「ダウンロード数」があります。これは開発環境をインストールした人の数ですので、概ね開発者の数として読み替えることができます。

情報量

React、Vue、Angularを情報量で比較する際の参考値として、「検索数」が挙げられます。これは必ずしもコンテンツの量ではありませんが、それだけ多くの人が検索していることは流通する情報の量が多いと推定されます。

プロダクトサポート

プロダクトサポートの主な観点は、継続的な機能開発や、セキュリティパッチなどが潤沢に提供されているかを示しています。なお、比較する3つのフレームワークはいずれも、脆弱性が発見された場合、セキュリティパッチがすぐに提供されています。

Reactのリリースは通常、1年に1回程度行われます。メジャーバージョンアップは2年以上の間隔で行われます。

Vueのリリースは通常、1年に2回程度行われます。メジャーバージョンアップは1年以上の間隔で行われます。

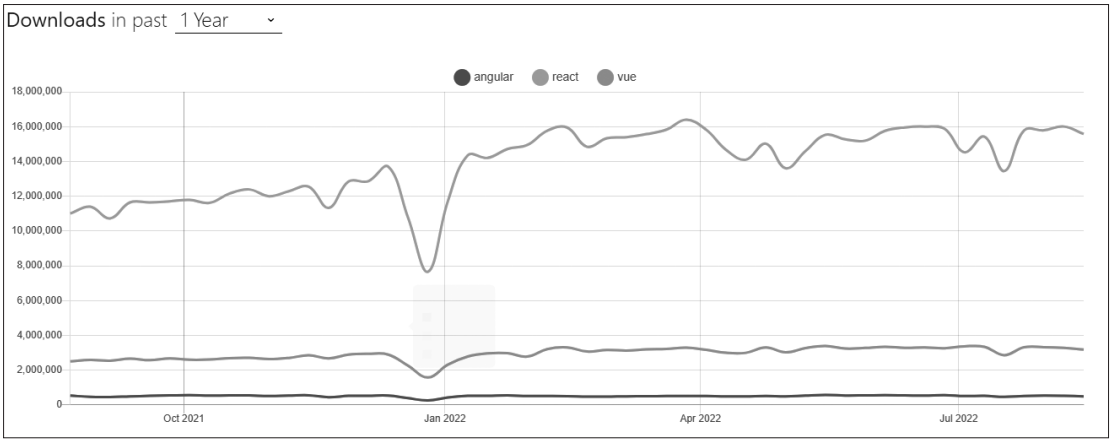
Angularのリリースは通常、1年に2回程度行われます。メジャーバージョンアップは1年以上の間隔で行われます。

総じて、これらのフレームワークはどれも頻繁にアップデートが行われ、セキュリティパッチも早急に提供されることが多いです。Reactのリリース頻度は最も低く、Vueのリリース頻度は中間、Angularのリリース頻度は最も高くなっています。

表A-2-3 3つのアーキテクチャの比較表

	React（Next.js）	Vue（Nuxt.js）	Angular
開発者数	◎1000万	○200万	△20万
情報量	◎非常に多い	○多い	△普通
プロダクトサポート	○活発	◎非常に活発	◎非常に活発

図A-2-1 検索されている回数の比較



図A-2-2　ダウンロードされている回数の比較

総合的な比較結果（フロントエンド）

フロントエンドの比較結果をまとめてみましょう。比較結果をもとに、アーキテクチャを決定していきます。

表A-2-4　3つのアーキテクチャの比較表

		React（Next.js）	Vue（Nuxt.js）	Angular
機能要件適合性		ー	ー	ー
非機能要件適合性	ミッションクリティカル性	○普通	△やや弱い	○普通
	要求パフォーマンス	○通常範囲	△他に劣る	○通常範囲
汎用性		○普通	○普通	○普通
利用コスト		◎なし	◎なし	◎なし
開発・保守コスト		○	◎	△
流行り	開発者数	◎1000万	○200万	△20万
	情報量	◎非常に多い	○多い	△普通
	プロダクトサポート	○活発	◎非常に活発	◎非常に活発

A-2-2 サーバーサイドアーキテクチャの比較

続いて、サーバーサイドアーキテクチャの比較を見てみましょう。フロントエンドと同様、機能要件は割愛いたします。

非機能要件適合性（ミッションクリティカル性）

セキュリティ堅牢性

サーバーサイドアーキテクチャはエンティティモデル（データベース）と密接な関係があります。構成によっては物理的・論理的に同じネットワーク空間やセグメントに存在し、サーバーサイドで開発したシステムのセキュリティ脅威が、そのまま保管データに影響してしまいます。その結果、セキュリティに対する脆弱性がある場合、ユーザーの個人情報や重要なデータが漏洩する可能性があります。

Python（Django）は、セキュリティに重点を置いたフレームワークであり、SQLインジェクションやクロスサイトスクリプティング（XSS）などの攻撃に対する防御機能が組み込まれています。また、DjangoはCSRFトークンをデフォルトで提供し、開発者がセキュリティ上のリスクを最小限に抑えることができます。既知の主要なセキュリティ脅威に対しての機能を備えています。

Java（Spring）は、世界中の様々なミッションクリティカルなシステム・サービスで利用されており、セキュリティに関する多くの実績・機能があります。例えば、Spring Securityは、アプリケーションの認証や認可に使用される強力なセキュリティフレームワークです。また、Javaは静的型付け言語であり、コンパイル時に型エラーを検出できるため、ランタイムでの意図せぬエラーがセキュリティ脅威に繋がる可能性を減らすことができます。

PHPは、Web開発に広く使用されているプログラミング言語であり、セキュリティ上の問題が報告されることがあります。前述のアーキテクチャに比較するとフレームワークによって提供されるセキュリティ機能が少ない傾向があります。PHPはまた、型の柔軟性が高く、コーディングミスによる脆弱性が発生する可能性があります。

型安全性

Pythonは、動的型付け言語であり、変数の型を宣言する必要がありません。このため、実行時に型の不一致が起こる可能性があります。Pythonは柔軟で表現力が高い反面、型エラーが実行時に発生することがあります。ただし、Pythonの静的型検査ツールや型アノテーションのサポートにより、型の安全性を向上させることができます。

Java（Spring）は、静的型付け言語であり、コンパイル時に型のチェックが行われます。これにより、実行時に型の不一致によるエラーを減らすことができます。また、Javaのジェネリクス機能や型注釈などの強力な型システムにより、より安全なコードを作成することができます。

PHPは、動的型付け言語であり、変数の型を宣言する必要がありません。このため、実行時に型の不一致が起こる可能性があります。PHPは柔軟な言語であり、動的なWeb開発に適していますが、型の安全性には課題があります。ただし、最近のバージョンでは、型アノテーションのサポートが向上しており、型の安全性を向上させることができます。

総合的に見て、Java（Spring）は静的型付け言語であり、コンパイル時に型のチェックが行われるため、型の安全性が高いとされています。一方、Python（Django）とPHPは、動的型付け言語であり、柔軟性と表現力が高い反面、型の安全性には課題があります。ただし、PythonとPHPの最新の

バージョンでは、静的型検査や型アノテーションのサポートが向上しており、型の安全性を向上させることができます。重要な点は、各言語やフレームワークの特性を理解し、開発プロジェクトの要件に応じて適切な言語や型のアプローチを選択することです。

可用性

サーバーサイドは、Webサービスの特性上、24時間365日の稼働を要求される場合が多いです。したがって、可用性は重要な非機能要件となります。システムに障害が発生した場合でも、レプリケーションを動的にできてかつ、障害の発生していないサーバーやインスタンス上で処理が継続できるように、冗長化やフェイルオーバーの機能が必要です。

Python（Django）は、優れた可用性を担保する設計となっています。Djangoのアーキテクチャは、垂直および水平スケーリングをサポートし、Webサイトのトラフィックに応じてリソースを割り当てることができます。また、Djangoには、フェイルオーバー機能やバックアップ機能、復元機能などが組み込まれており、システム障害が発生した場合でも、サイトの可用性を維持できます。

Java（Spring）は、高い可用性を提供するために設計されています。Spring CloudやSpring Data、Spring Bootなど様々な可用性を保証する機能が用意されています。フェイルオーバー、クラスタリング、ロードバランシングなどの機能を提供し、システム障害が発生した場合でも、サイトの可用性を維持することができます。また、Javaは、複数のサーバーでの分散処理を可能にするため、大規模なWebサイトでも高い可用性を提供することができます。

PHPは、可用性に関する問題があるとされています。PHPは、多くのWebサイトで使用されていますが、高トラフィックのWebサイトでの可用性の問題が報告されることがあります。PHPの可用性を向上させるためには、フェイルオーバーやロードバランシングなどの機能を実装する必要があります。インタプリタ言語の特徴としてあらかじめコンパイルを行わないため、ソースコードを再配置するだけでシステムを止めずにアップデートが行えます。

表A-2-5 3つのアーキテクチャの比較表

	Python（Django）	Java（Spring）	PHP
セキュリティ堅牢性	○高い	○高い	×低い
型安全性	○高い	○高い	×低い
可用性	○高い	○高い	×低い

非機能要件適合性（要求パフォーマンス）

処理速度

サーバーサイドは多数のユーザーから同時リクエストを処理する必要があります。そのため、プログラムのパフォーマンス（処理速度、レスポンス時間、スループットなど）は非常に重要です。パフォーマンスが低下すると、ユーザーの体験が悪化するため、サービス提供者の信頼性が低下することがあります。

また、同時に多数を処理するWebシステムの特性からセッションの管理やトランザクションの一貫性、オブジェクトのスコープ管理なども重要になります。

Python（Django）はインタプリタ型言語であるため、JavaやC++のようなコンパイル型言語と比較すると、処理速度が遅いとされています。また、Djangoは、フレームワークの機能が多いため、処理速度が低下する可能性があります。

Java（Spring）はコンパイル型言語であるため、PythonやPHPと比較して、処理速度が速いとされています。また、Springフレームワークは、高速でスケーラブルなWebアプリケーションを構築するために設計されており、高速な処理が可能です。

PHPはインタプリタ型言語であるため、処理速度が遅いとされています。ただし、PHPは、Web開発に適した言語であり、Webアプリケーションの開発においては、処理速度があまり問題とならない場合があります。

総合的に見て、Java（Spring）が最も処理速度が速いとされています。一方、Python（Django）とPHPは処理速度が遅いとされていますが、Web開発に適した言語であるため、処理速度があまり問題とならない場合があります。

スケーラビリティ

Webシステムは同時アクセスが多く、時間帯やサービスの状況、イベントやキャンペーンにより極端に、利用者が増減します。そのため利用者数が増えたときにシステムを拡張できるように設計される必要があります。結果スケーラビリティが重要な非機能要件となります。

Python（Django）はスケーラビリティが高いフレームワークであり、水平および垂直スケーリングをサポートしています。DjangoのMTV（Model-Template-View）アーキテクチャにより、アプリケーションの機能を分割して開発することができ、コンポーネントの単一化が容易になります。また、キャッシングやクラスタリングなどの機能を提供しており、高負荷状態でもスケーラビリティを確保することができます。

Java（Spring）は高いスケーラビリティと実績を持つプログラミング言語であり、JavaEEフレームワークは、スケーラビリティに関する多くの機能を提供しています。Java EEは、クラスタリング、ロードバランシング、フェイルオーバーなどの機能を提供し、高負荷状態でもスケーラビリティを確保することができます。

PHPは単一プロセスおよび単一スレッドで実行されるため、アクセス数が増えるほどプロセスが増え消費リソースが増大するためスケーラビリティが低いとされています。言語仕様としてPHPは、ス

ケーラビリティの機能や考慮が少なく、ツールやフレームワークにより開発サイドで強化する必要があります。

総合的に見て、Python（Django）およびJava（Spring）は、スケーラビリティが高いことが知られています。一方、PHPは、スケーラビリティに関する問題があるとされていますが、適切な設定や手動でのスケーリングによって改善することができます。重要な点は、どの言語やフレームワークを使用するにしても、スケーラビリティを確保するために、水平および垂直スケーリング、クラスタリング、ロードバランシングなどの機能を使用することが重要です。

表A-2-6　3つの言語の比較表

	Python（Django）	Java（Spring）	PHP
処理速度	×低い	○高い	×低い
スケーラビリティ	○高い	○高い	×低い

汎用性

サーバーサイドにおける「汎用性」とは、実行できるOSやVMの種類など動作環境の広さとなります。

Python（Django）はWindows、macOS、Linuxなどの主要なオペレーティングシステム上で動作させることができます。

Java（Spring）はVM上で動作します。JavaVMは主要なオペレーティングシステムだけでなく、スマートフォンのような軽量OSなどでも動作します。JavaVMでは、Scalaなど他の言語も動作するなど非常に強い汎用性を持ちます。

PHPは主にWebアプリケーション開発に使用される言語であり、Windows、macOS、Linuxなどの主要なオペレーティングシステム上で動作することができます。ただし、環境やモジュールの準備が必要な場合が多く、他の2言語よりやや汎用性は劣ります。

表A-2-7　3つの言語の比較表

	Python（Django）	Java（spring）	PHP
汎用性	○高い	◎非常に高い	△普通

開発・保守コスト

本書は、実プロジェクトではなく「機能要件」がないため、見積もりを行うことができません。開発保守コストの比較はできないため、本書では「学習コスト」として説明していきます。

Python（Django）は比較的簡単な文法と読みやすいコードが特徴であり、初心者にとって学びやすい言語の1つです。Djangoは、PythonのWebアプリケーションフレームワークであり、Pythonを学ぶことが前提条件ですが、Django自体は、初心者にとっても比較的学びやすいフレームワークです。

Java（spring）は文法がやや複雑であり、初心者にとっては学びにくい言語とされています。しかし、オブジェクト指向プログラミングの概念が強く組み込まれており、プログラミングの基礎を身につけた後に学ぶことが望ましい言語です。Springフレームワークは、非常に利用実績が多いので学習の際に書籍やサイトなどの情報が豊富です。

PHPは比較的簡単な文法と読みやすいコードが特徴であり、初心者にとって学びやすい言語の1つです。型や記法のルールが緩やかでHTMLへの知識があれば取り組みやすくWeb画面に置いて表現が容易です。

総合的に見て、Python（Django）およびPHPは、初心者にとって比較的学びやすい言語であり、Java（spring）は、学習コストが高い言語とされています。

表A-2-8 3つの言語の比較表

	Python（Django）	Java（spring）	PHP
学習コスト	○低い	△普通	◎非常に低い

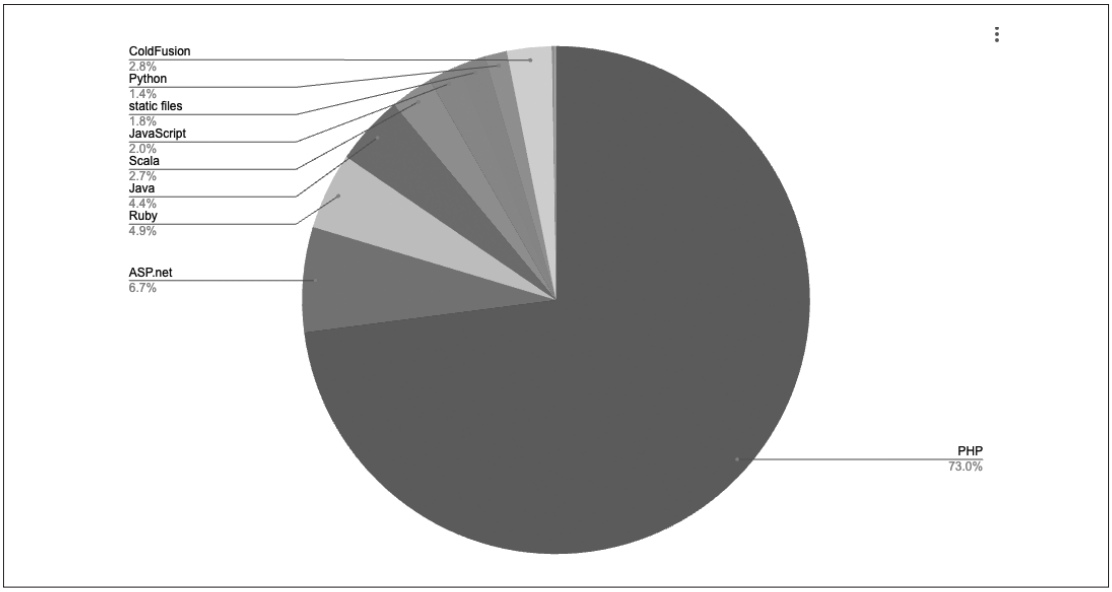
流行り

本書がサーバーサイドで比較している3つのアーキテクチャはいずれも「メジャー」と言えます。そのため、必要十分な情報量・開発者数・サポートがありますが、さらに比較を行います。

開発者数

「開発者数」を正確にはかった統計はないため、参考に「使用されているWebシステム上の開発言語のシェア」を見てみましょう。

w3techs.comでは、Webサイトの開発言語を自動的に収集し、類推しています。その統計によると2023年4月の時点で「Webサイトの開発言語シェア」は以下の通りです。



図A-2-3　Webサイトの開発言語シェア

言語シェアによると圧倒的に1位なのはPHPですが、これは400万以上のサイトで導入されているWordpressがPHPで開発されているためです。本書のアーキテクチャ比較ではJavaが4.4%。Pythonが1.4%で続きます。

情報量

Python（Django）、Java（spring）、PHPを情報量で比較する場合の参考値として「検索数」が挙げられます。これは必ずしもコンテンツの量ではありませんが、それだけ多くの人が検索していることは流通する情報の量が多いと推定されます。

python	201,000
java	90,500
php	49,500

django	18,100
spring	22,200
lalavel	1,300

図A-2-4　検索者数の比較

情報量だけで比較すれば、Python（Django）、Java（Spring）、PHPは概ねPython>Java>PHPになります。フレームワークで比較すると概ね、Djangoと Springが同じ程度で多く、例に挙げたlalavel（PHPのフレームワーク）が少なくなっています。

プロダクトサポート

下記のアーキテクチャではいずれも、セキュリティパッチは重要度に応じて随時提供されます。

Python（Django）のリリースは通常、1年に2回程度行われます。メジャーバージョンアップは約3年ごとに行われます。

Java（spring）のリリースは通常、1年に4回程度行われます。メジャーバージョンアップは2年以上の間隔で行われます。

PHPのリリースは通常、1年に1回程度行われます。メジャーバージョンアップは2年以上の間隔で行われます。

総じて言えることは、これらのフレームワークはどれも頻繁にアップデートが行われ、セキュリティパッチも早急に提供されることが多いです。ただし、Java（spring）のリリース頻度は最も高く、PHPのリリース頻度は最も低くなっています。また、Python（Django）はリリース頻度が中間です。

表A-2-9 3つの言語の比較表

	Python（Django）	Java（spring）	PHP
開発者数	△普通	○多い	◎非常に多い
情報量	○多い	○多い	△普通
プロダクトサポート	○活発	◎非常に活発	△普通

総合的な比較結果（サーバーサイド）

サーバーサイドの比較結果をまとめてみましょう。比較を見ると、概ねJava（Spring）がサーバーサイドアーキテクチャとしての大きな評価を勝ち取っています。ただし、本書では、今後の注目度を考慮してPythonを選択しています。

表A-2-10 3つの言語の比較表

		Python Django	Java Spring	PHP
機能要件適合性		—	—	—
非機能要件適合性	ミッションクリティカル性	○高い	○高い	×低い
	要求パフォーマンス	△普通	○高い	×低い
汎用性		○高い	◎非常に高い	△普通
利用コスト		◎なし	◎なし	◎なし
開発・保守コスト		○高い	△普通	◎非常に低い
流行り	開発者数	△普通	○多い	◎非常に多い
	情報量	○多い	○多い	△普通
	プロダクトサポート	○活発	◎非常に活発	△普通