LightHouse Academy

6.2 Spring 紹介

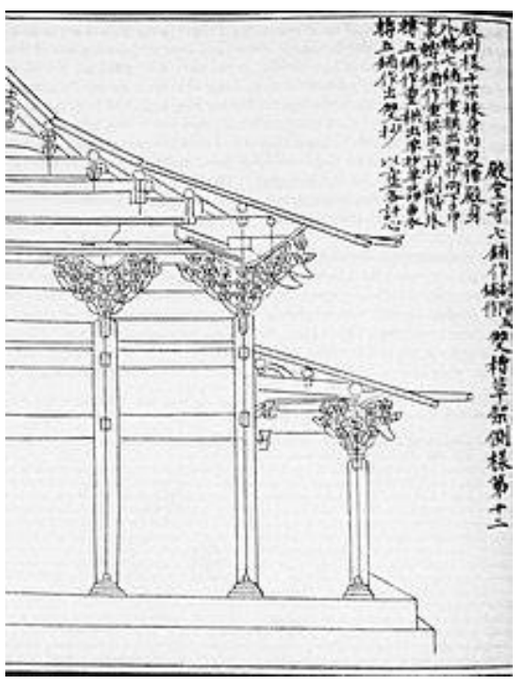
- MVC モデル
- Spring
- Maven

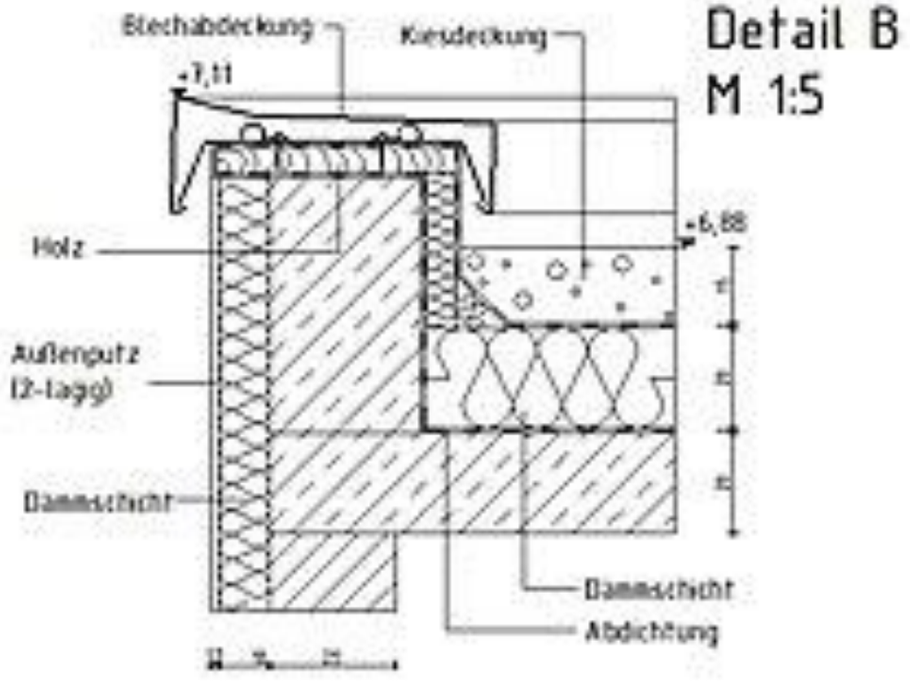
www.lighthouseit.academy

目次

ソフトウェアアーキテクチャ

- **ソフトウェアアーキテクチャ**[Software Architecture]とは、ソフトウェアの全体構造と構成要素に関する抽象的な記述です。ソフトウェアの設計をゼロから始めるのではなく、既存のソフトウェアアーキテクチャパターンから、各部分の機能を実現します。建築のデザインパターンに由来しています。





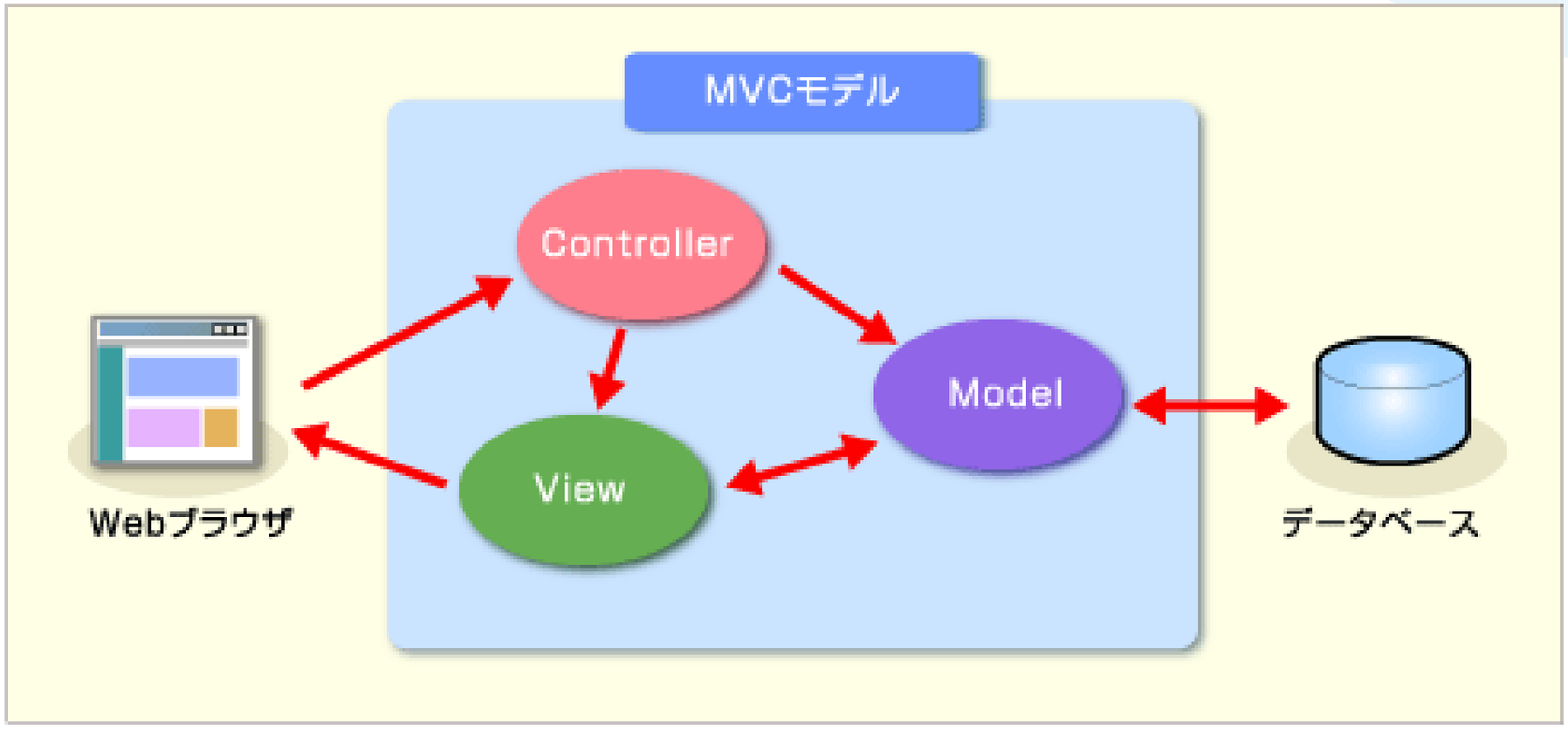
アーキテクチャの例

- 一般的なアーキテクチャパターン：
  - MVC、MVP、MVVM モデル：様々なグラフィカル・アプリケーションに適用されるアーキテクチャの一種。
  - 多層アーキテクチャ：モジュール化したサーバシステム。
  - マスター・スレーブ・アーキテクチャ：サーバ側とクライアント側で構成されるネットワークアーキテクチャ。
  - パイプ・フィルタ・モデル：プロセス間通信を維持する統一のインタフェース。
  - ピア・ツー・ピア・ネットワーク（P2P）：分散型ネットワーク通信モデル。
  - ブラックボードモード： AI 知識表現モデルなど。

- ここでは、**MVC アーキテクチャパターン**について紹介します。

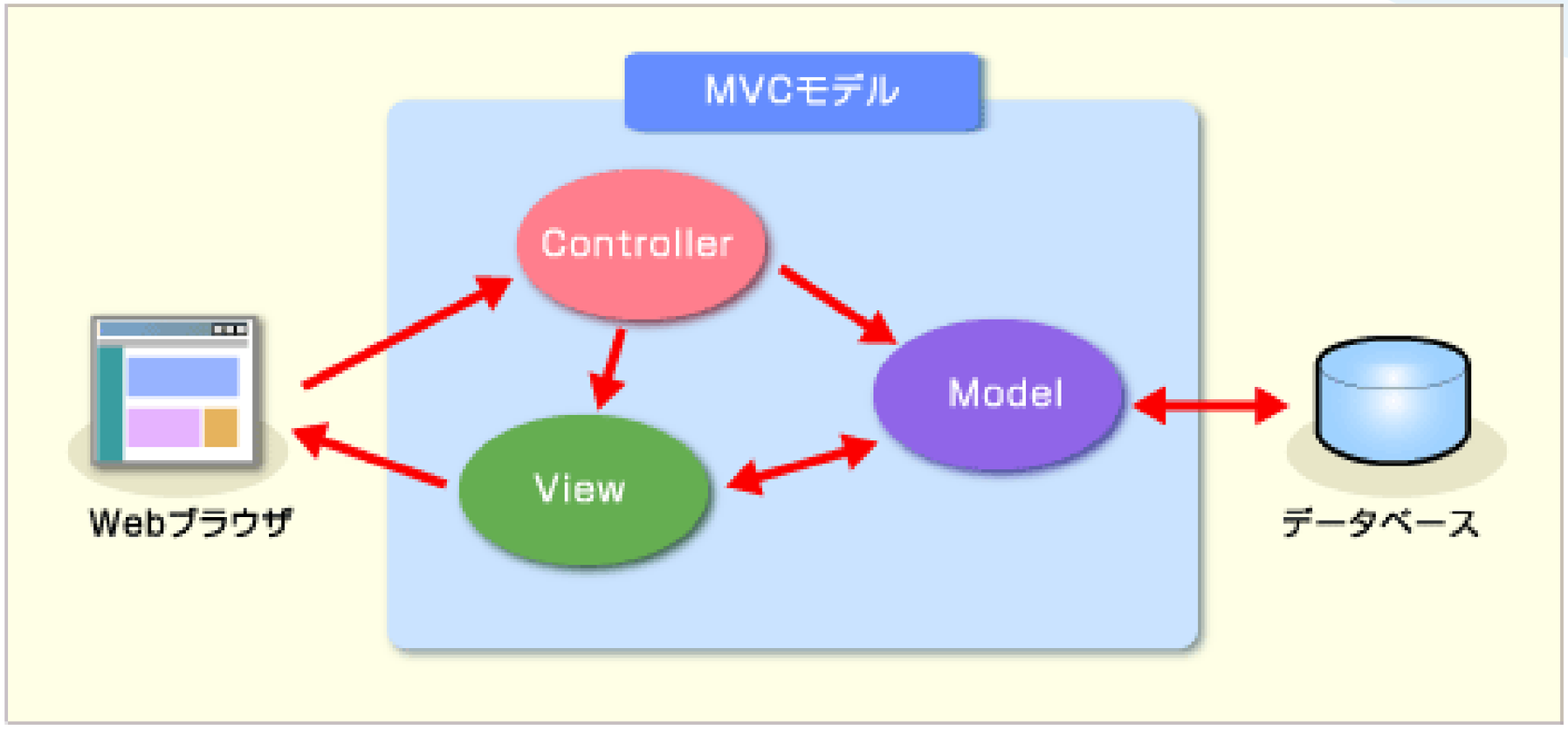
MVC モデル

- **MVC**（Model-view-controller）とは、ソフトウェアシステムを**モデル**[Model]、**ビュー**[View]、**コントローラー**[Controller]の 3 つの基本部分に分けるモデルです。



MVC モデルの歴史

- MVC モデルは、1978 年にパロアルト研究所（Xerox PARC）が Smalltalk というプログラミング言語のために考案したデスクトップソフトウェアのアーキテクチャパターンとして提案されたものです。しかし、インターネットアプリケーションでも広く利用されて、最も主流なウェブアプリケーションアーキテクチャの一つとなっています。



MVC モデルのメリット

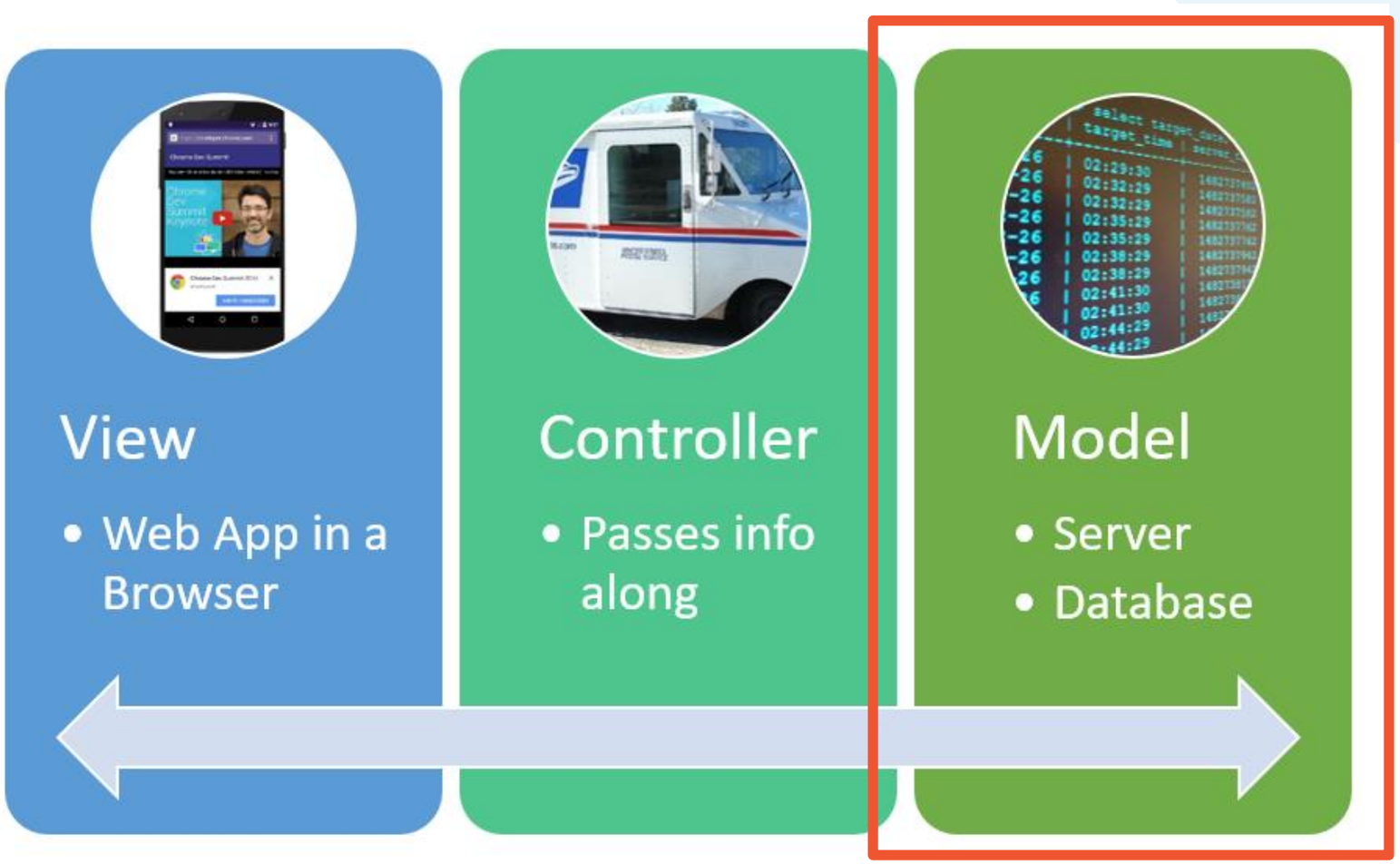
- MVC モデルには、以下のようなメリットがあります：
  - プログラムの**修正・拡張**を簡略化し、プログラムの一部を再利用することが可能になります。

  - より直感的なプログラム構成が可能になります。各部分は自分の**機能**だけを発揮すればよく、各部品の**関係**も明確かつ厳密に定義することができます。

  - 専門家は専門性によってグループ化され、**一部の機能開発に集中**することができます：
    - モデル（Model）：プログラマーによるデータ構造・アルゴリズム設計、データベース専門家によるデータ管理・データベース設計。
    - ビュー（View）：デザイナーによるフロントエンドのユーザーインタフェースデザイン。
    - コントローラ（Controller）：サーバー開発者は、クライアントのリクエストの処理と転送のロジック設計。

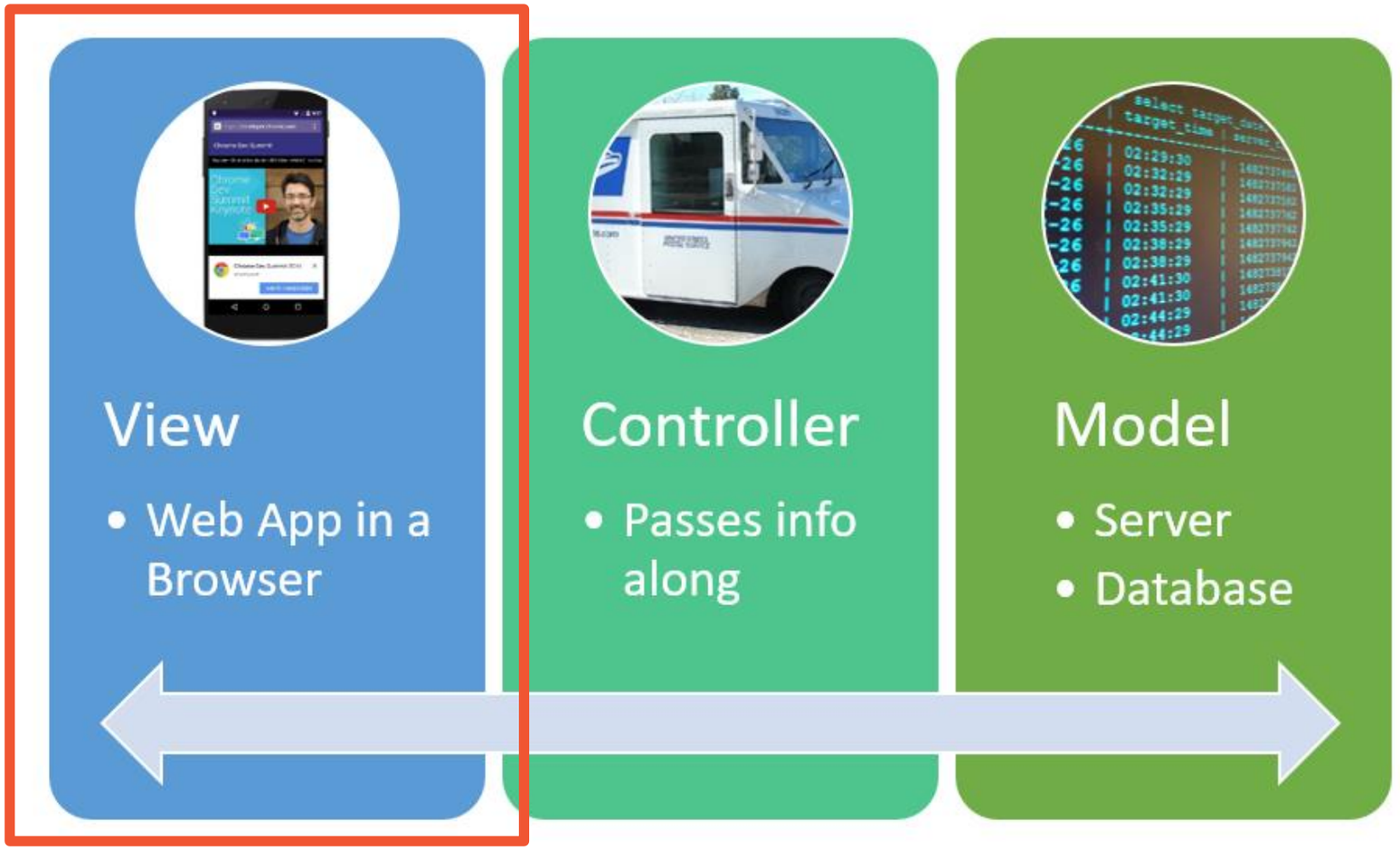
MVC コンポーネント：モデル

- **モデル**は、ビジネスロジックとその処理方法に関連する**データをまとめる**ために使用されます。サーバの場合では、データへのアクセスに使われるオブジェクトや、データへのアクセスに関連するデータベース機能などが含まれます。



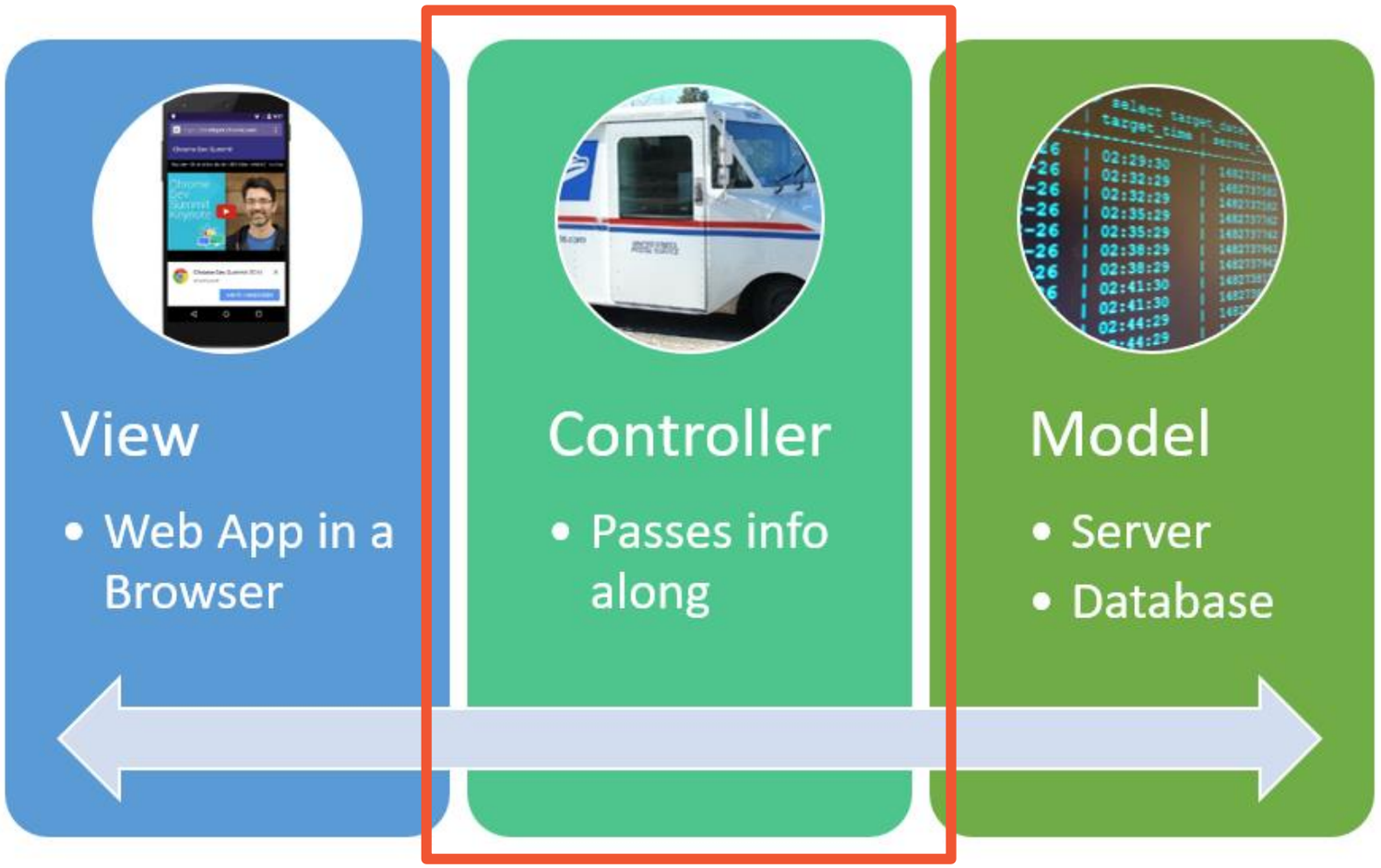
MVC コンポーネント：ビュー

- **ビュー**は、クライアントに転送されたデータを**表示**します。その主な役割は、ユーザが理解できる形でデータモデルを表現することです。また、ユーザから**入力を受け付け**、それをコントローラに渡す役割も果たします。



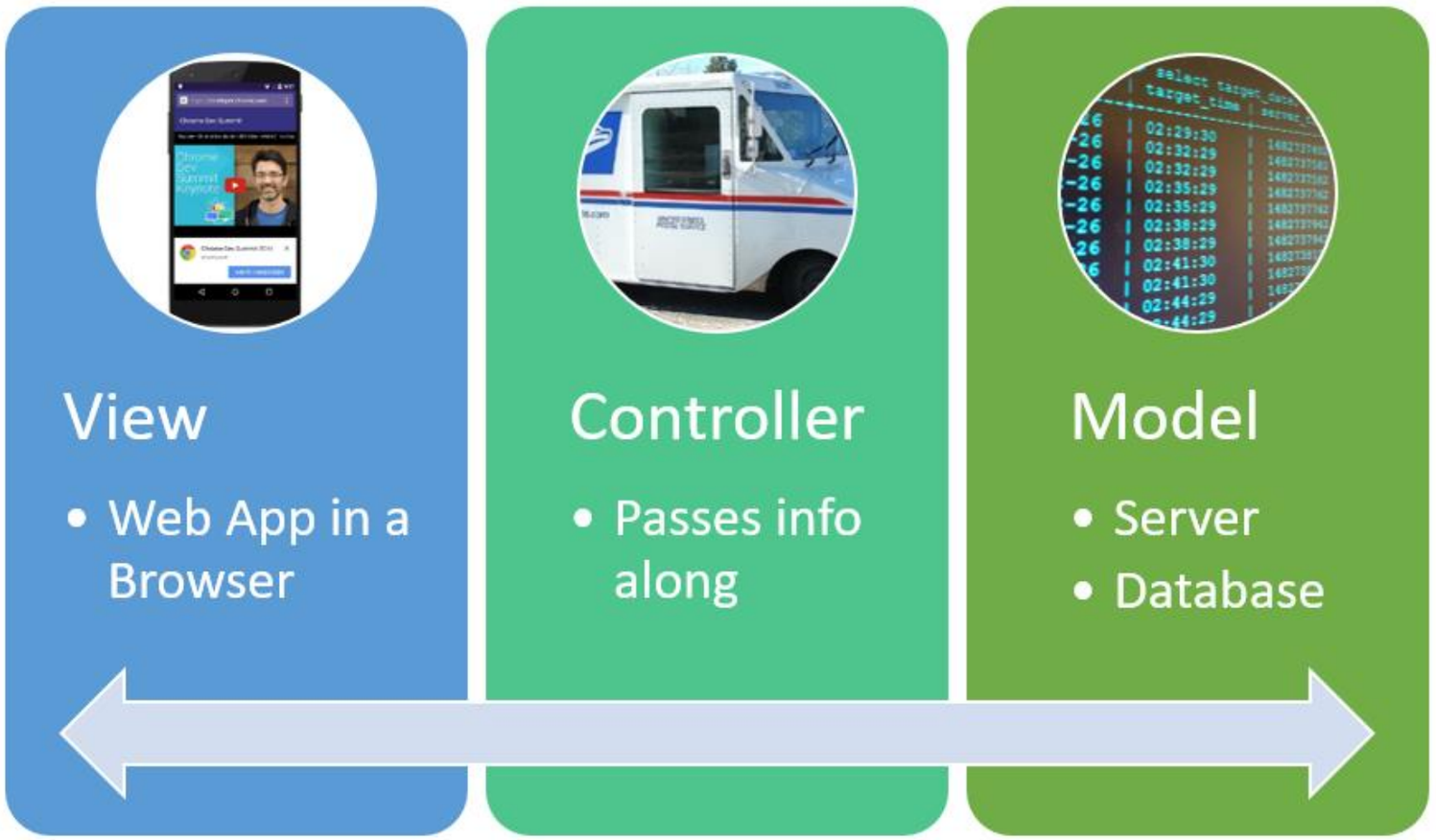
MVC コンポーネント: コントローラ

- **コントローラ**は、異なるコンポーネント間の相互作用を確定し、主要な**ロジックを制御**します。サーバの場合、クライアントから来る HTTP **リクエスト**を処理し、対応するデータモデルを生成し、正しい**レスポンス**をクライアントに返送します。

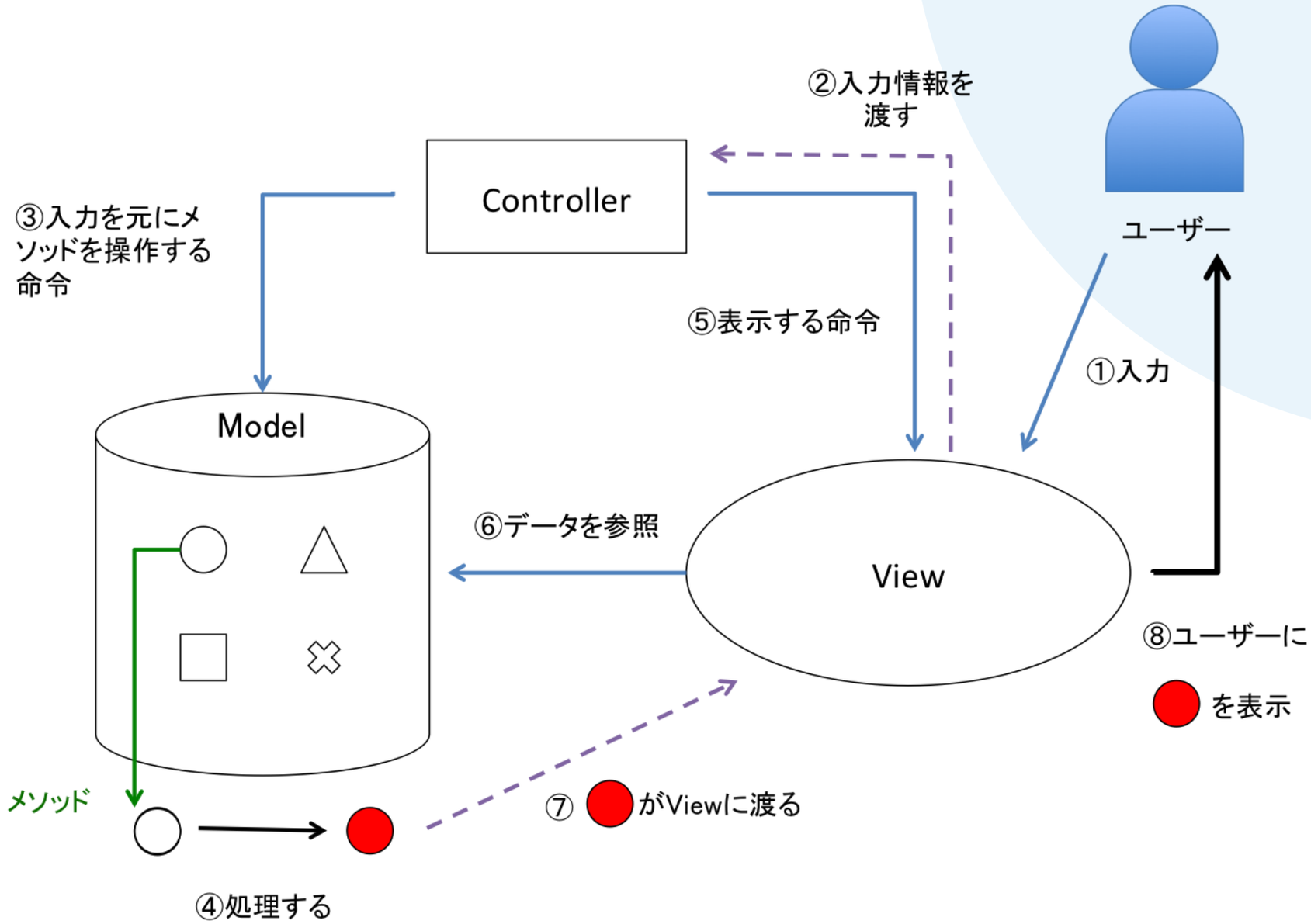


MVC のまとめ

- モデル・Model はプロジェクトのデータモデル、ビュー・View は画面の表示とユーザ入力、コントローラ・Controller はこれらの間のデータ転送を制御します。



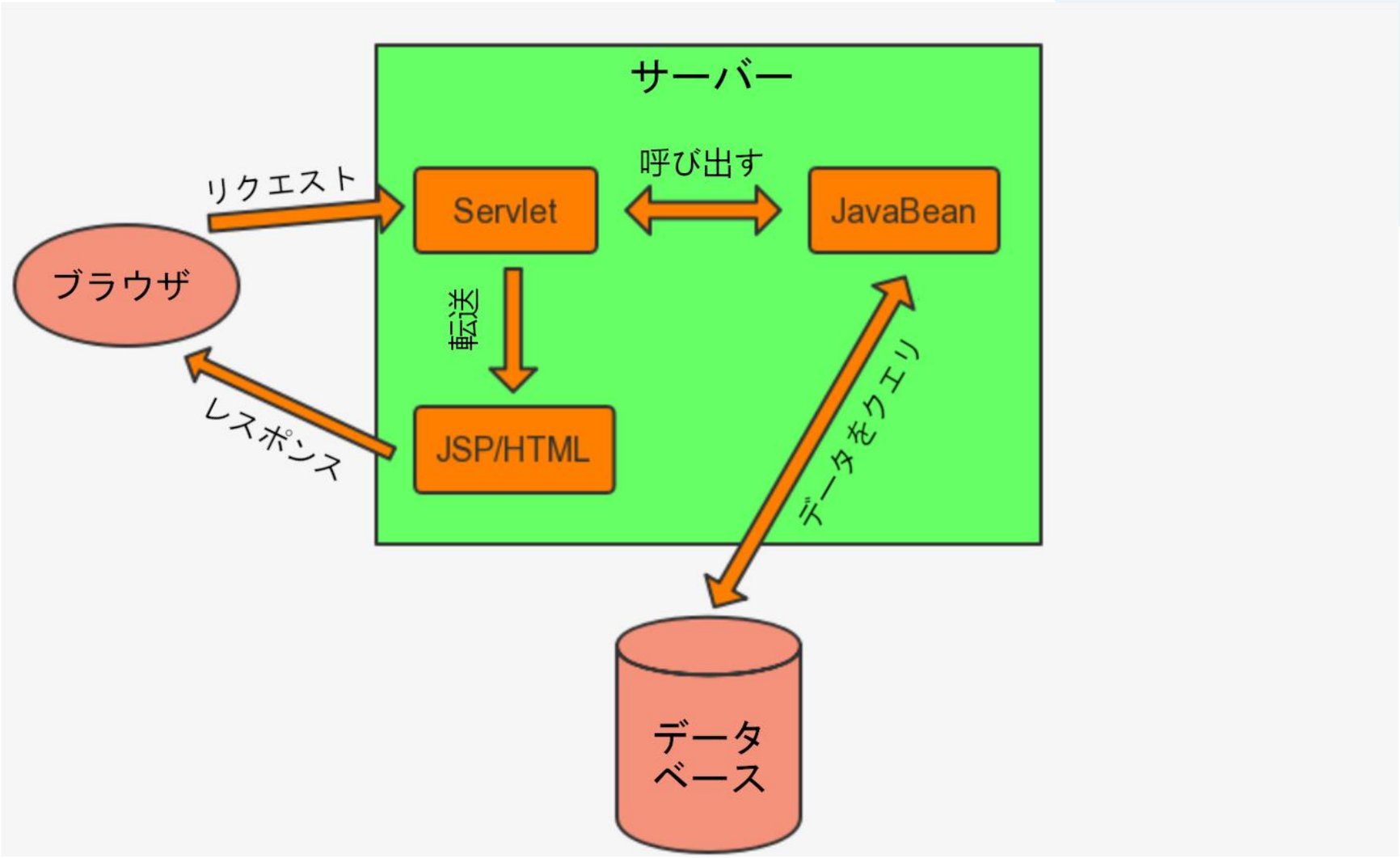
MVC の流れ



参考：https://qiita.com/s_emoto/items/975cc38a3e0de462966a

練習

- この図はある Servlet に基づいた MVC 構造を表現しています。M、V、C はそれぞれどの部分に相当するのでしょうか？



目次

開発フレームワーク

- **フレームワーク**[framework]とは、アプリケーションを開発する際によく使われる汎用的な機能を提供し、実際のアプリケーション開発の基礎とするツールソフトウェアです。

- つまり、フレームワークを使うと、各アプリケーションに共通する部分が最初から既にできています。特定のアプリを作るには、他のアプリと**異なる部分のみ**を書けばよいのです。

開発フレームワークのメリット

- ソフトウェア開発のフレームワークの利用は、以下のようなメリットがあります：
  - 生産性の向上：基本部分の開発を省略することによって、ソフトウェアの**主要なロジックに集中**することができ、「車輪を再発明する」必要がないのです。
  - 質の保証：フレームワークの使用**ルールを守るだけ**で、ある程度の品質を保ったアプリケーションを開発することができます。
  - テスト工程の短縮：一部の機能は、フレームワークにすでに実装され、テストされました。これらの機能を自ら**単体テストする必要がなくなりました**（　§ 6.6.1）。
  - 保守性の向上：ルールに従って作られるため、アプリケーションの全体が把握しやすくなり、コードの**保守性**も高まります。

開発フレームワークのデメリット

- 当然ながら、開発フレームワークを使うには、学習コストが高い、フレームワークの選定に時間がかかる、フレームワーク自体にバグがある、などのデメリットもあります。しかしながら、どっちというとメリットのほうがデメリットを上回るので、現場の開発ではフレームワークなしでの開発はなかなか見れないのでしょう。

一般的な開発フレームワーク

- **Spring**
  - Spring は、Java 向けに設計されたオープンソースの**フルスタック**[Full-stack]アプリケーションフレームワークです。Spring の中核機能の 1 つである**制御の反転**[Inversion of Control, IoC]は、理論上にはあらゆる Java アプリケーションで使用できますが、Spring は Java EE プラットフォームで構築したウェブアプリケーション向けの拡張サポートも豊富に提供しています。Spring は、直接なモデルを実装していませんが、Java コミュニティで普及され最も有名なフレームワークの 1 つになっています。

- **Spring Boot**
  - Spring Boot は、ウェブアプリケーション開発に特化した Spring の派生フレームワークです。Spring と比べて、Spring Boot ではウェブアプリケーション開発に必要な配置はすでに用意された故に、簡単なウェブアプリケーションを素早く開発することができます。

次へ

前へ

- **Apache Struts**
  - Struts は 2001 年から使われている Java のフレームワークで、Apache Software Foundation がスポンサーになっているオープンソースプロジェクトとして有名です。近年、その脆弱性処理の仕組みが不十分なため、他のフレームワークに移行するユーザーも増えています。

- **JSF**
  - JSF（JavaServer Faces）は、2004 年に開発され、Java EE 仕様に採用された Java 標準フレームワークです。Apache Struts と同様の MVC パターンを採用しているが、コンポーネントベースやイベント・ドリブンなどの機能や、従来の静的ページの開発モデルに近い XML ファイルや AJAX 技術のサポートなど、いくつかの相違点があります。

次へ

- **Play Framework**
  - Play Framework は、Scala 言語を使って開発されたウェブアプリケーションフレームワークです。 そのため、Java だけでなく、Scala でも使用することができます。Ruby on Rails や Django といった他のフレームワークの影響を強く受け、軽量かつ効率的なウェブ開発フレームワークです。

- まとめ：上記以外にも多くのフレームワークがあり、「最強のフレームワーク」のようなものは**存在しません**。フレームワークにはそれぞれの特徴があり、システム開発の目的によってどれを選ぶか柔軟に判断する必要があります。

Spring フレームワーク

- **Spring** は、Java EE アプリケーションの迅速な開発を支援するフレームワークです。コンテナやインフラ機能を幅広く提供し、他の汎用ラブライブとも容易に統合できます。

- Spring フレームワークは、**MVC フレームワーク**によってグラフィカルなアプリケーションを開発できます。その上、データベース対応、クラウド、通信セキュリティ、画像処理など、さまざまな機能に対応するコンポーネントを提供しています。

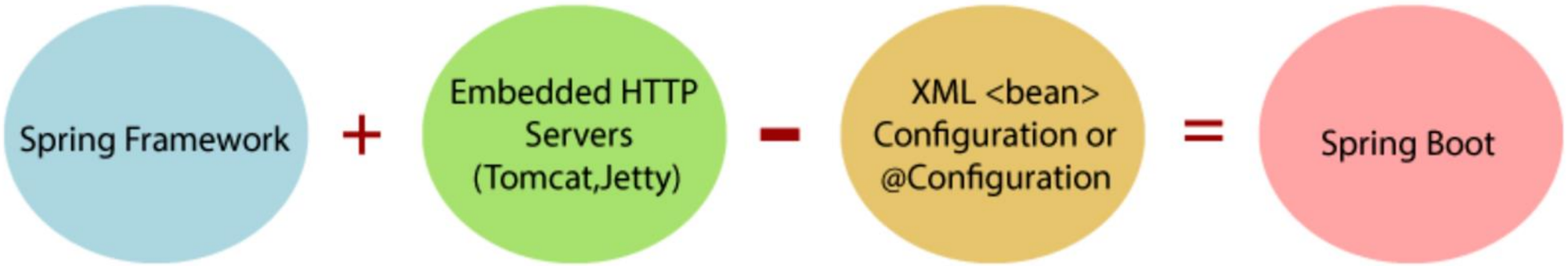
Spring Boot

- Spring フレームワークは強力な機能を持っていますが、最も基本的なウェブアプリケーションでは、利用が必要な機能はかなり限られます。コンポーネントが複雑すぎて、サーバの設定方法が高すぎる専門性を要求してるゆえに、**学習コストはかなり高い**のです。そこで、**Spring Boot** が登場しました。

- **Spring Boot** は、 Spring に基づいたフレームワークであり、Spring の一連のコンポーネントをあらかじめ組み立つことによって、Spring による Java のウェブアプリケーションを**できるだけ少ないコード**と配置で開発できます。

次へ

- Spring Boot の目標は、すぐにも使えるアプリケーション・アーキテクチャを提供することです。Spring Boot の構築済みのアーキテクチャをベースに開発を進められるので、時間と労力の節約になります。



Spring Tool Suite

- **STS**（Spring Tool Suite または Spring Tools）は、Spring が Spring Boot 開発者向けに提供する Eclipse のような**統合開発環境**です。

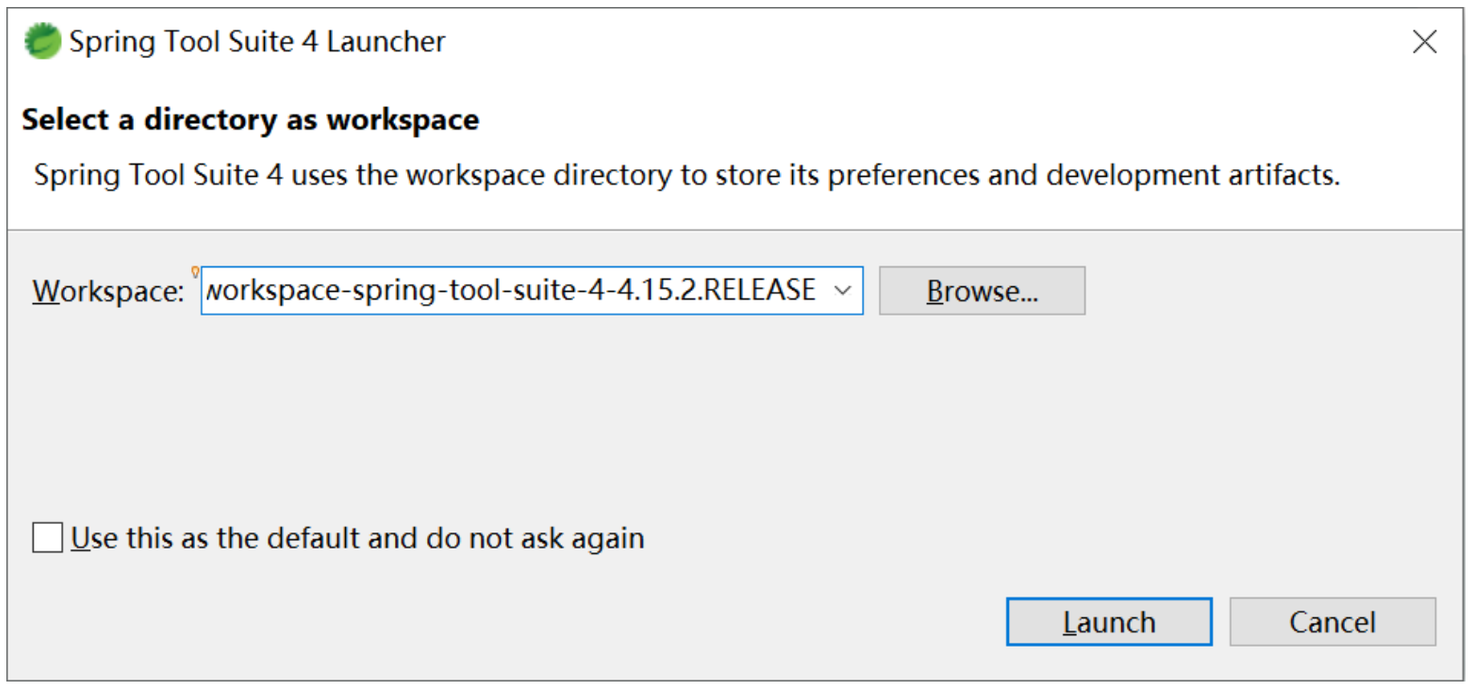
- STS の基本的な使い方は Eclipse と**ほぼ同じ**であり、Eclipse を Spring Boot 開発用に「アップグレード」したものと理解してもいいでしょう。

- STS は特別なインストールを必要とせず、ウェブサイト https://spring.io/tools から対応する .zip ファイルをダウンロードし、ダブルクリックで解凍するだけです。

Spring Tool Suite の起動

- 解凍したら、フォルダ（sts-version-number.RELEASE、例：sts-4.15.2.RELEASE）ごとをパソコンの任意の場所に置いて、その中にある **SpringToolSuite4.exe** をダブルクリックすると起動します。

- 初回起動時には Eclipse と同様に、ワークスペース選択のダイアログが表示されます。必要に応じて選択してください。



目次

Java プロジェクトのビルドの難問

- Java アプリケーションを開発するために必要な手順を考えましょう：
  1. まず、どの**パッケージを依存**（インポート）するかを決定する必要があります。例えば、Servlet を使いたければ Servlet の Jar パッケージをクラスパスに入れべき、Tomcat を使いたければ Tomcat のパッケージをクラスパスに入れる必要があります。
  2. 次に、プロジェクトの**ディレクトリ構造**を決定する必要があります。 例えば、src ディレクトリには Java のソースコード、resources ディレクトリにはプロパティファイルとか。
  3. さらに、JDK のバージョンや、コンパイルとパッケージ化のプロセスなど、**ビルドに関連する配置**を決める必要があります。
  4. 最後に、Eclipse 中だけではなく、サーバーのコマンドラインでもビルドできるように**コマンド**を書かなければなりません。

Maven

- これらの作業は難しいというわけではありませんが、非常に些細で時間をかかります。すべてのプロジェクトに自分で設定するのは大変なことになります。標準化されたプロジェクト管理およびビルドツールが必要なのです。

- **Maven** は、Java プロジェクトに特化して作られた**管理・ビルドツール**です。主な機能は：
  - 標準化されたディレクトリ構造の提供
  - 標準化されたビルドプロセス（コンパイル、テスト、パッケージ、リリース）の提供
  - パッケージ依存関係の管理機能の提供

- 後で作成して行く Spring Boot プロジェクトも、Maven で管理されます。Maven の構造と使い方を見てみましょう。

Maven プロジェクトの構造

- Maven を使用して「a-maven-project」というプロジェクトを管理すると仮定すると、その**ディレクトリ構造**はデフォルトで以下のようになります：

```
a-maven-project
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       └── resources
└── target
```

次へ

Maven 構造の解説

- ここで：
  - pom.xml は Maven **プロジェクトの記述**ファイル
  - src/main/java フォルダには、プログラムの **Java ソースコード**が格納
  - src/main/resources フォルダには、様々な**リソース**ファイル、プロパティファイルが格納
  - src/test/java フォルダには、**テスト用のソース**コードが格納
  - src/test/resources フォルダには**テスト用のリソース**が格納
  - target フォルダーは、コンパイルされ、パッケージ化されたプログラムやパッケージが格納

POM ファイル

- **POM**（Project Object Model）は、Maven の基本コンポーネントで、**XML**（HTML に似っているマークアップ言語）ファイルで記述されます。このファイルはプロジェクトのルートディレクトリに置かれ、**pom.xml** という名であります。

- POM ファイルには、プロジェクトに関する情報と、Maven がプロジェクトのビルドに使用する様々な設定の詳細が含まれています。

- POM で設定できるもの：
  - 名前、バージョンなどのプロジェクトの基本情報、
  - プロジェクトの開発者・開発チームに関する情報、
  - プロジェクトの依存関係情報、
  - プロジェクトのビルドプロセス設定情報、
  - その他プラグイン情報など。

pom.xml の例

- 典型的な pom.xml ファイルの構造は以下のように：

```
1  <project ...>
2    <modelVersion>4.0.0</modelVersion>
3
4    <groupId>com.lighthouseit.java</groupId>
5    <artifactId>hello</artifactId>
6    <version>1.0</version>
7    <packaging>jar</packaging>
8
9    <properties> ... </properties>
10
11   <dependencies>
12     <dependency>
13       <groupId>javax.servlet</groupId>
14       <artifactId>javax.servlet-api</artifactId>
15       <version>3.1.0</version>
16     </dependency>
17     ...
18   </dependencies>
19 </project>
```

POM のバージョン番号（通常は 4.0.0）

プロジェクトの基本情報

Maven の設定情報

プロジェクトの
依存関係情報

プロジェクトの基本情報

- プロジェクトに関する最も重要な基本情報の一部は以下の通りです：
  - **groupId**：**開発会社や組織の名前**。Java のパッケージの名前に似ています。
  - **artifactId**：**プロジェクト名**、Java のクラス名のようなもの。
  - **version**：プロジェクトの**バージョン番号**。
- 各 Maven プロジェクトは、groupId、artifactId、および version によって**一意に識別**することができます。

依存関係管理

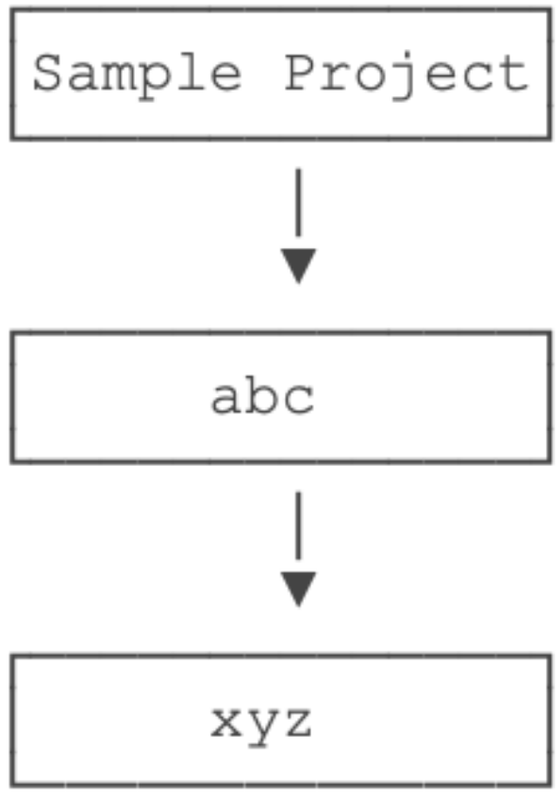
- この 3 つの情報は、他の外部ライブラリに依存する際の判断材料になっています。例えば、Servlet に依存する場合は、このように **<dependency>** を加えます：

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
</dependency>
```

- <dependency> タグで依存関係を宣言すると、Maven は自動的にそのプロジェクトの Jar パッケージをダウンロードし、クラスパスに配置してくれます。

推移的な依存関係

- Maven は、**推移的な依存関係**を管理するプロセスを簡素化します。例えば、私たちのプロジェクトは abc というパッケージに依存し、さらに abc は xyz に依存しています。この場合、私たちのプロジェクトも xyz パッケージに依存する必要があります。



- abc への依存を宣言すると、Maven は自動的に abc と xyz の両方をダウンロードし、abc が依存するものを自分で確認する必要がなくなります。

- もちろん、この xyz が他の何らかのパッケージに依存している場合も、Maven は自動的にそれを識別してダウンロードします。

推移的な依存の例

- 具体的な例を見てみましょう：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>1.4.2.RELEASE</version>
</dependency>
```

- Spring Boot の依存関係を宣言すると、Maven は自動的にそれを解析し、最終的に**約 20〜30** の他の依存関係を必要とすると判明します：

```
spring-boot-starter-web
  spring-boot-starter
    spring-boot
    sprint-boot-autoconfigure
    spring-boot-starter-logging
      logback-classic
        logback-core
        slf4j-api
      jcl-over-slf4j
        slf4j-api
```

```
      jcl-over-slf4j
        slf4j-api
      jul-to-slf4j
        slf4j-api
      log4j-over-slf4j
        slf4j-api
  spring-core
  snakeyaml
spring-boot-starter-tomcat
  tomcat-embed-core
  tomcat-embed-el
  tomcat-embed-websocket
    tomcat-embed-core
jackson-databind
...
```

- これらの依存関係を手動で管理するのは、非常に時間がかかり、ミスの可能性も高いです。

依存関係を検索

- 最後の問題ですが、Tomcat のようなパッケージに依存したい場合、その正確な groupId、artifactId、および version を取得するにはどうしたらよいのでしょうか？

- https://search.maven.org でキーワードで検索し、依存したいパッケージの情報を見つけることができます：

tomcat

Group ID	Artifact ID	Latest Version		Updated
com.weicoder	tomcat	3.5.3	(43)	08-Aug-2022
org.apache.tomcat	tomcat	10.1.0-M17	(99+)	14-Jul-2022
...7.java.dependencies	tomcat	0.1.2	(2)	09-Apr-2022

まとめ

Sum Up

1.MVC アーキテクチャパターンの概念：

① **Model** は、データモデルを意味する。

② **View** は、ビューを表示することを意味する。

③ **Control** は、制御ロジックを意味する。

2.ソフトウェア開発フレームワーク：Spring と Spring Boot フレームワークの概念。

3.Maven の基本的な使用方法。

Light in Your Career.

THANK YOU!