

Spring研修の報告

今町 直登

1. なぜフレームワークが必要なのか？

- 社内システムの課題

2. なぜSpring Frameworkなのか？

- Spring Frameworkを使うメリット・特徴

3. どうやってSpring Frameworkを学ぶのか？

- 書籍、e-learning、研修

4. Spring Frameworkについて

- 簡単な紹介（時間があれば）

5. Spring研修を通して得たもの

- 書籍やe-Learningでは得られない、貴重な体験・学び

なぜフレームワークが必要なのか

現状

・既存システムの保守性の悪さ

例) **MVCモデルの破綻**（ビジネスロジックの散らばり・もつれ）

- ① JSPファイル (View)
- ② Servletクラス (Front Controller)
- ③ ビジネスロジッククラス (Controller)
- ④ データアクセスクラス [DAO] (Model)
- ⑤ データ転送クラス [DTO] (Model)

の各階層の役割があいまいになってしまっている。

⇒ソースコードが難解になり、解析に時間がかかる。

⇒モジュールの疎結合化ができていないので、改修ポイントが増える。

⇒システムの品質の低下・改修スピードの低下を招く。

⇒すべてがコストとして跳ね返ってくる...

（初期設計時の設計思想があいまいであるところか来ている？）

なぜフレームワークが必要なのか

現状

・既存システムの保守性の悪さ

例) MVCモデルの破綻（ビジネスロジックの散らばり・もつれ）

- ① JSPファイル (View)
- ② Servletクラス (Front Controller)
- ③ ビジネスロジッククラス (Controller)
- ④ データアクセスクラス [DAO] (Model)
- ⑤ データ転送クラス [DTO] (Model)

の各階層の役割があいまいになってしまっている。

⇒ソースコードが難解になり、解析に時間がかかる。

⇒モジュールの疎結合化ができていないので、改修ポイントが増える。

⇒システムの品質の低下・改修スピードの低下を招く。

⇒すべてがコストとして跳ね返ってくる...

（初期設計時の設計思想があいまいであるところか来ている？）

特定の設計思想（フレームワーク）を導入することで解決

なぜSpring frameworkなのか

疑問点

新規にWebアプリケーションの開発を行うとして、



バックエンドを開発する上で
最良の言語・フレームワークは何か？

なぜSpring frameworkなのか

疑問点

新規にWebアプリケーションの開発を行うとして、



バックエンドを開発する上で
最良の言語・フレームワークは何か？

言語・フレームワーク選びで何を判断基準とすべきか？

フレームワークの採用条件（個人的な考え）



ソースコードの
保守性



技術の汎用性



将来性

①ソースコードの保守性

- テスト容易性、コードの再利用性、各モジュールの疎結合性など

②技術の汎用性

- 使われている技術・設計思想はその他の言語・フレームワークに応用できるものか
- フレームワーク固有となる技術が使われていないかどうか

③将来性

- 5, 10年先も生き残るフレームワークかどうか
- フレームワーク固有の技術が革新的であり、今後のデファクトスタンダードとなりうるか

Spring Frameworkの特徴



将来性

(1) エンタープライズ向け

Webアプリケーション開発の定番

⇒ StrutsやSeasarといった主要なフレームワークがすべてEOLとなり、JavaのWeb系フレームワークは事実上Springに一本化されつつある

(2) 新しい技術・ニーズに即座に対応

⇒ セキュリティ対策 (Spring Security)

⇒ クラウド対応 (Spring Cloudなど)

⇒ リアクティブプログラミングなどの新技術の導入

(3) オープンソース & 企業側のバックアップ

⇒ オープンソースで開発

⇒ Pivotal社 (Dellの孫会社) が開発をサポート

⇒ 多くの日本企業での導入実績

Spring Frameworkの特徴



ソースコードの
保守性

- ・ 依存オブジェクトの注入 (DI)
- ・ アスペクト指向プログラミング (AOP)

⇒モジュール間を疎結合化する仕組み・設計思想
⇒テスト容易性、保守性、再利用性の高いコードを書く
上で重要なコア技術

Spring Frameworkの特徴



ソースコードの
保守性

- ・ 依存オブジェクトの注入 (DI)
- ・ アスペクト指向プログラミング (AOP)

⇒モジュール間を疎結合化する仕組み・設計思想
⇒テスト容易性、保守性、再利用性の高いコードを書く
上で重要なコア技術



技術の汎用性

- ・ 学びが得られるフレームワーク

⇒DIやAOPは、Ruby on Rails, Angular (Javascript),
StrutsやSeasar (Java)などの多くのフレームワークで
導入されている仕組み
⇒これらの考え方は別言語・フレームワークでも
応用可能な設計思想

Spring Frameworkの欠点



学習コスト

・ 学習コストが高い

- ⇒ 初学者にとって、DIの概念を理解することが難しい...
- ⇒ 単一のフレームワークではなく、
複数のコンポーネント（技術）の集合体



技術選定の
難しさ

・ 要件・人員に合わせた技術選定が必要

- ⇒ 単一の選択肢をSpring Frameworkを提供してくれるわけではなく、自由度の高いフレームワーク。
- ⇒ 例えば、DBアクセスの技術として、Spring JDBC, Spring Data JPA, MyBatis, Doma2, Spring Data JDBCなどの選択肢がある。
- ⇒ 要件・人員の技術力に応じて選択肢しなくてはならない

どうやってSpring frameworkを勉強するの？

(1) 書籍

- ・日本語のSpringの関連書籍はどれも情報が古く、最新の開発手法を学ぶ上で不適切。

⇒すでにレガシーな設定方法（XML）、セキュリティ脆弱性が報告されているテンプレートエンジン（JSP）を例にしているなど。

(2) e-Learning (Udemyなど)

- ・基本的に、英語の授業（新しいものを学ぶ上では敷居が高い）。
- ・レッスンの質にバラつきあり。

(3) Pivotal認定 Core Springトレーニングコース（4日間）

- ・日本で3人ほどしかいない認定トレーナーのもと、短期集中で学べる。
- ・Pivotal公式のコースで、Springの最新の情報を体系的に学べる。

⇒2017年夏リリースのSpring Framework v5.xに対応

<https://www.casareal.co.jp/ls/service/openseminar/pivotal/p016>



1. Spring Frameworkとは

- 複数のコンポーネントの集合体

2. 依存オブジェクトの注入 (DI: Dependency Injection)

- モジュールを疎結合に

3. アスペクト指向プログラミング (AOP)

- 横断的関心事(ログ/セキュリティ/例外処理)をビジネスロジックから分離

4. REST API

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

Spring Frameworkとは

- 複数のコンポーネントの集合体

- Spring Frameworkは単一のフレームワークではなく、複数のコンポーネント（技術）の集合体。
- 複数のコンポーネントを利用して、1つのWebアプリケーションを組み上げるイメージ。



Spring Security



Spring Data



Spring AMQP



Spring Batch



Spring Boot



Spring Framework



Spring Cloud



Spring Social



Spring Mobile

Spring Frameworkとは

- 複数のコンポーネントの集合体

- それぞれの技術を学ぶのに、研修だと2〜3日以上。
- DBアクセスだけでも、Spring MVC, Spring JDBC, Spring Data JPA, Spring Data JDBC, MyBatis, Doma2などの複数の選択がある。



Spring Security



Spring Data



Spring AMQP



Spring Batch



Spring Boot



spring
by Pivotal™

Spring Framework



Spring Cloud



Spring Social



Spring Mobile

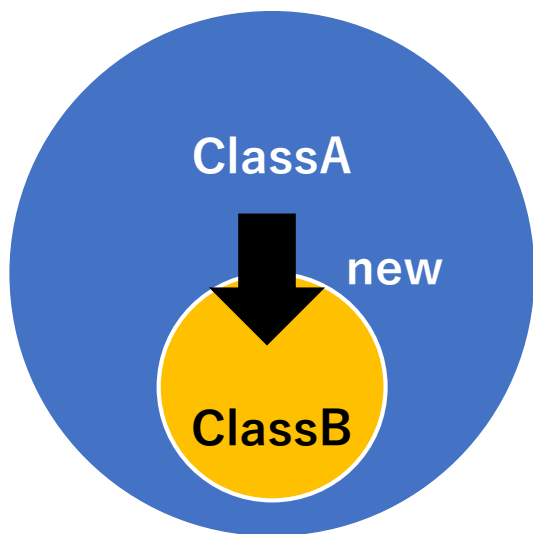
ごく一部

依存オブジェクトの注入（DI: Dependency Injection）とは

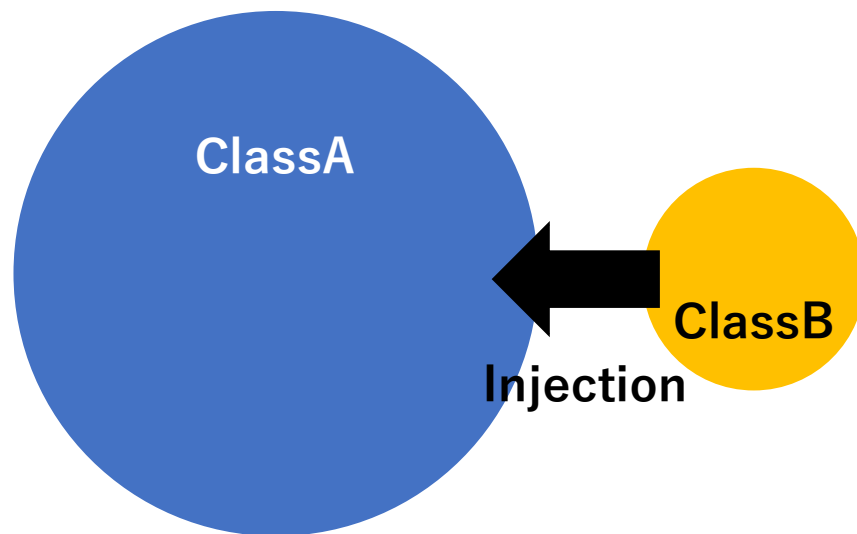
- モジュールを疎結合に

- DIとはデザインパターンの1つ。

従来のClass設計



DIを使ったClass設計

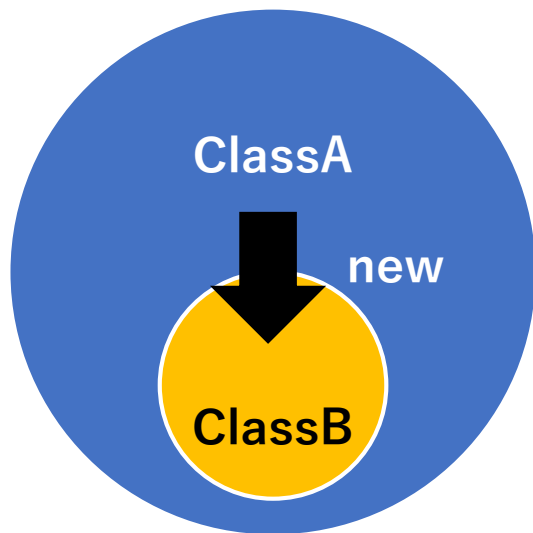


必要なインスタンス（依存性）を外から代入すること。

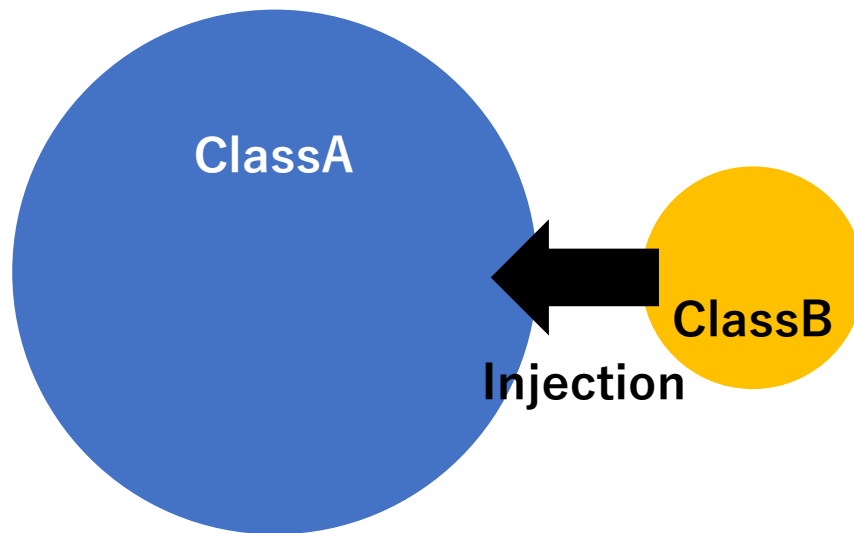
依存オブジェクトの注入（DI: Dependency Injection）とは

- モジュールを疎結合に

従来のClass設計



DIを使ったClass設計

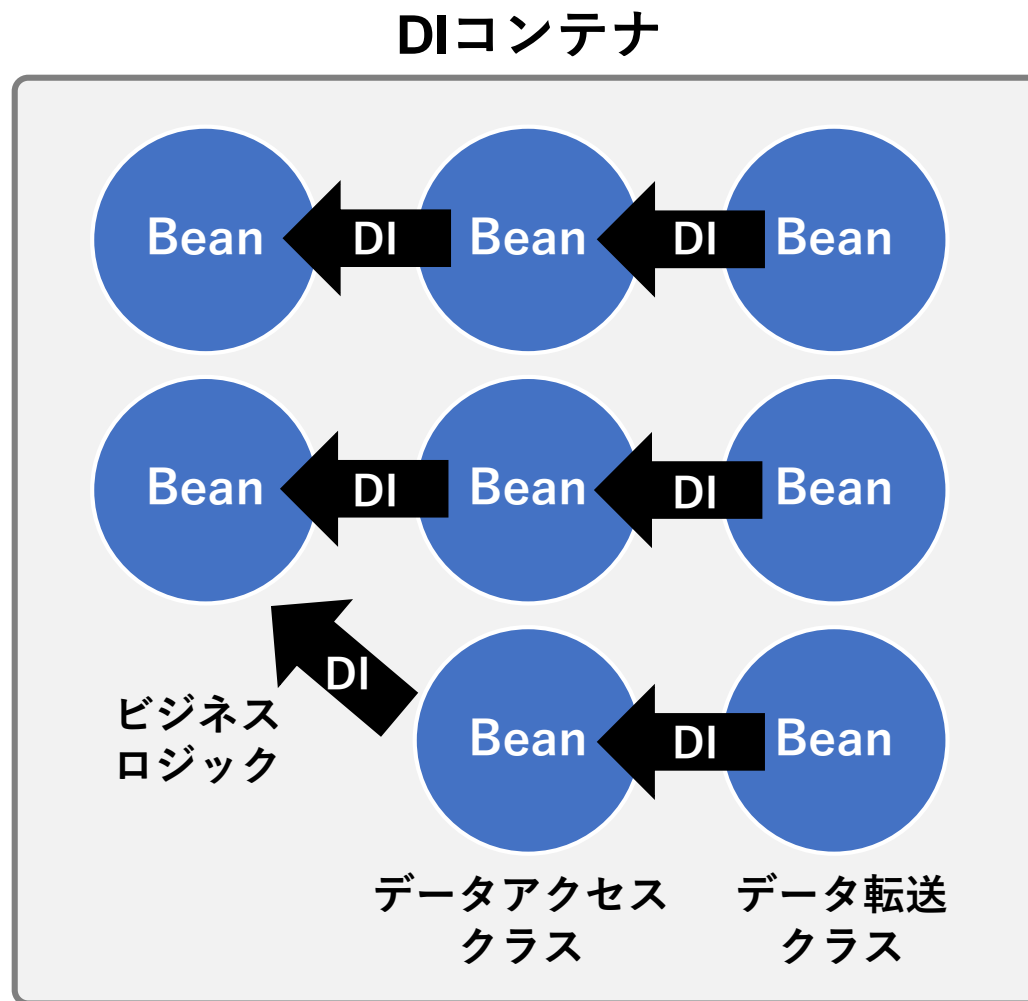


- ClassAの動作は、ClassBに依存している。
- ClassBが完成しないと、ClassAを動かすことができない。

必要なインスタンス（依存性）を外から代入すること。

依存オブジェクトの注入 (DI: Dependency Injection) とは

- DIコンテナ



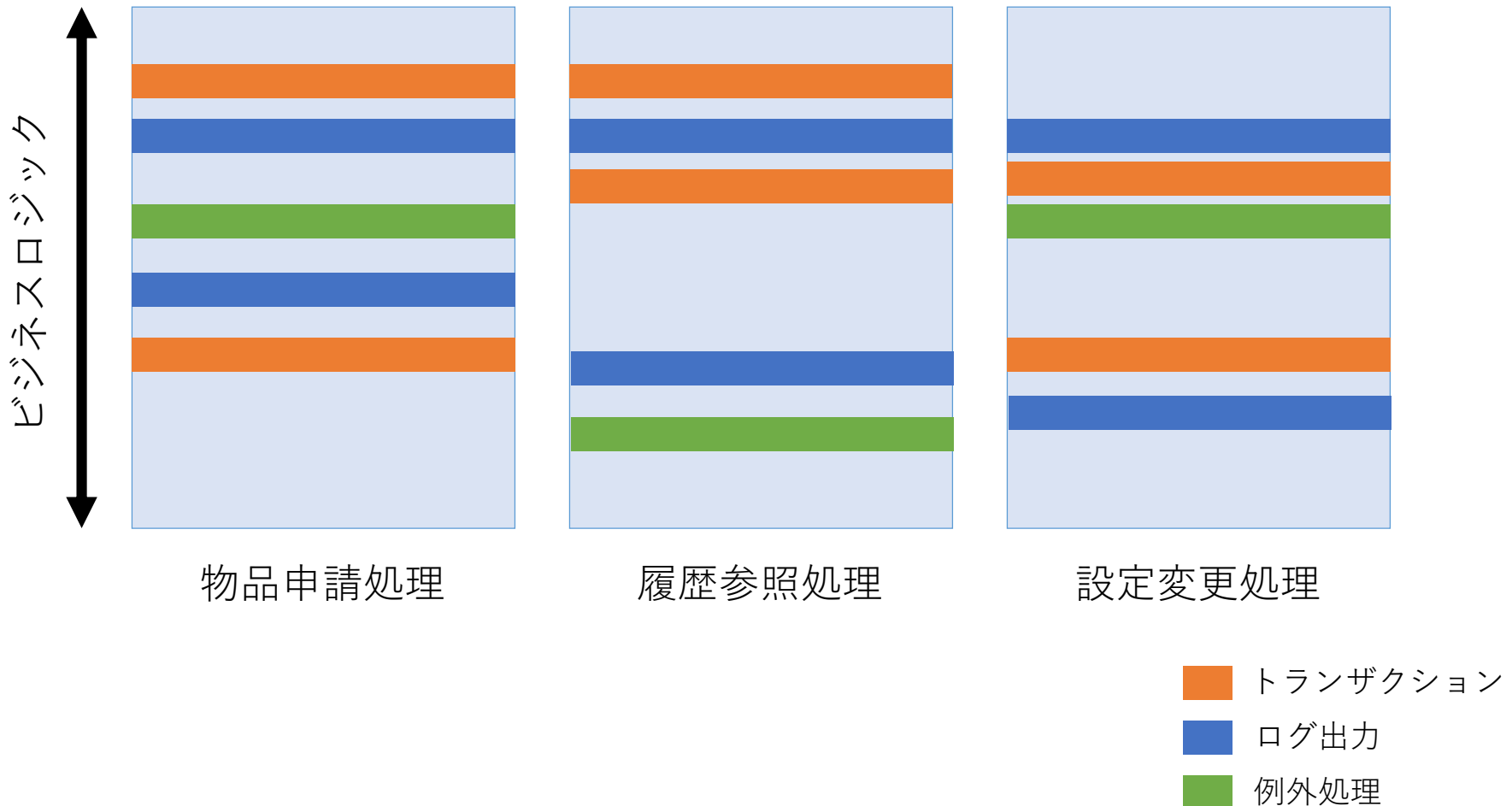
Springが内部で持っているインスタンス (Bean) の格納庫

アスペクト指向プログラミング（AOP）とは

- 横断的関心事（ログ/セキュリティ/例外処理など）をビジネスロジックから分離

問題点

- ・ コードの散らばり・もつれ（Code scattering / Code tangling）

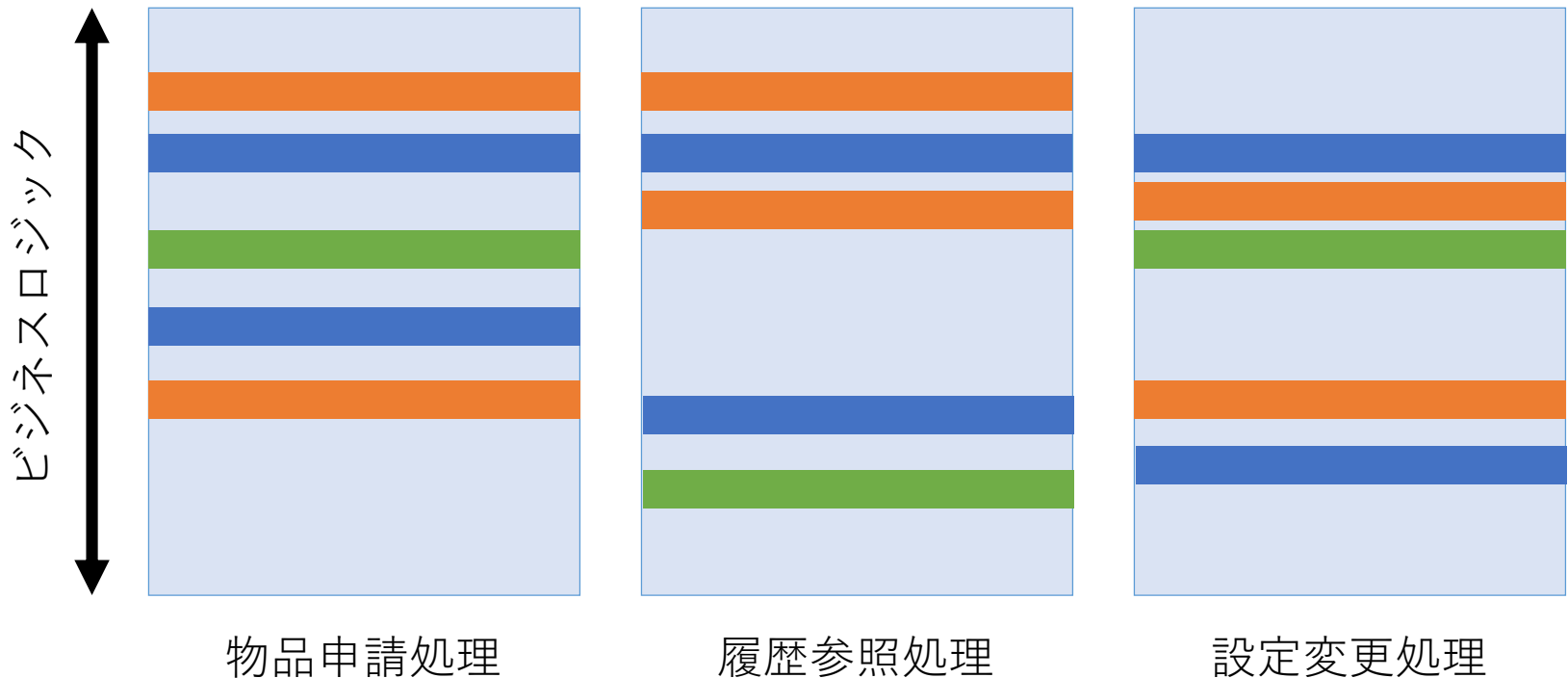


アスペクト指向プログラミング（AOP）とは

- 横断的関心事（ログ/セキュリティ/例外処理など）をビジネスロジックから分離

問題点

- ・ コードの散らばり・もつれ（Code scattering / Code tangling）

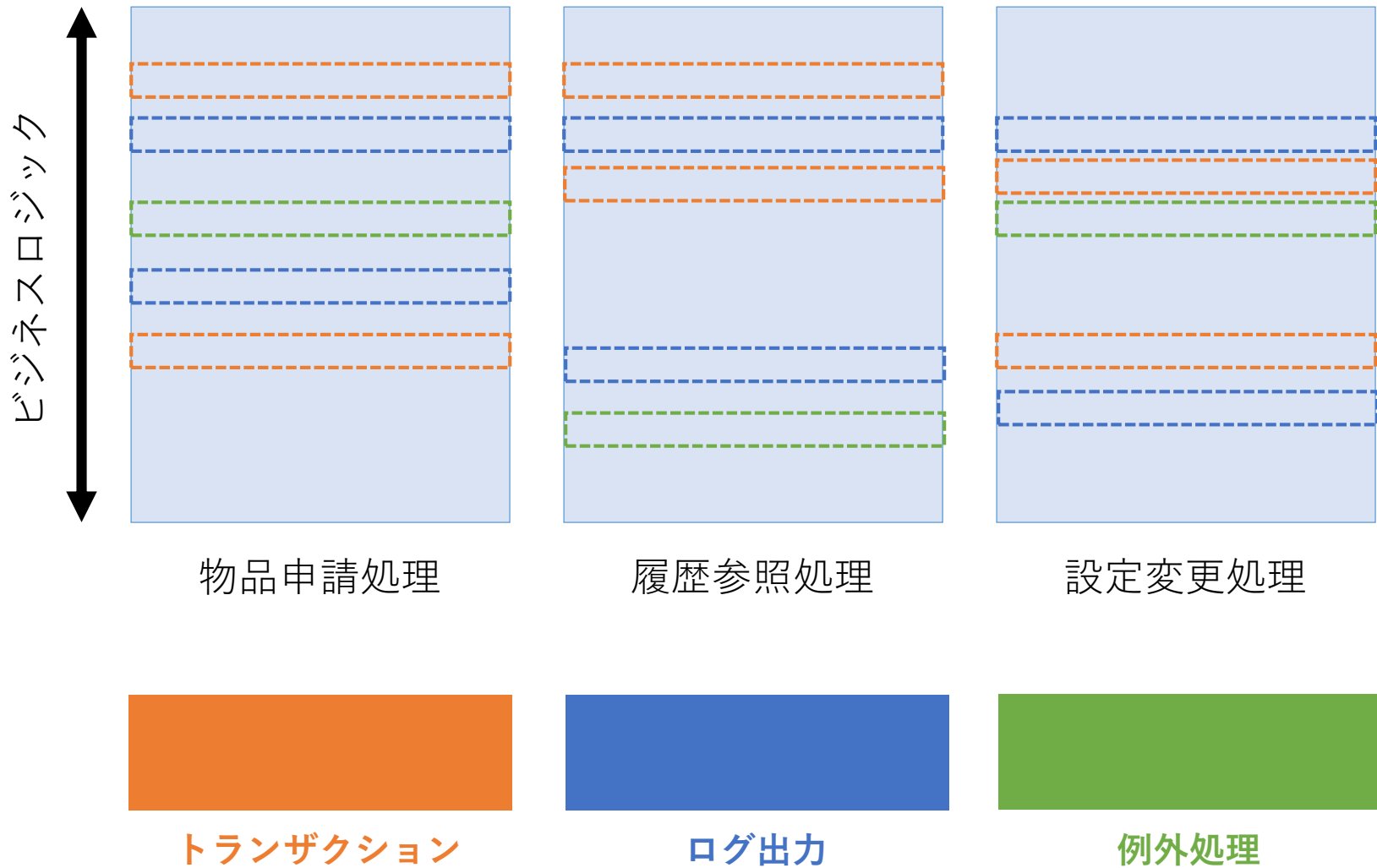


ビジネスロジックに別の処理が入り込んでいる。
⇒結果として、処理内容がわかりにくくなる...

オレンジ トランザクション
青 ログ出力
緑 例外処理

アスペクト指向プログラミング（AOP）とは

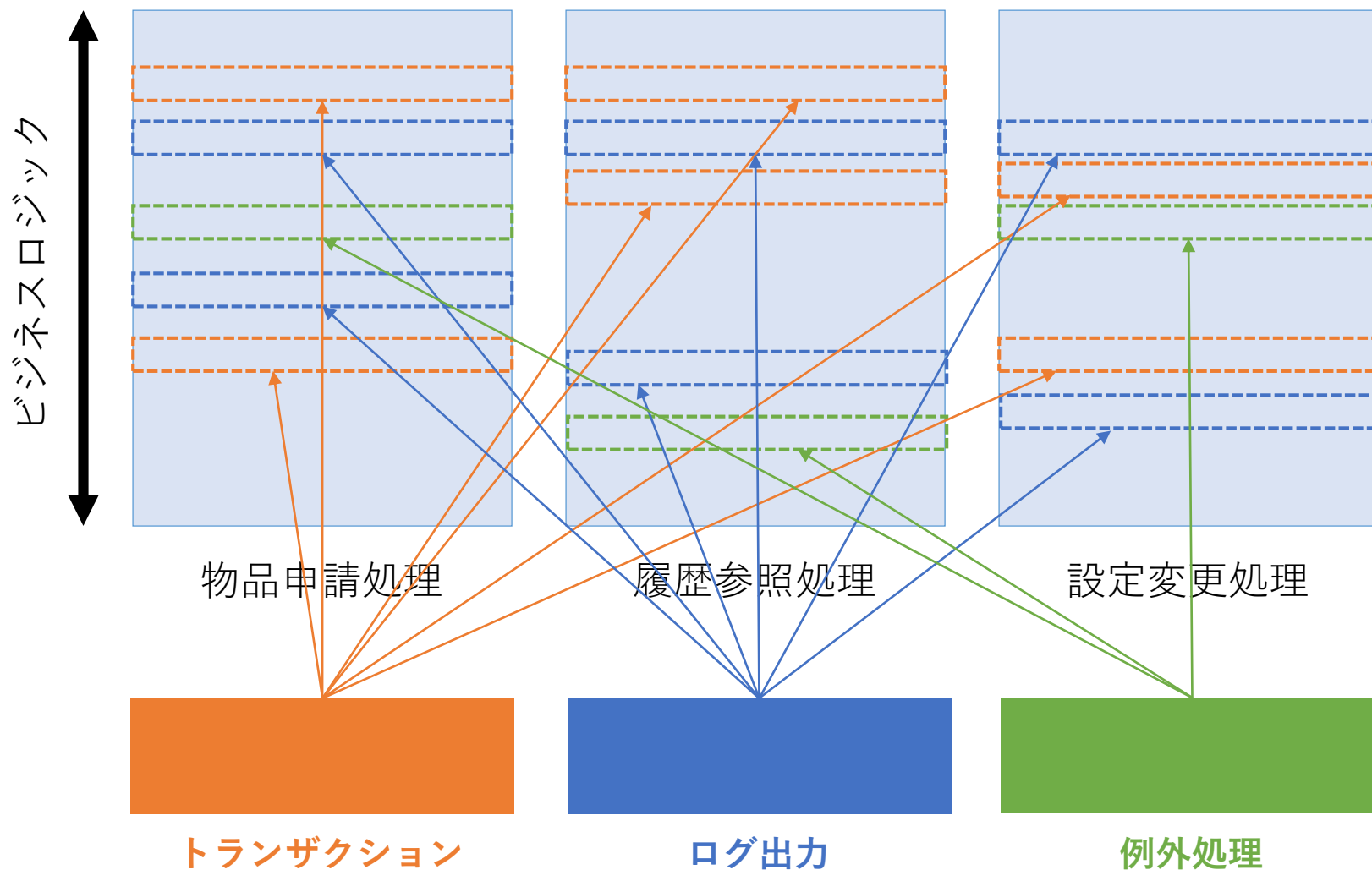
- 横断的関心事（ログ/セキュリティ/例外処理など）をビジネスロジックから分離



横断的関心事はビジネスロジックに組み込まないという戦略。

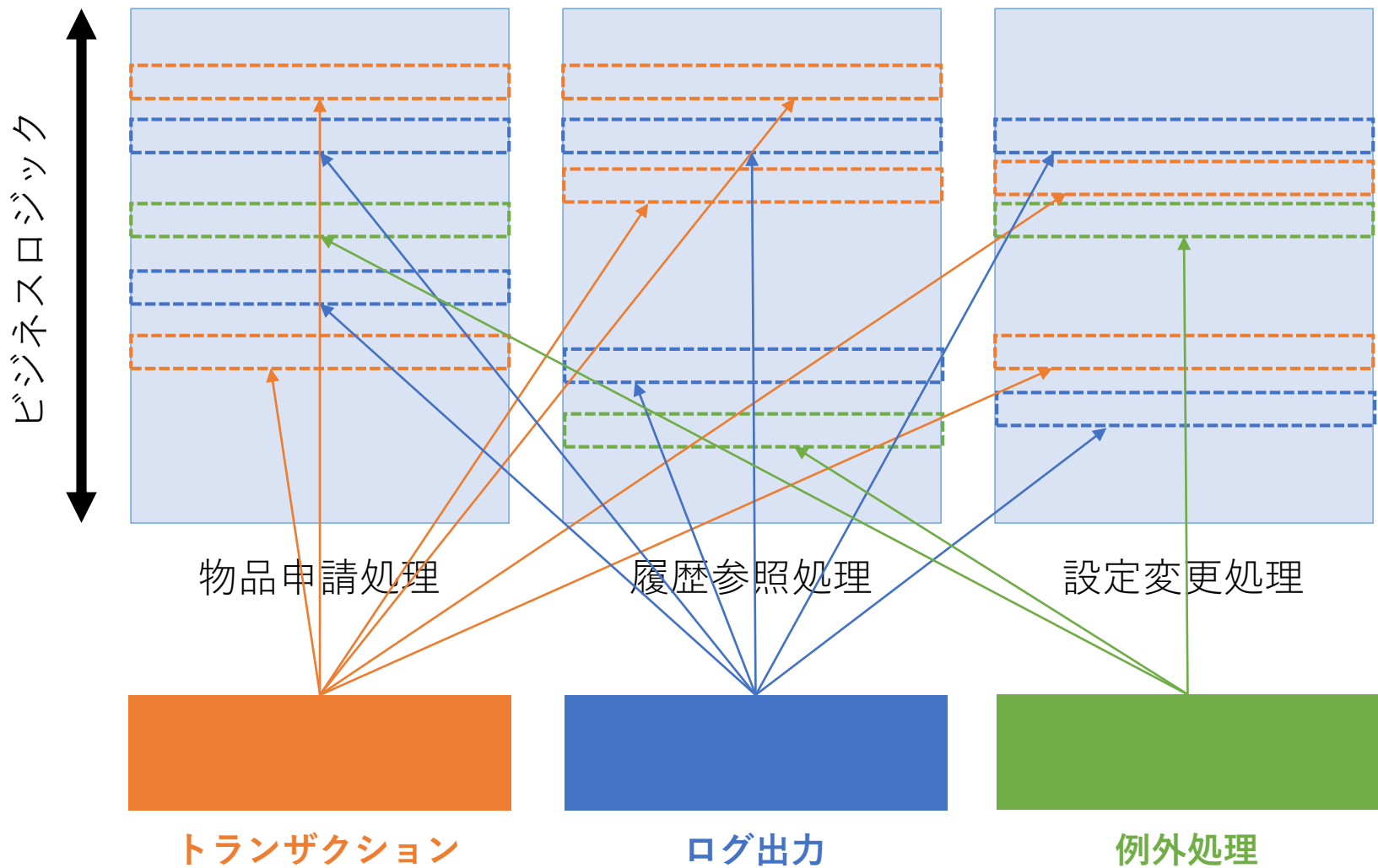
アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離



アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離



横断的関心事を割り込み処理としてビジネスロジックに組み込む。
⇒特定のメソッドを通過したときに割り込み処理が発生。

アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離

(1) 割り込み処理を記述したクラス

```
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @After("execution(void set*(*))")
    public void trackChange() {
        logger.info("フィールド変数に値が格納されました。");
    }
}
```

(2) ビジネスロジッククラス (仮) 本当はデータ転送クラスです...

```
public class AuthorService {
    private String firstName;
    private String LastName;

    public void setFirstName(String fn) {this.firstName = fn}
    public void setLastName(String ln) {this.lastName = ln}
}
```


アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離

(1) 割り込み処理を記述したクラス

```
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @After("execution(void set*(*))")
    public void trackChange() {
        logger.info("フィールド変数に値が格納されました。");
    }
}
```

setXxxメソッドの処理後に実行

(2) ビジネスロジッククラス (仮)

```
public class AuthorService {
    private String firstName;
    private String LastName;

    public void setFirstName(String fn) {this.firstName = fn}
    public void setFirstNme(String ln) {this.lastName = ln}
}
```

アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離

(1) 割り込み処理を記述したクラス

```
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @After("execution(void set*(*))")
    public void trackChange() {
        logger.info("フィールド変数に値が格納されました。");
    }
}
```

setXxxメソッドの処理後に実行

(2) ビジネスロジッククラス (仮)

```
public class AuthorService {
    private String firstName;
    private String LastName;

    public void setFirstName(String fn) {this.firstName = fn}
    public void setFirstName(String ln) {this.lastName = ln}
}
```

割り込みの処理対象となるメソッド

アスペクト指向プログラミング (AOP) とは

- 横断的関心事 (ログ/セキュリティ/例外処理など) をビジネスロジックから分離

(1) 割り込み処理を記述したクラス

```
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());
```

```
@After("execution(void set*(*))")
```

setXxxメソッドの処理後に実行

```
public void trackChange() {
    logger.info("フィールド変数に値が格納されました。");
}
```

**AOPを使うことで、ビジネスロジック作成時に
ログや例外処理に気を配る必要がなくなる！**

(2) エンティティサービス (仮)

```
public class AuthorService {
    private String firstName;
    private String LastName;
```

割り込みの処理対象となるメソッド

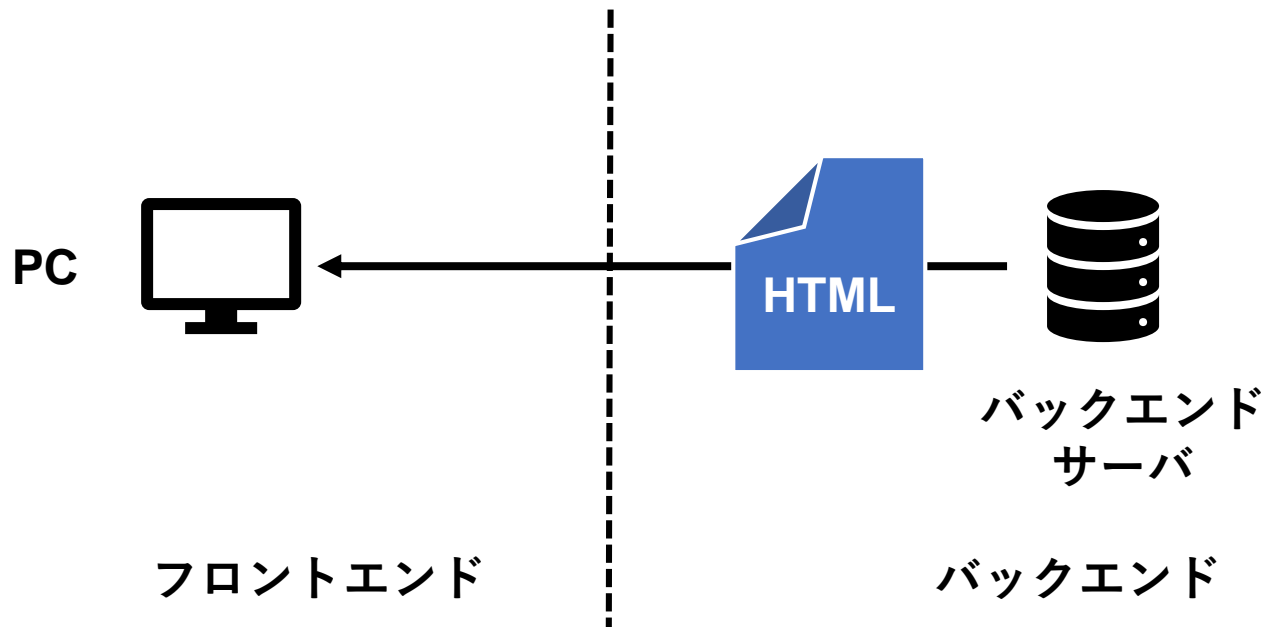
```
public void setFirstName(String fn) {this.firstName = fn}
public void setLastName(String ln) {this.lastName = ln}
```

```
}
```

REST APIとは

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

従来のWebアプリケーション

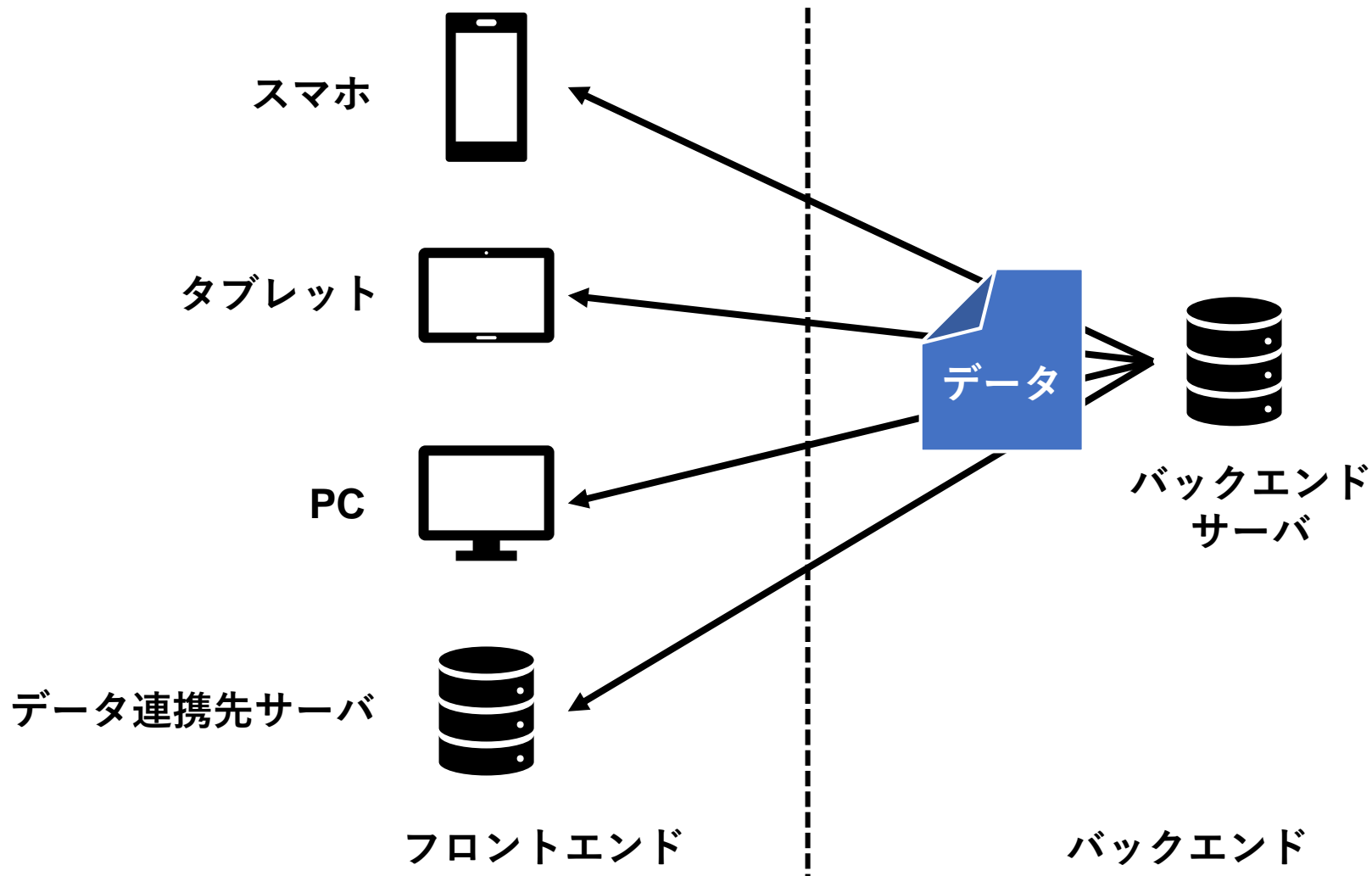


サーバ側でHTMLをレンダリングしてPCへ送る

REST APIとは

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

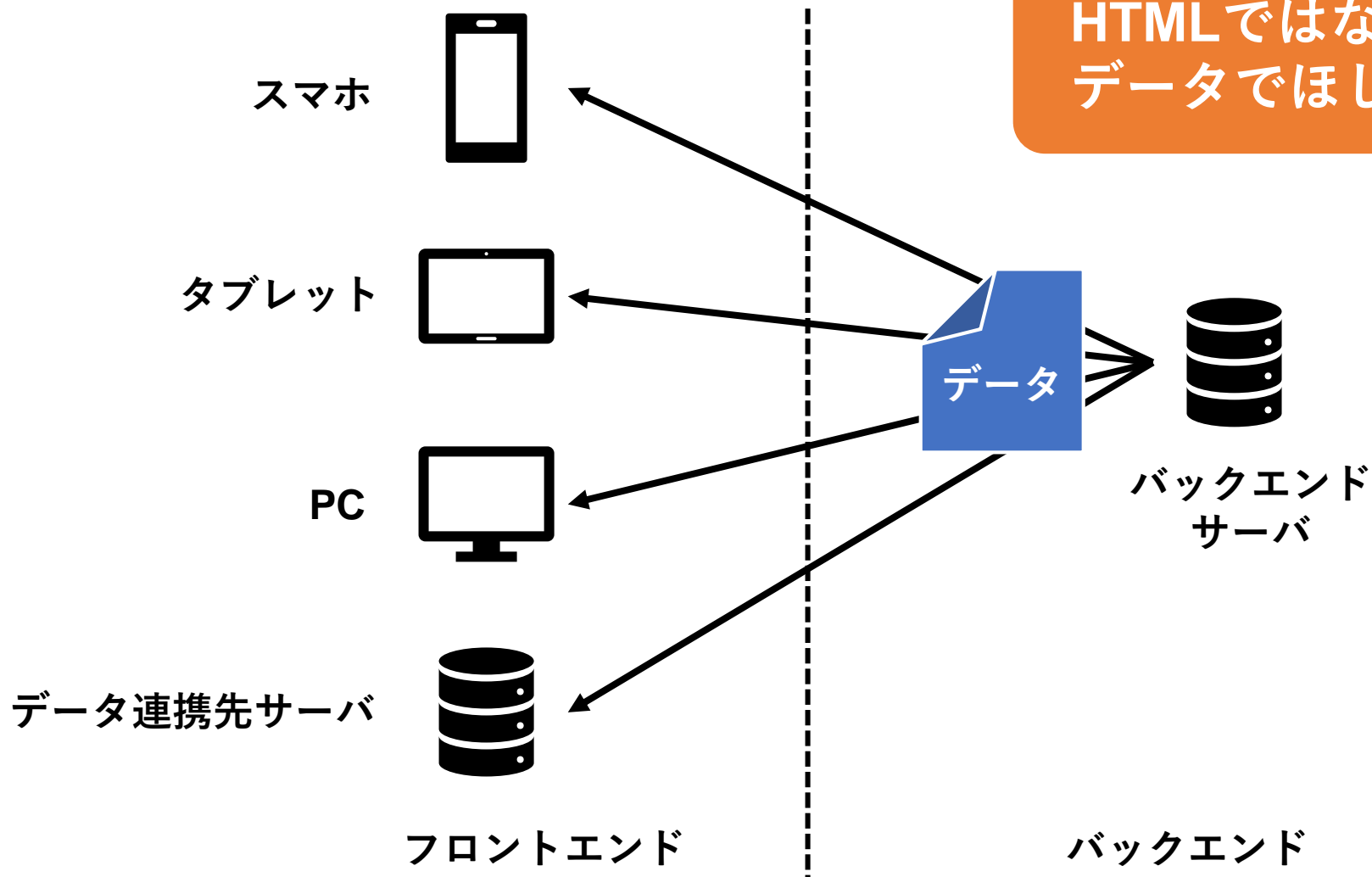
現代：クライアント側の多様化



REST APIとは

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

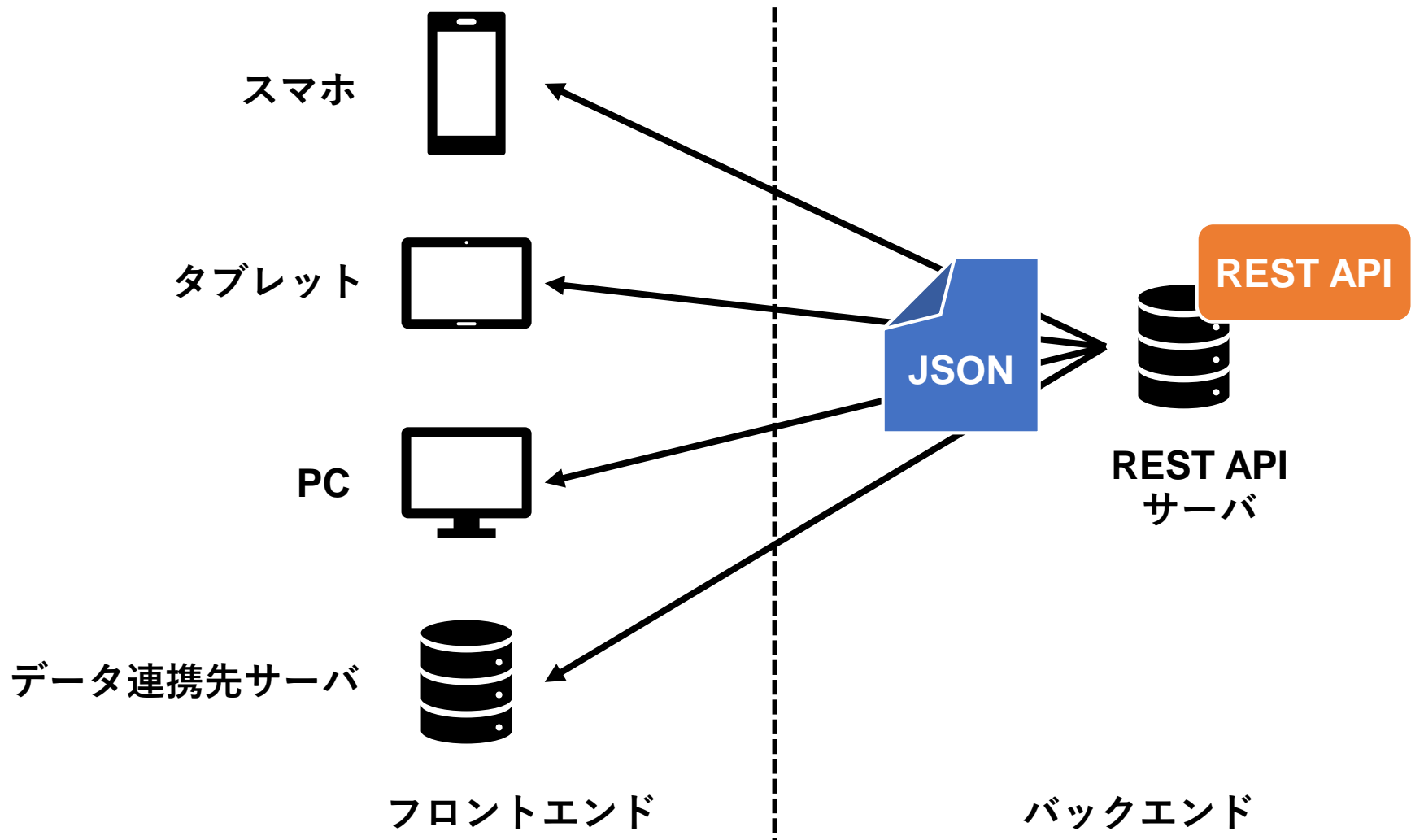
現代：クライアント側の多様化



REST APIとは

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

現代：クライアント側の多様化



REST APIとは

- クライアントの多様化に対応・バックエンドとフロントエンドの分離

REST APIの特徴

1. ユニークな識別子（URL）でリソースを特定できること（URLとリソース間が紐付いている）
2. リクエストを受け取った際に、特定の形式（JSONやXML）でレスポンスを返すこと
3. インターフェース（HTTPの利用）が統一されていること

REST APIのメリット

- クライアント・サーバ間はリソースとリソース操作に必要な情報だけをやり取りする
- クライアントとサーバの両者を疎結合化できる（テストが容易になる）

REST APIサーバ：

特定のURLでリクエストを送ると、データを返してくれる。
⇒Spring MVCがREST APIをサポート。

Core Spring研修のメモをまとめました

- GitHub Pages上で公開

Core Spring Memo



[Home](#)

[このページについて](#)

[Springメモ](#)

[ハンズオン](#)

[GitHub](#)



Core Spring Memo

Spring Framework 5.x / Spring Boot 2.x

[Springメモへ →](#)

Dependency Injection(DI)

依存性の注入

Aspect-Oriented
Programming(AOP)

割り込み処理

Bean LifeCycle

Beanインスタンスのライフサイクル

<https://imamachi-n.github.io/core-spring-memo/>

Core Spring研修のメモをまとめました

- GitHub Pages上で公開

Core Spring Memo

[Home](#)[このページについて](#)[Springメモ](#)[ハンズオン](#)[GitHub](#)[Aspect oriented programming](#)[JDBC, Transactions, JPA, Spring Data](#)[Spring Boot](#)[Spring MVC and the Web Layer](#)[Security](#)[REST](#)[RESTとは?](#)[RESTfulの意味](#)[RESTfulであるメリット](#)[Spring REST](#)[RESTはステートレス \(stateless\)](#)[RESTで使われるHTTPメソッド](#)[REST APIにおけるAuthentication](#)[HTTP GETメソッド: リソースのフェッチ](#)[HTTPメソッドに基づくRequest Mapping](#)[メッセージコンバータ](#)[REST API用のControllerクラスの設定](#)[レスポンスヘッダーを独自に記述する](#)[HTTP PUTメソッド: リソースのアップデート](#)[任意のステータスコードをレスポンスとして返す](#)[受け取ったリクエストデータをメッセージコンバータを用いてJavaオブジェクトに格納する](#)[HTTP POSTメソッド: 新しいリソースを作る](#)

REST API用のControllerクラスの設定

今までのやり方だと、`@Controller` アノテーションをつけたControllerクラスにレスポンスボディを返すメソッド (`@ResponseBody`) を設定していた。

```
@Controller
public class OrderController {

    @GetMapping("/orders/{id}")
    @ResponseBody
    public Order getOrder(@PathVariable long id) {
        return orderService.findOrderById(id);
    }
}
```

実は、Spring側でREST API用のControllerクラスを定義するためのアノテーションが用意されている。それが `@RestController` アノテーションである。

@RestController

```
@RestController
public class OrderController {

    @GetMapping("/orders/{id}")
    public Order getOrder(@PathVariable long id) {
        return orderService.findOrderById(id);
    }
}
```

`@RestController` アノテーションをつけると、すべてのメソッドに `@ResponseBody` アノテーションをつ

Core Spring研修のメモをまとめました

- GitHub Pages上で公開

Core Spring Memo

HomeこのページについてSpringメモハンズオンGitHub

Contents

- はじめに
- Spring Container, Dependency Injection
- Aspect oriented programming
- JDBC, Transactions, JPA, Spring Data
- Spring Boot
- Spring MVC and the Web Layer
 - Spring MVCについて
 - DispatcherServletとは
 - web application contextとDispatcherServlet
 - Controllerクラスの定義
 - @RequestMappingとは
 - Requestパラメータの取得
 - URLの一部を引数に取る
 - @RequestParamと@PathVariableの違い
 - @RequestParamと@PathVariableの事例
 - Viewとしてサポートしているもの
 - ViewResolver
 - Spring BootでSpring MVCを使う場合
 - @EnableWebMvc
 - 静的ファイルの扱い（画像データ、CSSファイルなどなど）
 - [WIP] 積み残し課題(重要度低)
- Security

DispatcherServletとは

SpringにおけるDispatcher Servletとは、すべてのRequestを受け取り、Controllerに処理を振り分ける役割をしている（Front Controllerと呼ばれる）。

```
graph LR; Request[request (URL)] --> DispatcherServlet[DispatcherServlet (Front Controller)]; DispatcherServlet -- "Dispatch request" --> Controller1[Controller]; DispatcherServlet -- "Dispatch request" --> Controller2[Controller]; DispatcherServlet -- "Dispatch request" --> Controller3[Controller];
```

Dispatcher Servletクラスは、受け取ったRequestをもとに各Controller（ビジネスロジック）に処理を割り振り、Modelを受け取る。受け取ったModelをもとに、View（JSPやThymeleafなど）をサーバーサイドでレンダリングし、Responseとして返す。

このように、SpringではWeb Layerをできるだけ薄くし、関心事（ビジネスロジックとViewのレンダリング）の分離を行っている。

```
graph LR; Request[request (URL)] --> DispatcherServlet[DispatcherServlet]; DispatcherServlet -- "Dispatch request" --> Controller[Controller]; Controller -- Model --> DispatcherServlet; DispatcherServlet -- "render" --> View[View]; View -- "論理ビュー名を渡す" --> DispatcherServlet; DispatcherServlet -- response --> Response[response];
```

Core Spring研修のメモをまとめました

- GitHub Pages上で公開

Core Spring Memo



Home

このページについて

Springメモ

ハンズオン

GitHub

参考になるハンズオン集

GitHubで公開されているSpring Bootのハンズオンを集めてみました。

Spring 5 & Spring Boot 2ハンズオン - 課題：顧客管理アプリ

Building a Full Stack Polls app similar to twitter polls with Spring Boot, Spring Security, JWT, React and Ant Design

AtSea Shop Demonstration Application - A sample app that uses a Java Spring Boot backend connected to a database to display a fictitious art shop with a React front-end

Edit this page on GitHub

Last Updated: 7/22/2018 6:30:50 PM

<https://imamachi-n.github.io/core-spring-memo/>

研修を通じて得たもの

**(1) Springの使い方を学ぶだけにとどまらず、
Spring内部で何が行われているのか、より深い仕組みについて学べた。**

⇒複雑な処理が隠蔽させているので、あまり理解できていなくても使えてしまう。
しかし、想定外のエラーに直面したとき、内部の詳しい仕組みを理解していることは問題解決に重要。

(2) エンジニアとしての姿勢

⇒公式ドキュメントを読み込む（ブログ記事を鵜呑みにしない）。
⇒フレームワークのソースコードを読解・理解する。

**(3) 周りの受講者（30～40代）、
アーキテクト・プロジェクトマネージャ含む。**

⇒受講者の質問の質が高く、Q&A自体が勉強になった。
⇒例えば、スレッドセーフな設計・パフォーマンスの劣化につながるかなどシステム設計に対する考え方。

書籍やe-Learningでは得られない、貴重な体験・学びがあった！

研修を通じて得たもの

(4) Spring Framework自体が学びの多いフレームワーク

- Spring Data JPAを使うと、DBテーブルのリレーションが重要になってくるので、DBの設計が気になります。
- インスタンス（Bean）の生成と破棄（ライフサイクル）について意識するようになった。
- Spring Securityを使うと、セキュリティについて学びがある（平文のパスワードをDBに保存するのではなく、ハッシュ化した値を格納するなど）。

- 世の中の動き、自分の立ち位置、行動の助けになるもの
- 既存のシステムのどこが悪いのか、どこを気をつけないといけないのか？
- そのために、**Spring**がどのように活用できるか。
- 他の言語のフレームワークを探す上での指針を示す。

なぜSpring frameworkなのか

自分なりの答え

(1) Go言語 (Golang)

長所

- ・ Google発のプログラミング言語
- ・ 文法に厳しく、複雑なコードを書かせない仕様（継承が使えないなど）
- ・ 言語としてのパフォーマンスが高い（C言語に匹敵する速度）
- ・ 並列処理が比較的簡単に書ける
- ・ Webアプリケーション開発に必要なライブラリを標準搭載

⇒ 言語レベルでのサポートが強力！

⇒ ユーザのアクセスが多いECサイトを始めとするBtoCビジネス向け

短所

- ・ Web系フレームワークとして決定版となるものがない...
- ・ Webや書籍などで得られる情報が少ない...
- ・ 企業での採用事例に乏しい...
- ・ REST APIサーバ作成用で、View（JSPなど）のサポートに乏しい

なぜSpring frameworkなのか

疑問

新規にWebアプリケーションを開発するとして、
バックエンドを開発する上で最良の言語は何か？

自分なりの答え

(1) Go言語 (Golang)

長所

処理上、ボトルネックになる部分で局所的に採用するのはアリ
⇒ただし、汎用性があるかと聞かれると...??

・ 言語としてのパフォーマンスが同じ (C言語に匹敵する速さ)

・ 並列処理が比較的簡単に書ける

・ Webアプリケーション開発に必要なライブラリを標準搭載

⇒ 言語レベルでのサポートが強力！

⇒ ユーザのアクセスが多いECサイトを始めとするBtoCビジネス向け

短所

・ Web系フレームワークとして決定版となるものがない...

・ Webや書籍などで得られる情報が少ない...

・ 企業での採用事例に乏しい...

・ REST APIサーバ作成用で、View (JSPなど) のサポートに乏しい

なぜSpring frameworkなのか

自分なりの答え

(2) Java言語 (Spring Framework)

長所

- ・ 15年以上の歴史を持つ成熟したフレームワーク
- ⇒ StrutsやSeasarといった主要なフレームワークがすべてEOLとなり、JavaのWeb系フレームワークは事実上Springに一本化されつつある
- ・ 依存性の注入 (DI) ・ アスペクト指向プログラミング (AOP) の仕組みを導入したモダンで一般的なフレームワーク (後述します)
- ⇒ モジュール間を疎結合化する仕組み (現代では常識に)
- ⇒ テスト容易性、保守性、再利用性の高いコードを書く上で重要なコア技術
- ・ オープンソースだが、Pivotal社 (Dellの孫会社) がバックについている
- ・ エンタープライズ向けで、企業での採用事例が豊富
- ・ Webや書籍などで得られる情報も多い

短所

- ・ 学習コストがかなり高い...

Spring FrameworkでのWeb開発が有力という結論に。

なぜSpring frameworkなのか

自分なりの答え

(2) Java言語 (Spring Framework)

長所

- ・ 15年以上の歴史を持つ成熟したフレームワーク
- ⇒ StrutsやSeasarといった主要なフレームワークがすべてEOLとなり、JavaのWeb系フレームワークは事実上Springに一本化されつつある
- ・ 依存性の注入 (DI) ・ アスペクト指向プログラミング (AOP) の仕組み
- ①ソースコードのメンテナンスのしやすさ
- ②汎用的な技術を学べるかどうか (フレームワーク固有でない技術)
- ③将来性 (5, 10年先でも生き残るフレームワークかどうか) 技術
- ⇒ Spring FrameworkでのWeb開発が有力という結論に。
- ・ エンタープライズ向けで、企業での採用事例が豊富
- ・ Webや書籍などで得られる情報も多い

短所

- ・ 学習コストがかなり高い...

Spring frameworkの良さ



ソースコードの
保守性



技術の汎用性



将来性

- ①ソースコードのメンテナンスのしやすさ
 - ②汎用的な技術を学べるかどうか（フレームワーク固有でない技術）
 - ③将来性（5, 10年先でも生き残るフレームワークかどうか）
- ⇒ Spring FrameworkでのWeb開発が有力という結論に。