

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ら来ている？）

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ら来ている？）

フレームワーク(特定の型)に当てはめてシステムを設計する

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、



システム開発を行う上で
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、



システム開発を行う上で
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・設計思想がその他の言語・フレームワークに広く応用されているか
- エンジニアにとって学びの多いフレームワークであるかどうか

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

- ⇒15年以上の歴史を持つフレームワーク
- ⇒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

(2) 世の中のニーズ・新しい技術に対応

- ⇒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は、ASP.NET Core (C#), Ruby on Rails (Ruby), 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
⇒DIは別言語・フレームワークを使ったWebアプリ開発において基礎となる設計思想の1つ

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
- ⇒ ID, AOP, Spring MVC, Spring Data, Spring Security, Maven, JUnitと覚える必要のあるツールがいっぱい...



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などの選択肢がある。
- ⇒ 要件・人員の技術力に応じて選択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)

- ・ほとんどが英語の授業（新しいこと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人ほどしかいないハイレベルな認定トレーナーのもと、短期集中でSpringが学べる。
- ・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
- Spring Boot
- Spring Data
- Spring Security

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

ごく一部

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。
- 中でも重要なのが、Spring Boot, Spring Data, Spring Security。



Spring Security



Spring Data



Spring AMQP



Spring Batch



Spring Boot



Spring Framework



Spring Cloud



Spring Social

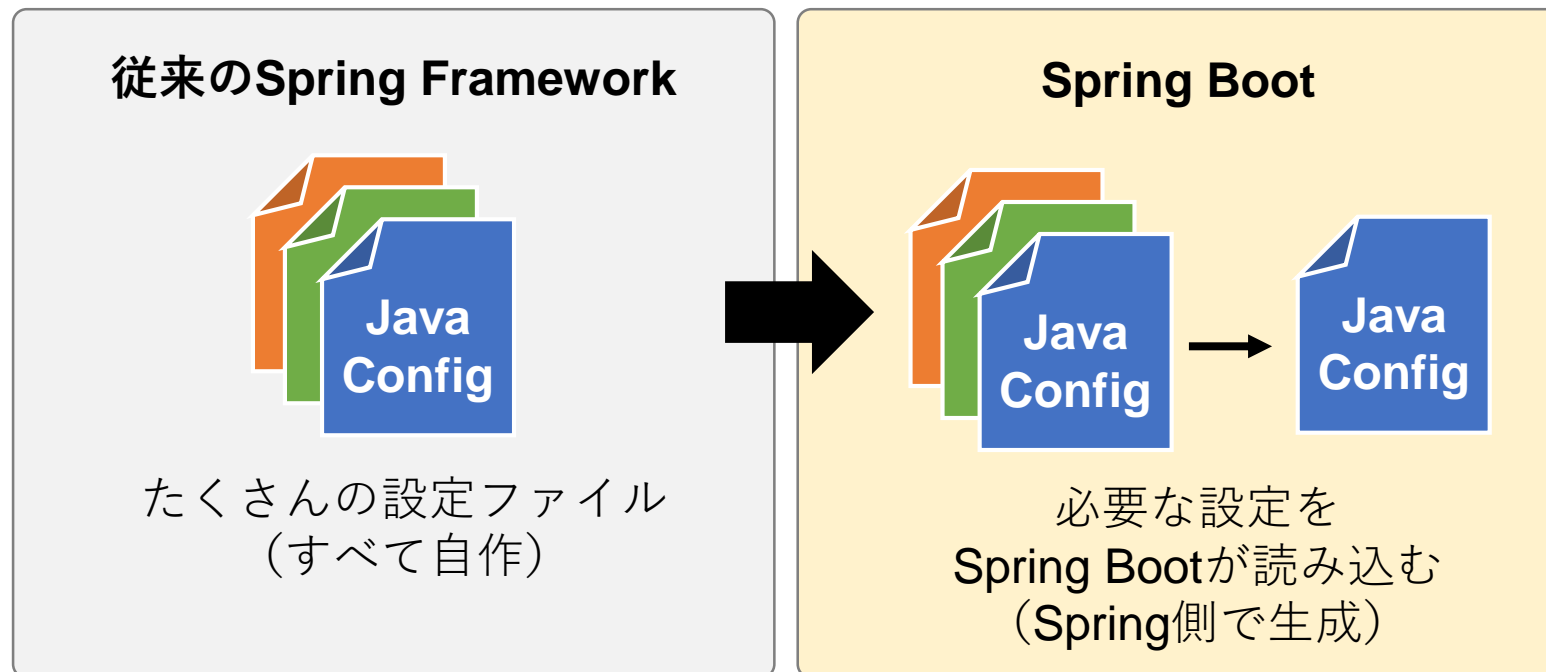


Spring Mobile

Spring Boot

- SpringによるWebアプリ開発がすぐに開始できる

- Spring frameworkで必要となるJava config（設定ファイル）のかたまり。
- Spring BootのAuto Configurationのjarファイル内で、本来は開発者側で用意する必要のあるJava Configを、事前に用意してくれている。
⇒ 設定用クラスがclasspathにあると、自動的に必要な設定を読み込んでくれる。



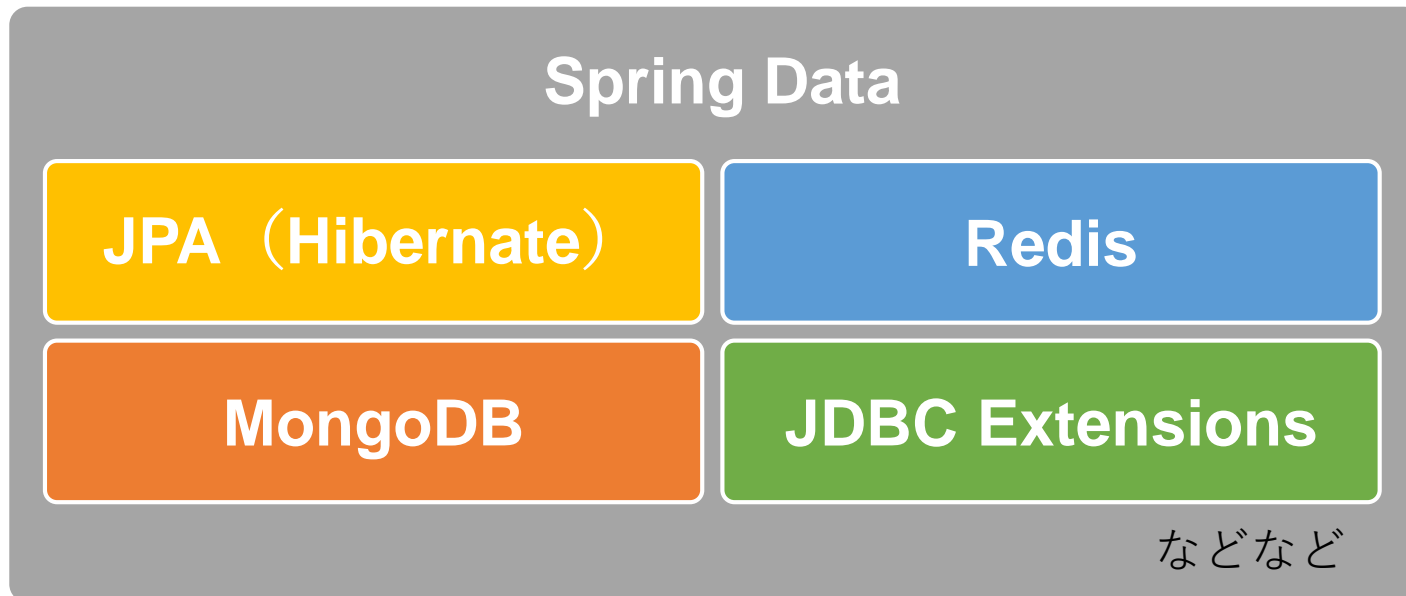
Spring Data

- データベースの違いを吸収し、より簡単にデータアクセスができるように

- データベースの違いを吸収し、同じ方法によるデータアクセスを可能にする。
⇒データベースの種類の違いさえも吸収。RDBもNoSQLも同様に扱えるように。
- ボイラープレートが減らして、可読性の高いコードへ。



Spring Data



Spring Data

- データベースの違いを吸収し、より簡単にデータアクセスができるように

- ボイラープレートが減らして、可読性の高いコードへ。

```
public List<Person> findByLastName(String lastName){
    List<Person> personList = new ArrayList<>();
    String sql = "select first_name, age from PERSON where last_name = ?";

    try(Connection conn = dataSource.getConnection();
        PerparedStatement ps = conn.prepareStatement(sql)){

        ps.setString(1, lastName);

        try(ResultSet rs = ps.executeQuery()){
            while(rs.next()){
                personList.add(new Person(rs.getString("first_name"), ...));
            }
        }
    }catch(SQLException e){
        /* エラー処理 */
    }

    return personList;
}
```

Spring Data

- データベースの違いを吸収し、より簡単にデータアクセスができるように

- ボイラープレートが減らして、可読性の高いコードへ。

```
public List<Person> findByLastName(String lastName){
    List<Person> personList = new ArrayList<>();
    String sql = "select first_name, age from PERSON where last_name = ?";

    try(Connection conn = dataSource.getConnection();
        PerparedStatement ps = conn.prepareStatement(sql)){

        ps.setString(1, lastName);

        try(ResultSet rs = ps.executeQuery()){
            while(rs.next()){
                personList.add(new Person(rs.getString("first_name"), ...));
            }
        }
    }catch(SQLException e){
        /* エラー処理 */
    }

    return personList;
}
```

その他のコードはいつも同じ内容になる
⇒ボイラープレート部分を削減したい！

Spring Data

- データベースの違いを吸収し、より簡単にデータアクセスができるように

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    public <S extends T> save(S entity);
    public <S extends T> Iterable<S> save(Iterable<S> entities);

    public T findOne(ID id);
    public Iterable<T> findAll();

    public void delete(ID id);
    public void delete(T entity);
    public void deleteAll();

}
```

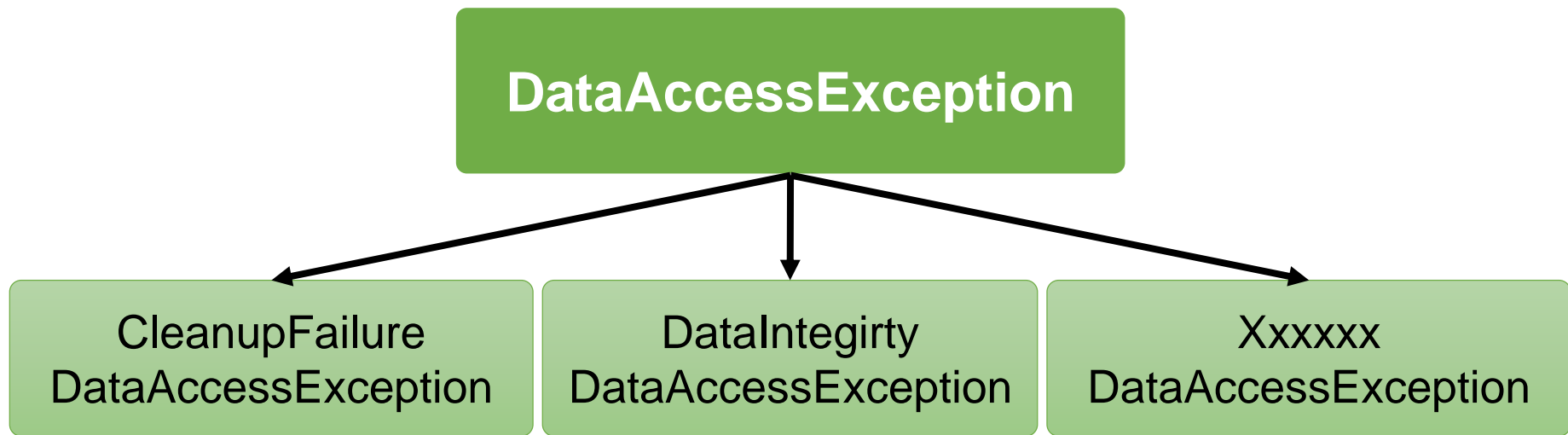
```
public interface PersonRepository extends CrudRepository<Car, Long> {
    List<Car> findByLastName(String lastName);
}
```

CrudRepositoryインターフェースを実装したクラスから、
DBへの参照・登録・削除がメソッドを呼び出すだけで可能となる

Spring Data

- 例外処理も簡単に

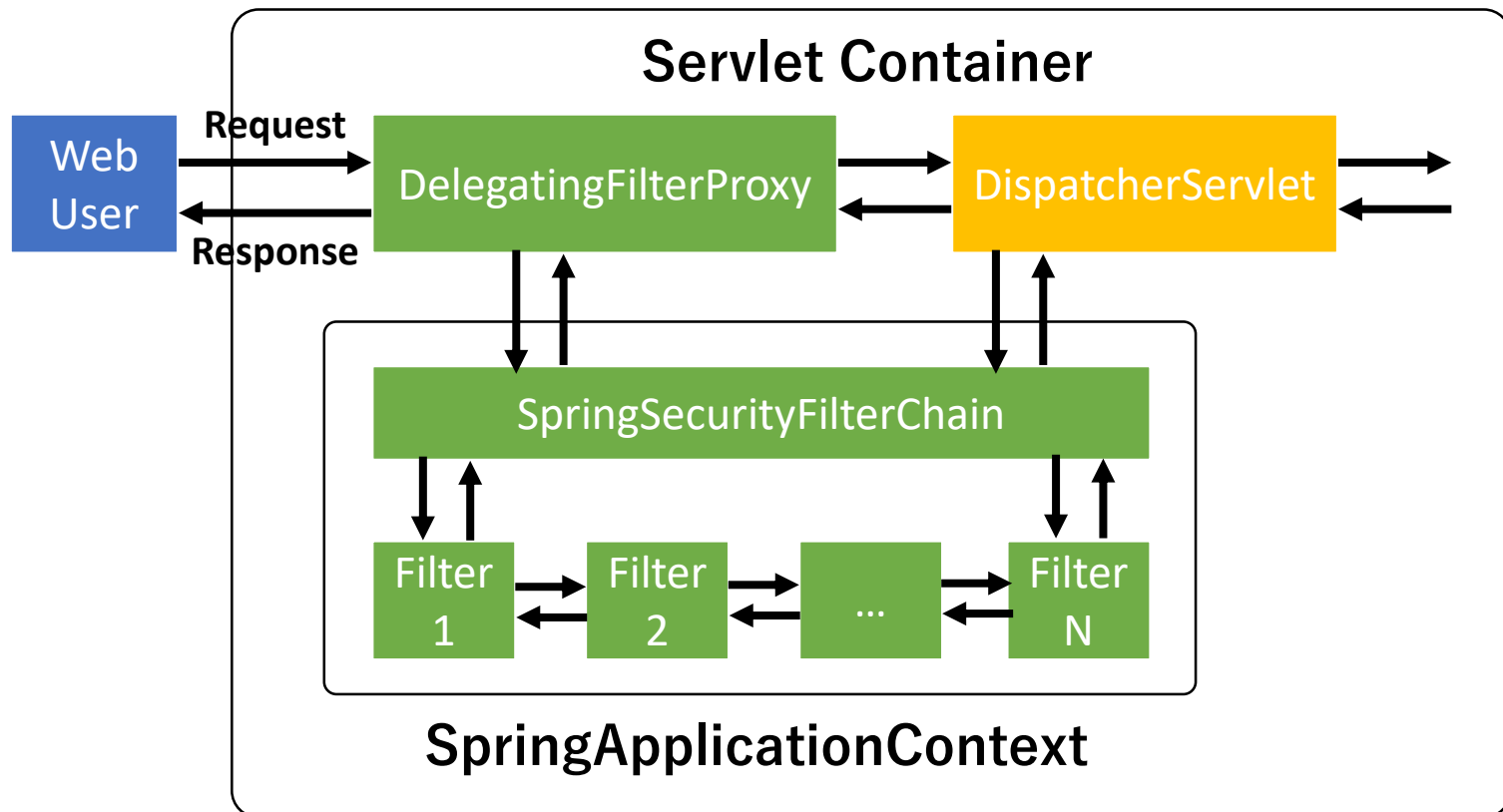
- 例外発生時は、SQLExceptionをラップしたDataAccessExceptionが呼ばれる。
- SQLExceptionよりも詳細なsub-Exceptionが吐かれる。
⇒DBの違いによるエラーコードの差異はSpring Dataが吸収。



Spring Security

- アプリケーション側でのセキュリティ設定を可能にする

- Servletクラス（DispatcherServlet）にRequestを渡す前にセキュリティフィルタリング（認証・認可などの処理）を行う。



Spring Security

-アプリケーション側でのセキュリティ設定を可能にする

- Spring Securityを導入した時点で、すべてのファイルへのアクセスに認可が必要となる。
- 以下の例のように、どのユーザにどのディレクトリへのアクセス権限を付与するか設定する。

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .mvcMatchers("/signup", "/about").permitAll()  
            .mvcMatchers("/accounts/edit*").hasRole("ADMIN")  
            .mvcMatchers("/accounts/**").hasAnyRole("USER", "ADMIN")  
            .anyRequest().authenticated();  
}
```

- BASIC認証（例. DBにユーザ・パスワードを保存）の場合、パスワードをハッシュ化するためのメソッドも用意されている。

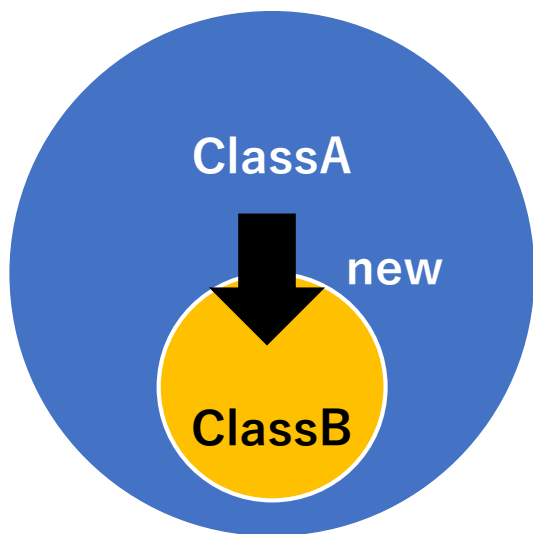
```
auth  
    .inMemoryAuthentication().passwordEncoder(new BCryptPasswordEncoder(12));
```

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

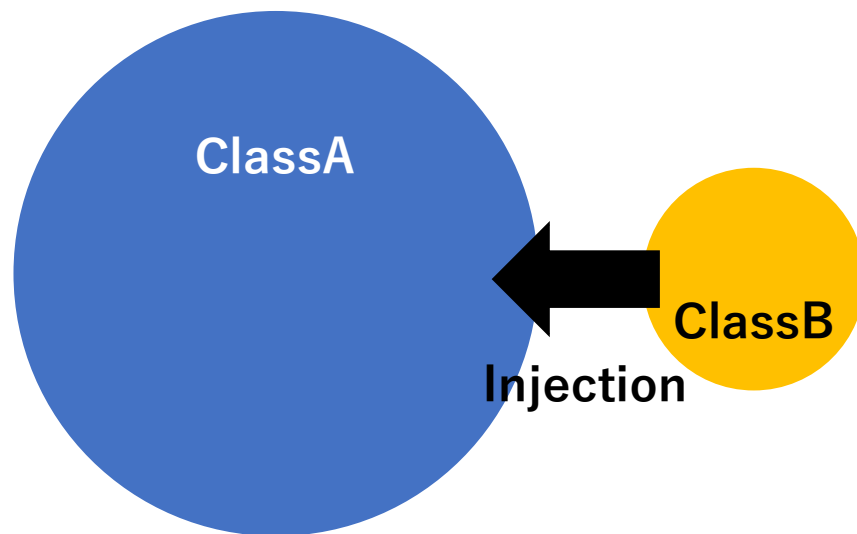
- モジュールを疎結合に

- 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

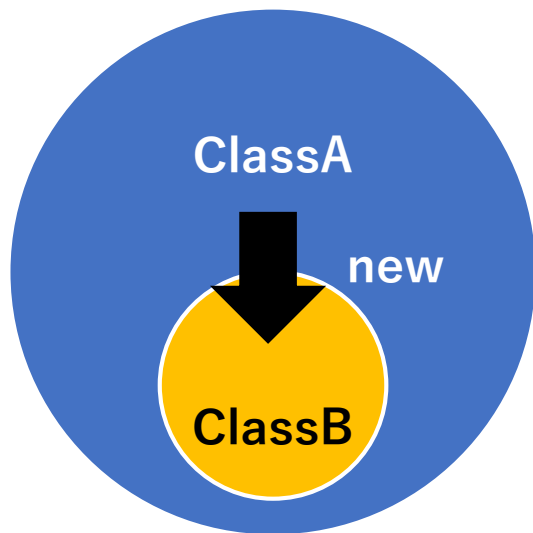


必要なインスタンス（依存性）を外から代入すること。

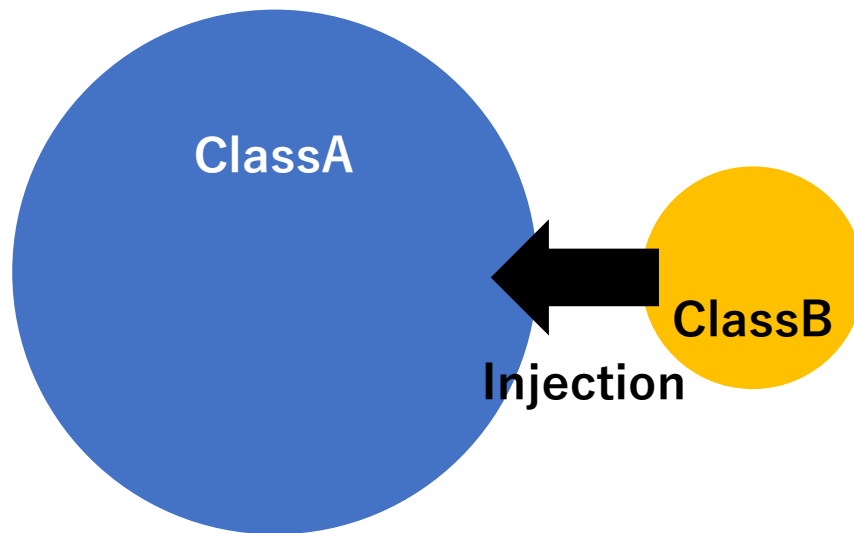
依存オブジェクトの注入（DI: Dependency Injection）とは

- モジュールを疎結合に

従来のClass設計



DIを使ったClass設計

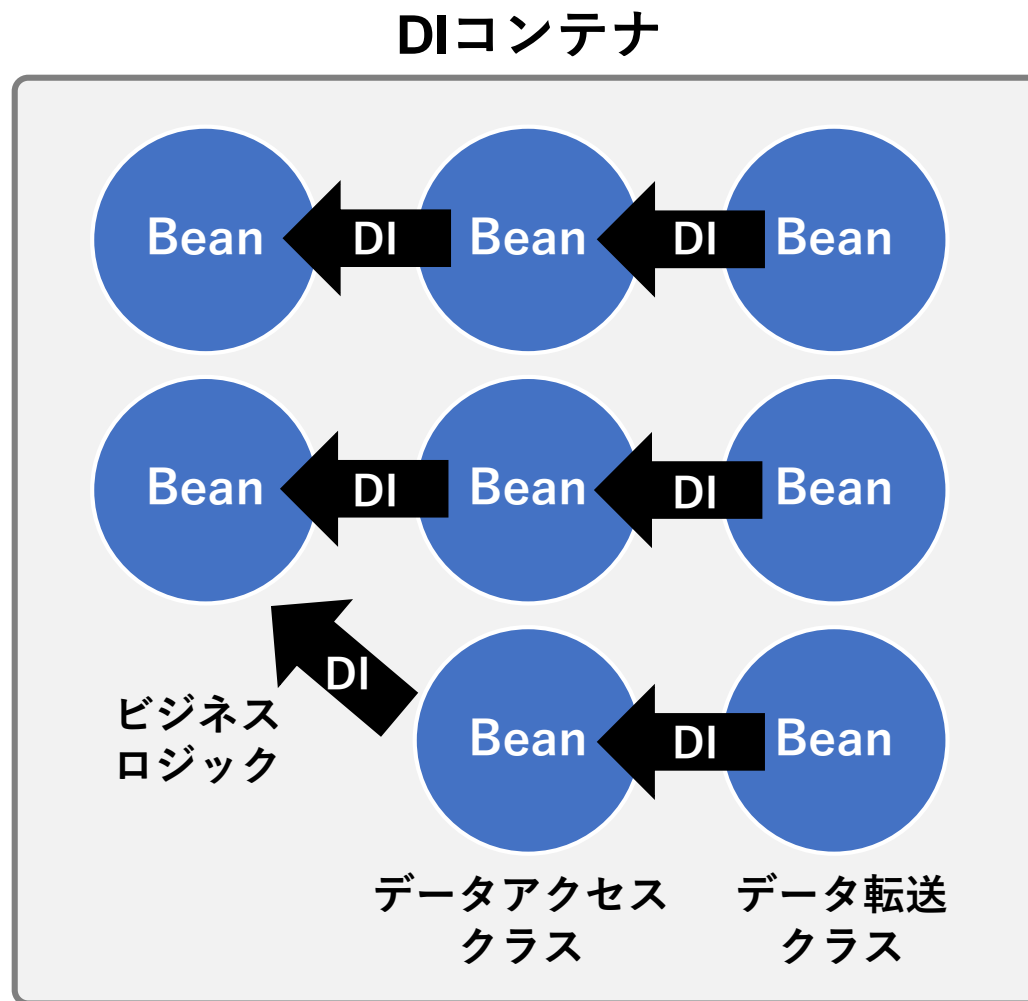


- ClassAの動作は、ClassBに依存している。
- ClassBが完成しないと、ClassAを動かすことができない。

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



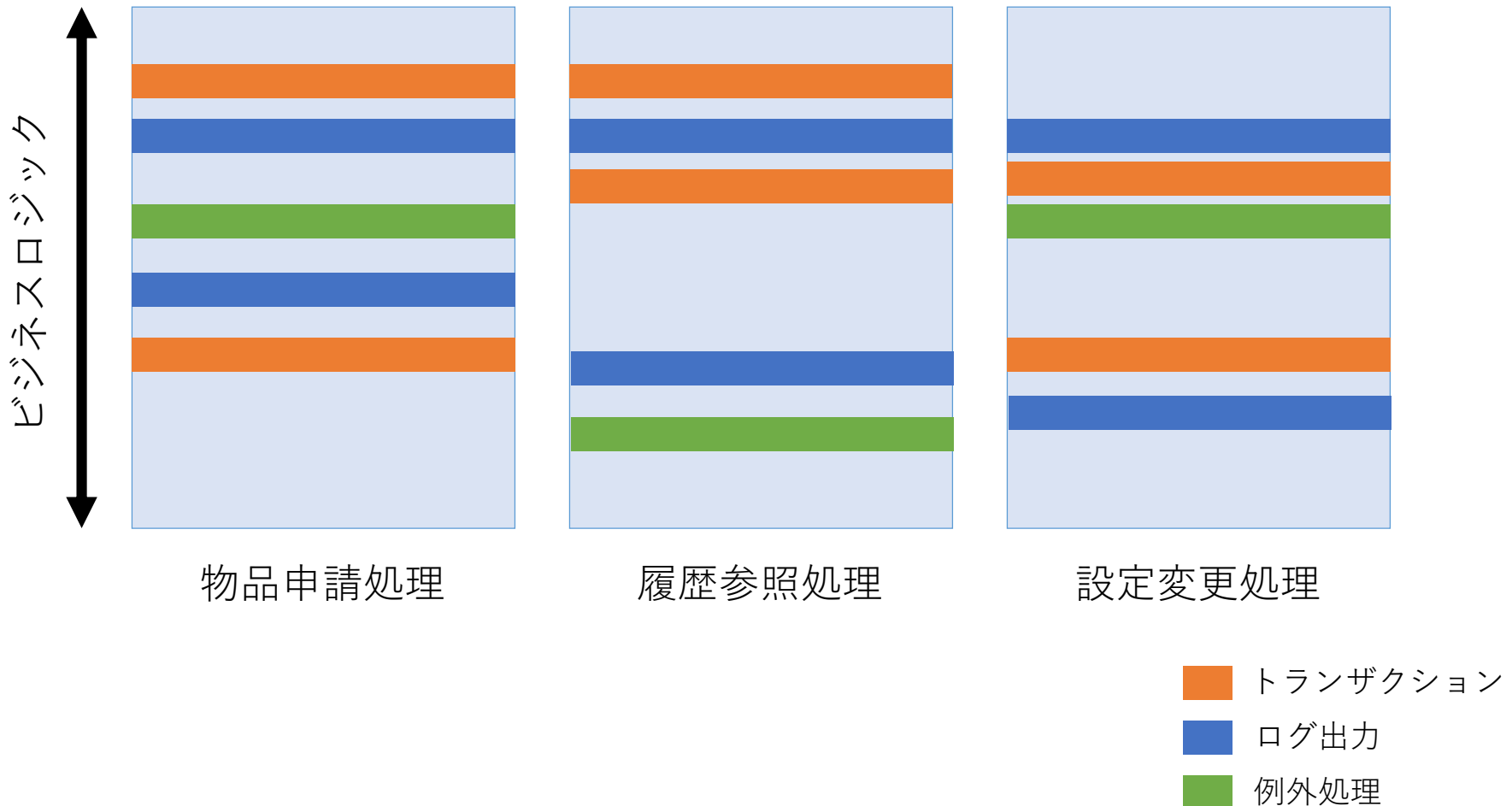
Springが内部で持っているインスタンス (Bean) の格納庫

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）

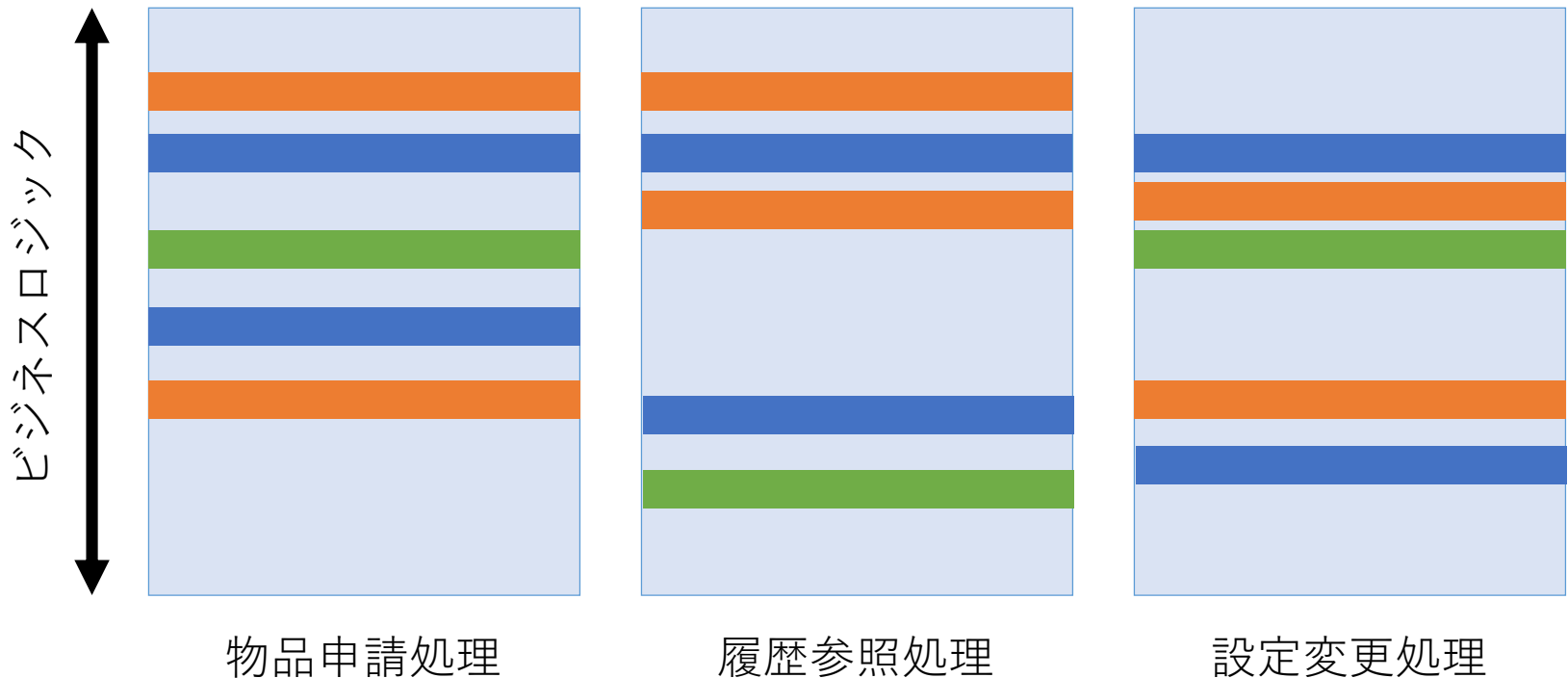


アスペクト指向プログラミング（AOP）とは

- 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）

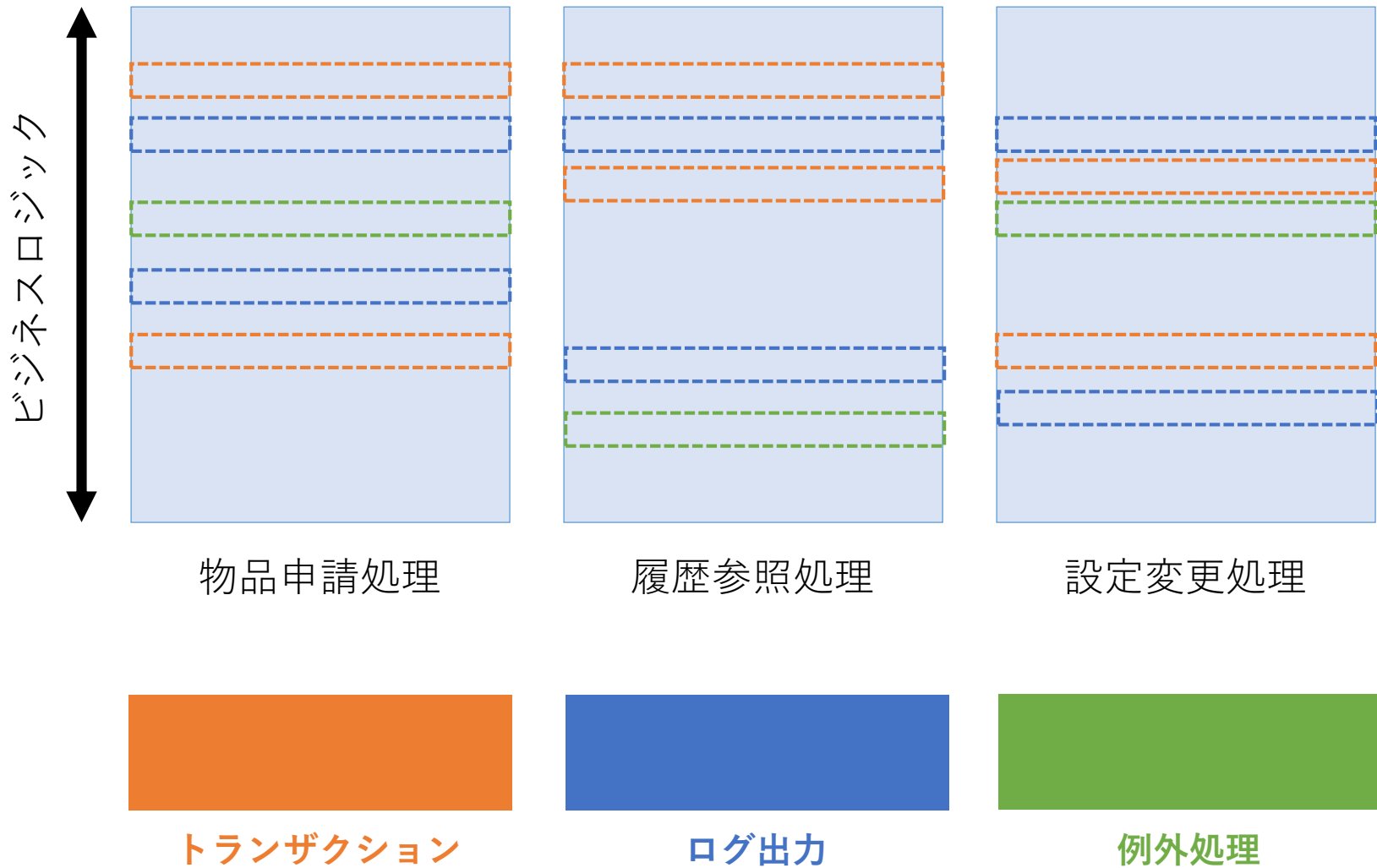


ビジネスロジックに別の処理が入り込んでいる。
⇒結果として、処理内容がわかりにくくなる...

- トランザクション
- ログ出力
- 例外処理

アスペクト指向プログラミング（AOP）とは

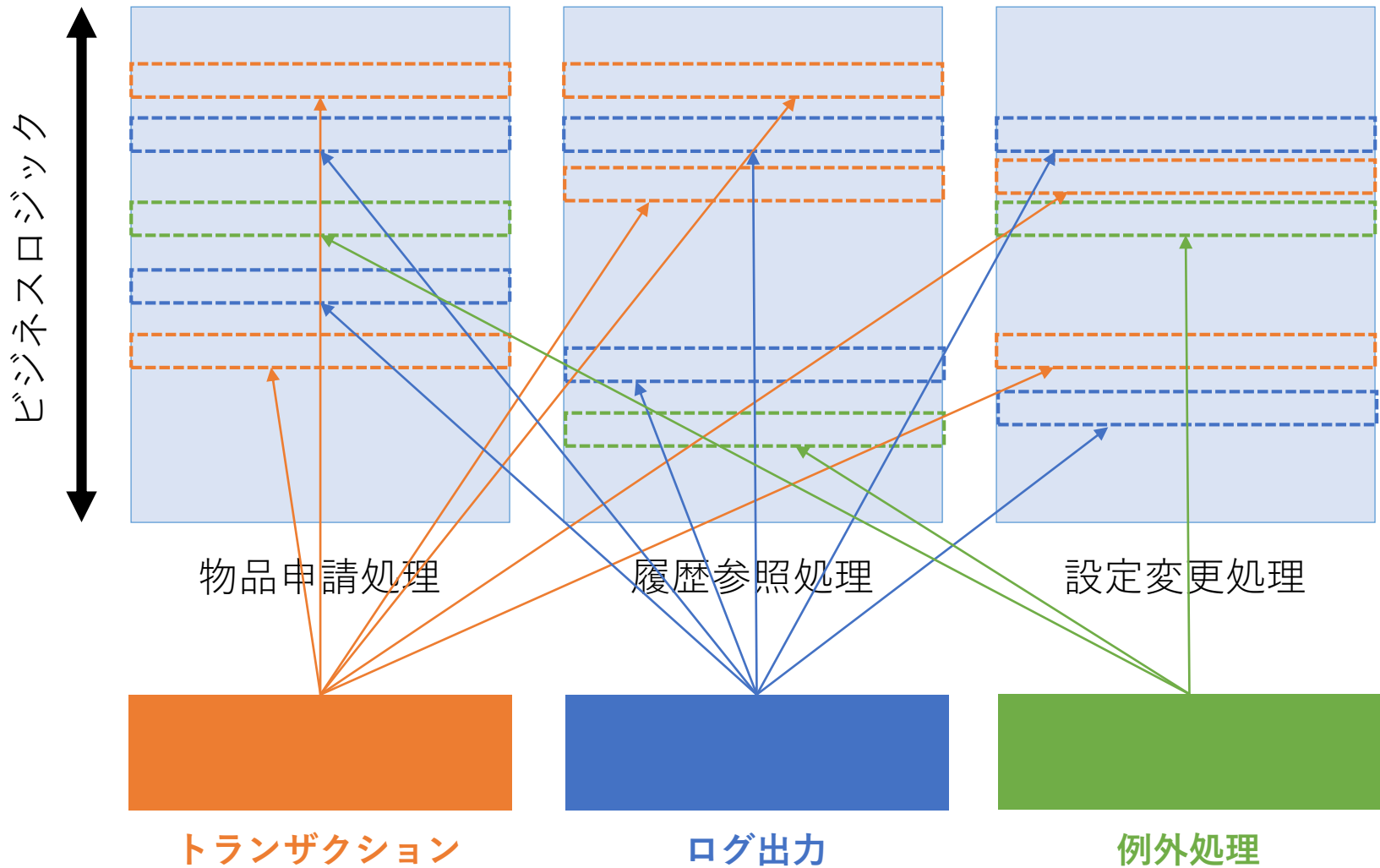
- 横断的関心事（ログ/セキュリティ/例外処理など）をビジネスロジックから分離



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。

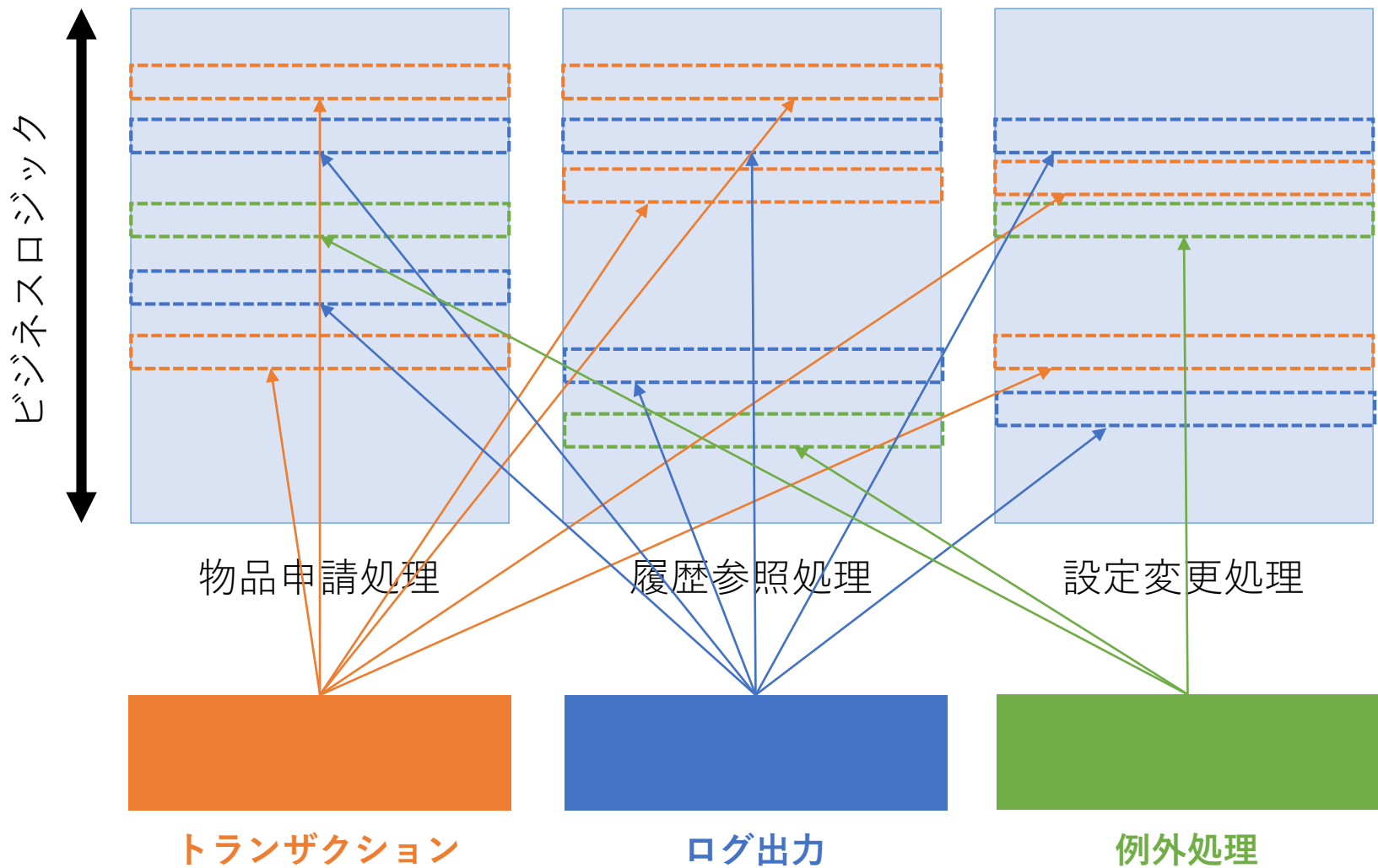
アスペクト指向プログラミング (AOP) とは

- 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voloンツツツツツ (仮)

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

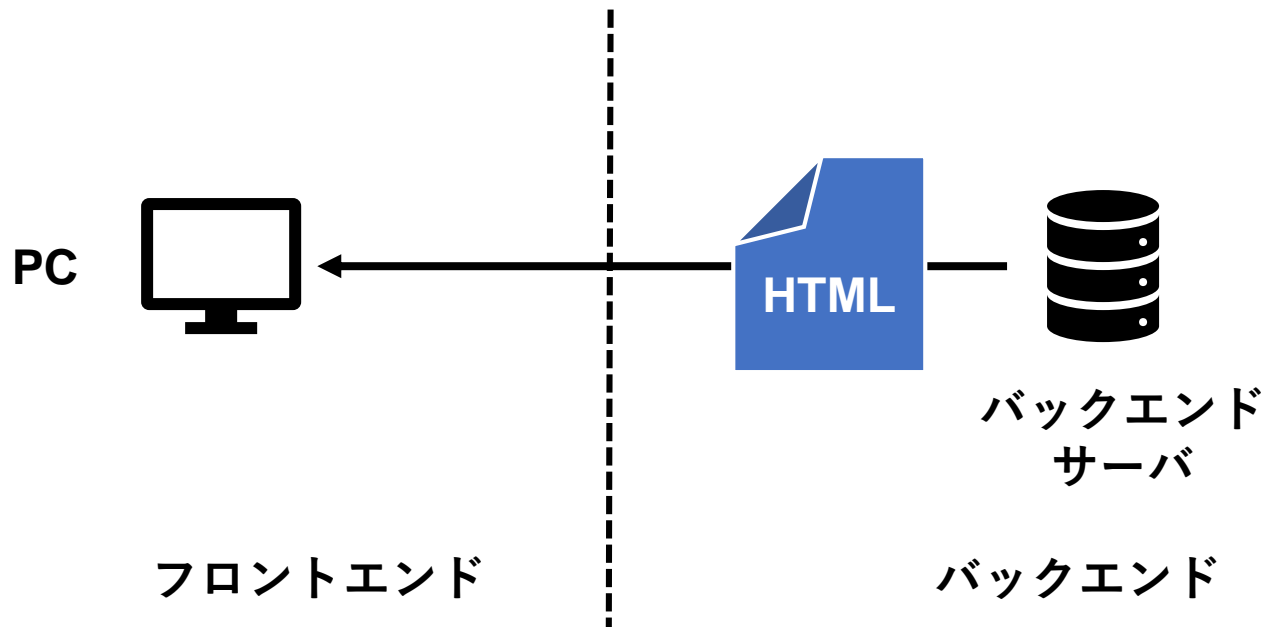
割り込みの処理対象となるメソッド

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

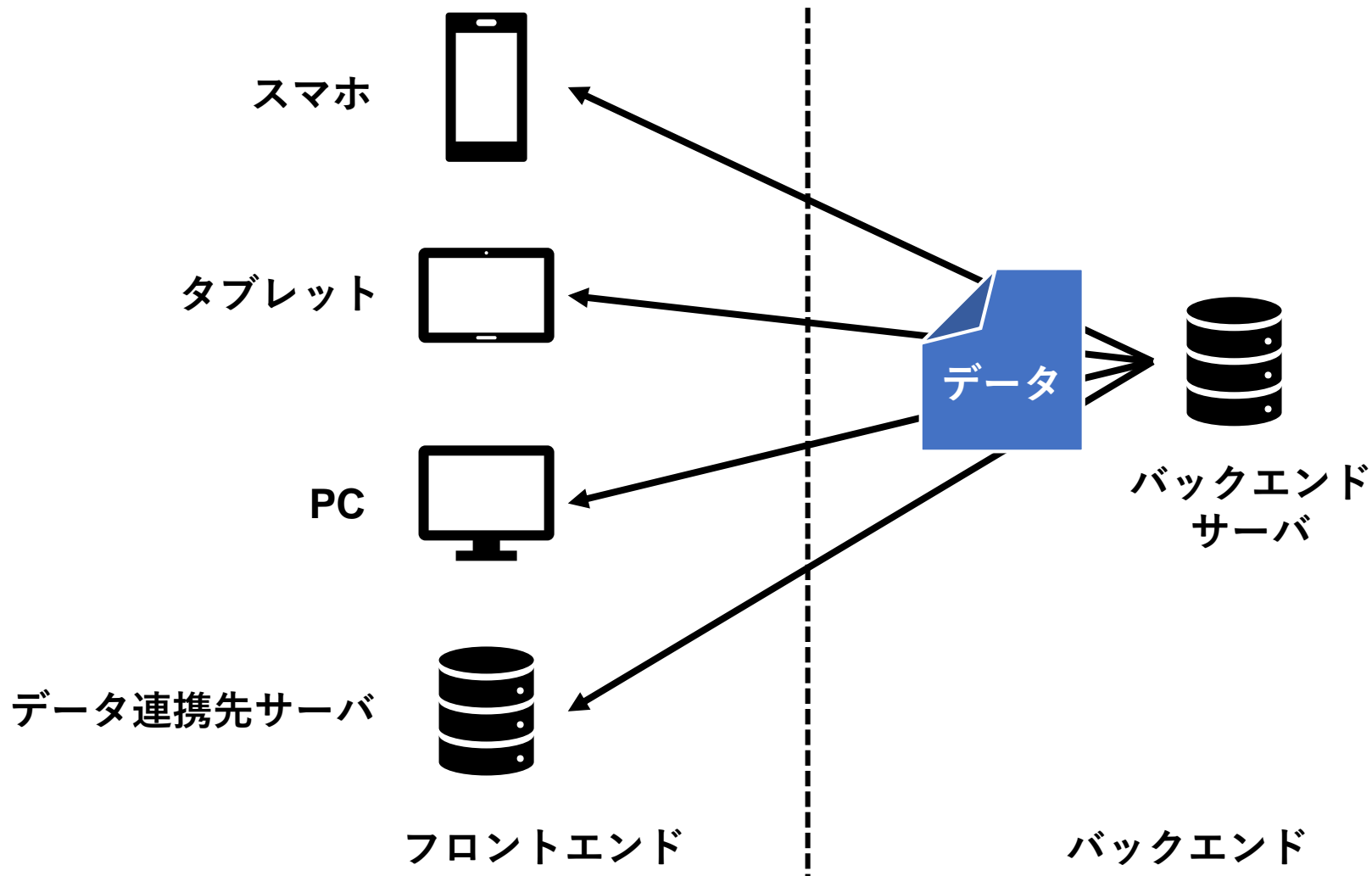


サーバ側でHTMLをレンダリングしてPCへ送る

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

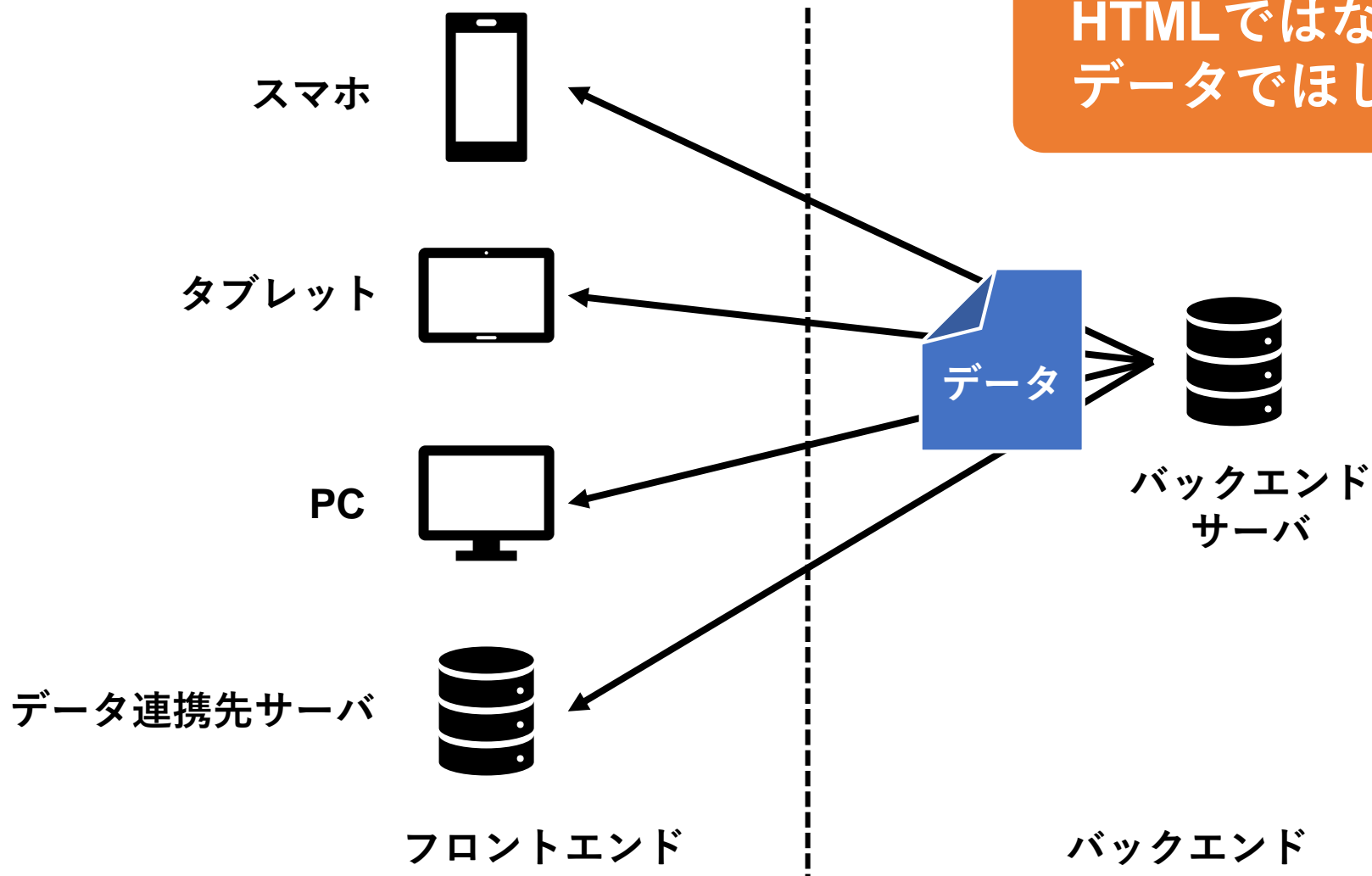
現代：クライアント側の多様化



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

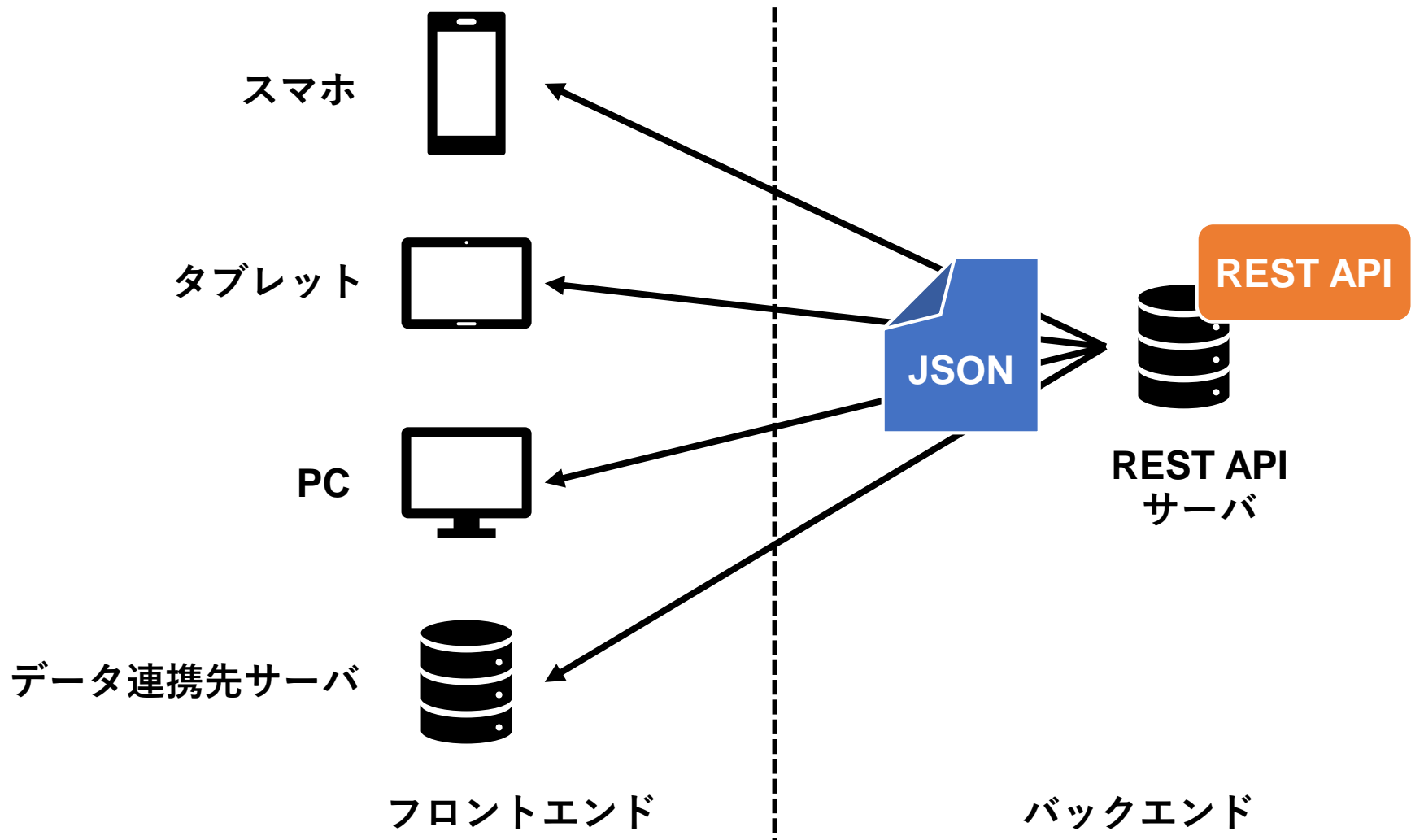
現代：クライアント側の多様化



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

- 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

- 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

- 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

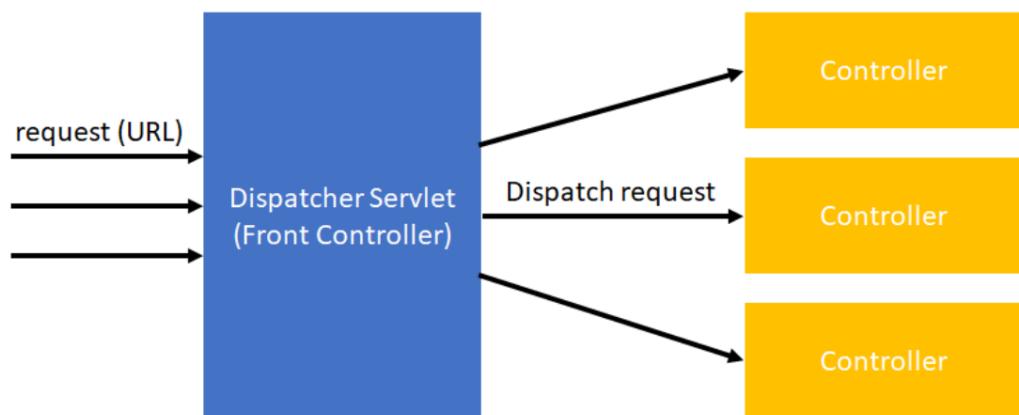
静的ファイルの扱い（画像データ、
CSSファイルなどなど）

[WIP] 積み残し課題(重要度低)

Security

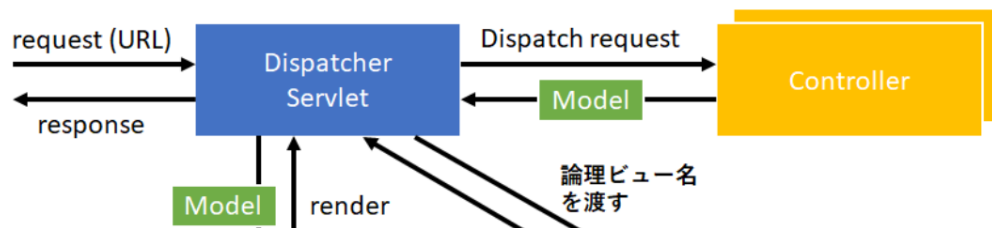
DispatcherServletとは

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とも呼ばれる）。



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。



Core Spring研修のメモ

- 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/>

Spring Boot + Reactを利用した基本的なWebアプリ

- GitHub上で公開しているもの

Imamachi-n / [spring-boot2-react-material-ui-carlist-app](#) Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Hands-On Full Stack Development with Spring Boot 2.0 and React Edit

spring-boot spring-security jwt react material-ui docker mariadb yarn maven spring-data-jpa rest-api Manage topics

Java 54.8% JavaScript 40.1% HTML 3.7% CSS 1.1% Shell 0.3%

Get Started

- Run MariaDB using Docker container (port:3307)

```
docker run -it --rm --name mariadb -e MYSQL_DATABASE=cardb_test -e MYSQL_ROOT_PASSWORD=password -p 3307:3306 -d mariadb
```
- Run Backend server (Spring Boot2 embedded Tomcat, port:8090)

```
# For MacOS, Linux
./mvnw spring-boot:run

# For Windows
mvnw.cmd spring-boot:run
```
- Run Frontend server (Nodejs + React, port:3000)

```
# Install all the dependencies listed within `package.json`
yarn install

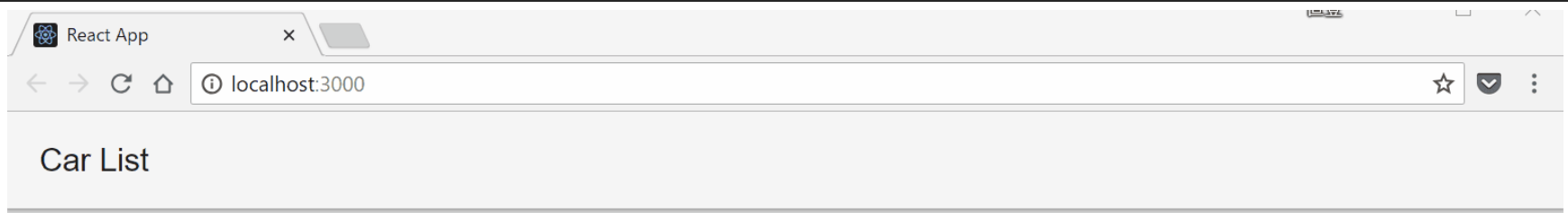
# Startup React app
yarn start
```



<https://github.com/Imamachi-n/spring-boot2-react-material-ui-carlist-app>

Spring Boot + Reactを利用した基本的なWebアプリ

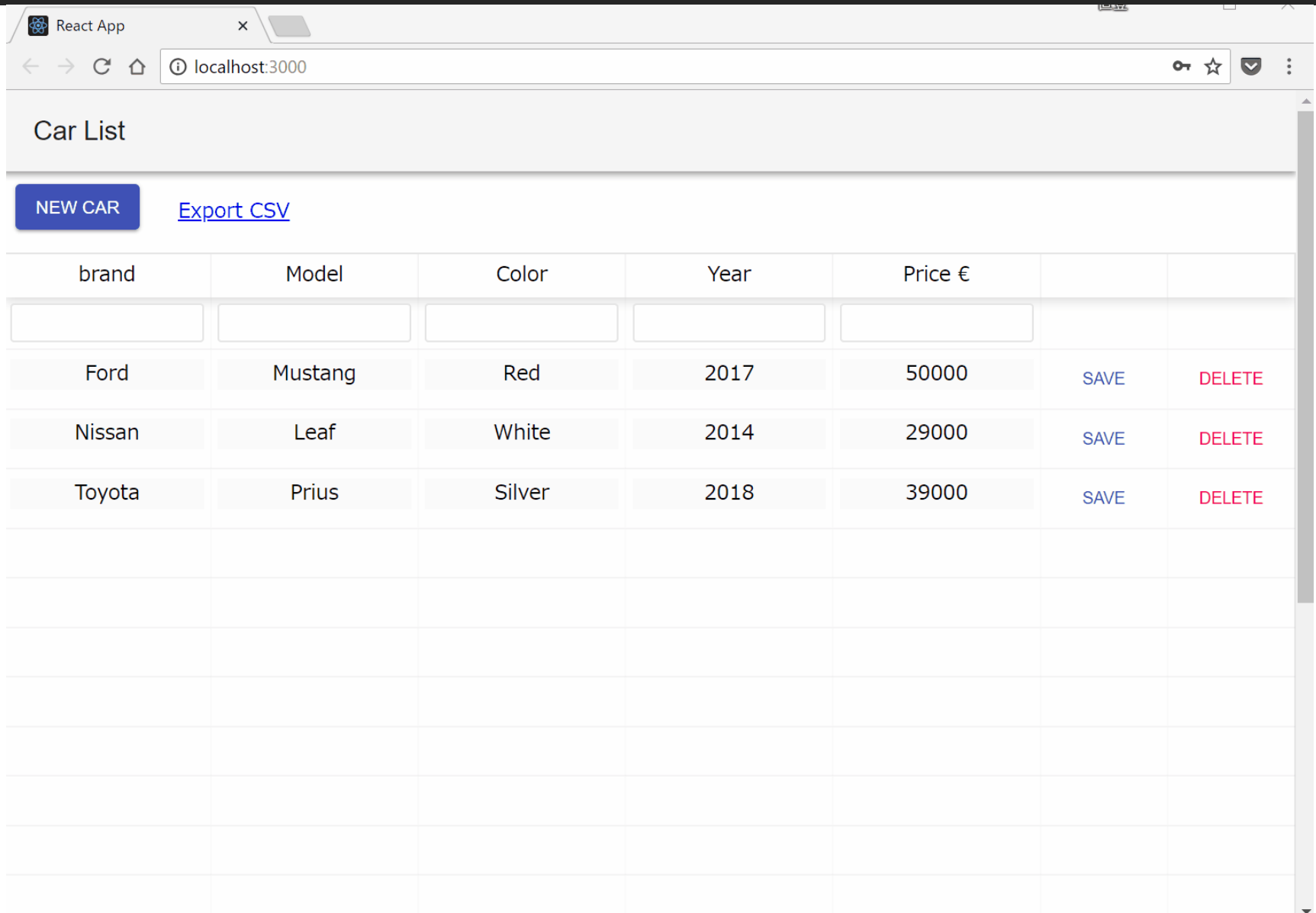
- GitHub上で公開しているもの



<https://github.com/lmamachi-n/spring-boot2-react-material-ui-carlist-app>

Spring Boot + Reactを利用した基本的なWebアプリ

- GitHub上で公開しているもの



<https://github.com/Imamachi-n/spring-boot2-react-material-ui-carlist-app>

Spring Frameworkの学習に使っているもの

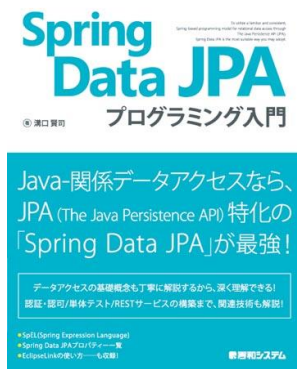
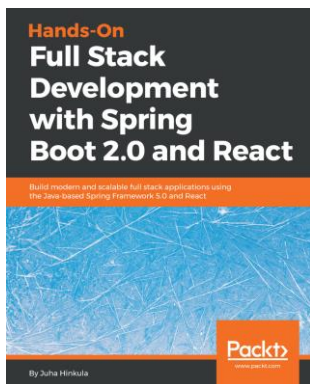
- 書籍・e-Learning・サンプルプロジェクトのソースコード

1. 書籍

- **Hands-On Full Stack Development with Spring Boot 2.0 and React**
⇒REST APIによるWebアプリ開発の全般を学べる。
⇒REST APIの実装, Spring Security (BASIC認証・パスワードのハッシュ化・JWT), React, Material-UI, JUnit, Jest, Enzyme
- **Spring Data JPA プログラミング入門**
⇒Spring Data JPAの多くの機能が紹介されており、辞書的な本。
⇒姉妹書はイマイチなので注意。

2. e-Learning (Udemy)

- **Spring Framework 5: Beginner to Guru**
⇒ハンズオン形式の授業で、実践的なコーディング力を鍛えられる。




Spring Frameworkの学習に使っているもの


- 書籍・e-Learning・サンプルプロジェクトのソースコード

3. GitHub上で公開されているサンプルプロジェクト

- Spring PetClinic Sample Application

<https://github.com/spring-projects/spring-petclinic>

 HOME FIND OWNERS VETERINARIANS ERROR				
Owners				
Name	Address	City	Telephone	Pets
Jeff Black	1450 Oak Blvd.	Monona	6085555387	Lucky
Jean Coleman	105 N. Lake St.	Monona	6085552654	Max Samantha
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Harold Davis	563 Friendly St.	Windsor	6085553198	Iggy
Maria Escobito	345 Maple St.	Madison	6085557683	Mulligan
Carlos Estaban	2335 Independence La.	Waunakee	6085555487	Lucky Sly
George Franklin	110 W. Liberty St.	Madison	6085551023	Leo
Peter McTavish	2387 S. Fair Way	Madison	6085552765	George
Eduardo Rodriguez	2693 Commerce St.	McFarland	6085558763	Jewel Rosy
David Schroeder	2749 Blackhawk Trail	Madison	6085559435	Freddy

 by Pivotal

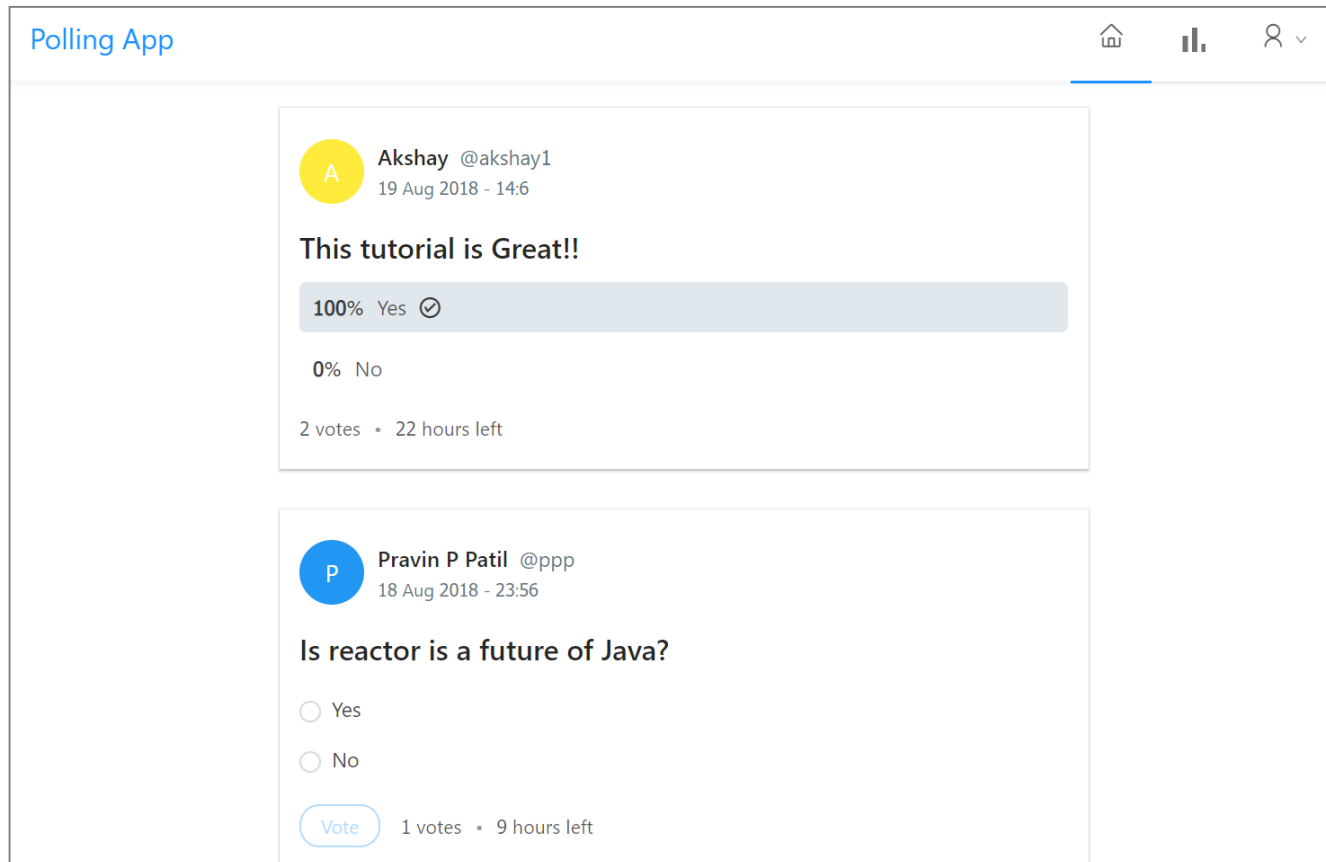
Spring Frameworkの学習に使っているもの

- 書籍・e-Learning・サンプルプロジェクトのソースコード

3. GitHub上で公開されているサンプルプロジェクト

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

<https://github.com/callicoder/spring-security-react-ant-design-polls-app>



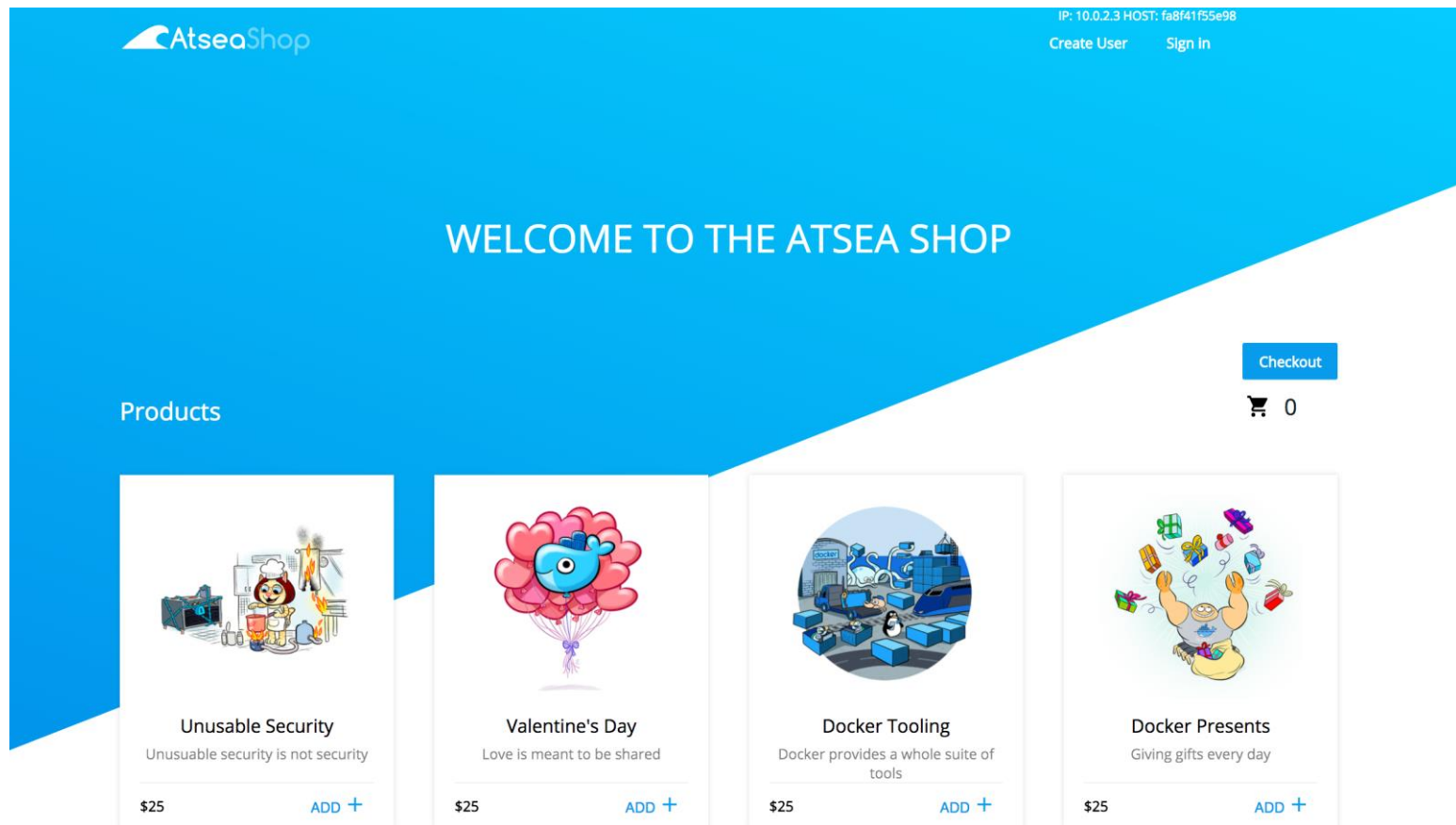
Spring Frameworkの学習に使っているもの

- 書籍・e-Learning・サンプルプロジェクトのソースコード

3. GitHub上で公開されているサンプルプロジェクト

- AtSea Shop Demonstration Application

<https://github.com/dockersamples/atsea-sample-shop-app>



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）。

- ・ 今の自分自身のスキルの中で何が足りないのか認識させてくれる。
- ・ エンジニアが持つべき、当たり前感覚・設計思想が身につく。

- 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高い (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。