Spring開発

今後の流れ

3月

- Spring フレームワークの概要について勉強
- 簡単なプロジェクトの作成
- Git リポジトリの作成

4月

プロジェクトの作成方法

Spring Initialize（プロジェクト作成ツール）

https://start.spring.io/　にて、プロジェクトを作成

【設定内容】

Project
　Gradle - Groovy
Spring Boot
　SNAPSHOTは開発中のバージョン
　Mxは正式リリース前
　無印が安定版
Group
　通常は組織名や会社名が入る
Artifact
　プロジェクト名
Name
　プロジェクト表示名（通常はプロジェクト名と同じ）
Description
　プロジェクトの説明
Package name
　自動生成
Packaging
　Springでは jar が一般的
Java
　Javaのバージョン（17が安定してそう？）

【依存関係】※検索すると見つけやすい

Dependencies
　Spring Boot DevTools
　　自動リロード
　Lombok
　　アノテーションなどの自動補完
　Spring Web
　　Springのアプリ機能一式
　MyBatis Framework（DB無しのテストではエラーが出るため不要）
　　JavaとSQLを繋ぐORマッパー
　Thymeleaf
　　HTMLテンプレートエンジン
　MySQL Driver（DB無しのテストではエラーが出るため不要）
　　JavaがMySQLに接続するライブラリ

【インポート方法】

「GENERATE」ボタンでプロジェクト作成
ダウンロードされたzipフォルダを解凍する
プロジェクト用のフォルダの中に解凍したプロジェクトを設置

Ecripseを開く
ファイル→インポート→既存のGradleプロジェクトを選択
次へをクリック
プロジェクトルートディレクトリに設置したプロジェクトを指定
完了をクリック

JAVA命名規約

https://qiita.com/fsd-fukufuku/items/cfbf30e568ef9113c099

パッケージ名	lowercase
クラス名	UpperCamelCase
メソッド名	lowerCamelCase

Springドキュメント

https://spring.pleiades.io/

必要な機能

・ルーティング

アクセスしたURLに応じて、HTMLやデータを返す

メソッド

GET, POST, PUT, DELTE

・DB操作

CRUD処理

create
read
update
delete

ORM

オブジェクトリレーションマッピング

・バリデーション

フォームの入力チェック

・認証機能

ログイン機能

ライブラリとツール

Lombok（ロンボック）

アノテーションによる「getter」「setter」の自動生成

@Getter, @Setter
@Data
@ALLArgsConstructor
@NoArgsConstructor

Grade（グレードル）

ビルドツール

Maven との違い？

Thymeleaf（タイムリーフ）

テンプレートエンジン

HTMLにデータを入れる

SpringFrameworkのコア機能

Dependency Injection : 依存性の注入

通称：DI

- @Controller → アプリケーション層（コントローラー）
- @Service → ドメイン層（ビジネスロジック）
- @Repository → インフラストラクチャ層（DBアクセス）
- @Bean → ？
- @Configuration → 設定クラスであることを示す
- @Autowired → フィールドインジェクション、セッターインジェクション、コンストラクタインジェクション（推奨）

Aspect Oriented Programming : アスペクト指向プログラミング

通称：AOP

固有用語

- Advice：特定のタイミングで実行されるコード
- JoinPoint：Adviceが実行される具体的な場所
- Pointcut：どの JoinPoint で Advice が実行されるかを指定
- Aspect：Advice と Pointcut を組み合わせたもの
- Interceptor：特定の操作の前後で処理を行う
- Target：Aspect が適用される対象

トランザクション管理

- @Transactional

Spring MVC

Spring MVC（Model-View-Controller） は、Webアプリケーションの構造を
「関心の分離（Separation of Concerns）」によって整理するアーキテクチャパターン

Model：データ（EntityやDTOなど）
View：表示（HTML, Thymeleafなど）
Controller：リクエストの制御（処理の入口）

フォルダ構成全体（src/main/java）

```
src
└── main
    └── java
        └── com.example.demo
            ├── controller      ← Web層（API/画面の受け口）
            ├── service         ← ビジネスロジック
            ├── repository      ← データアクセス層（DB処理）
            ├── model           ← エンティティ/DTO/リクエスト/レスポンスなど
            ├── security        ← 認証・認可に関する設定やフィルター等
            ├── externalapi     ← 外部APIとの通信ロジック
            ├── exception       ← 独自例外・例外ハンドリング
            ├── util/           ← 汎用クラス・ユーティリティ
            ├── config          ← 設定クラス（Security, CORS, DBなど）
            └── DemoApplication.java
```

フォルダ名	役割
controller	RES API / MVCコントローラー（@RestController, @Controller）
service	ビジネスロジック（@Service）
repository	DBアクセス（@Repository, MyBatis, JPA）
model	Entity, DTO（DataTransferObject）
security	Spring Securityの設定クラス、JWT認証、カスタムフィルター、UserDetailsなど
externalapi	他システム（他サービス、REST API等）との連携ロジック
exception	独自例外クラス ・例外レスポンス ・@ControllerAdviceによる例外ハンドラ
util	日付変換・文字列操作・共通処理などの静的メソッドユーティリティクラス
config	Springの設定系（SecurityConfig, WebMvcConfig, etc...）

フォルダ構成詳細（controller）

```
controller
├── api
│   ├── user/
│   │   ├── UserApiController.java
│   │   ├── request
│   │   └── response
│   └── product
│       ├── ProductApiController.java
│       ├── request
│       └── response
├── web
│   ├── user
│   │   └── UserWebController.java
│   └── product
│       └── ProductWebController.java
```

フォルダ名	役割
api	RES API（@RestController）
api/{ドメイン}	ドメインごとのコントローラー
api/{ドメイン}/request	フロントからのリクエスト構造定義 @RequestBody, @ModelAttribute バリーデーションの設定（@NotNull, @Size, @Email, etc...）
api/{ドメイン}/response	APIレスポンス専用クラス 必要な項目を整形して返す（Entityをそのまま渡さない）
web	WEB画面（@Controller）
web/{ドメイン}	ドメインごとのコントローラー

フォルダ構成詳細（service, repository, model）

```
service
├── user
|   └── UserService.java
├── product
|   └── ProductService.java

repository
├── user
|   └── UserRepository.java
├── product/
|   └── ProductRepository.java

model/
├── entity/
|   ├── User.java
|   └── Product.java
├── dto/
|   ├── user/
|   └── product/
```

フォルダ名	役割
service/{ドメイン}	ビジネスロジック（アプリケーション固有の処理）を記述する層 ControllerとRepositoryの「橋渡し」的存在 基本的に @service でマークされるクラス ビジネスルールやユースケースをここに実装 データの加工・集計・バリデーションなどもここで処理する
repository/{ドメイン}	データベースなどの**永続化処理（CRUD）を担当する層 主にエンティティの取得・保存・削除といったDBアクセスを担う @Repositoryでマークされたインターフェースやクラス
model/{ドメイン}/entity	DBのテーブルに対応するクラス（ORM） フィールドはDBカラムに対応（基本的にそのままDB保存・取得に使う）
model/{ドメイン}/dto	data transfer object データ受け渡し専用オブジェクト（Service層、外部API連携など） 他サービスの連携でEntityと切り離して使いたい時に使う

フォルダ構成詳細（service, repository, model）

```
security
 ├── auth
 |    ├── AuthFilter.java
 |    ├── AuthEntryPoint.java
 |    └── TokenProvider.java
 ├── config
 |    └── SecurityConfig.java
 ├── user
 |    └── CustomUserDetailsService.java

externalapi
 ├── payment
 |    ├── PaymentClient.java

exception
 ├── handler
 |    └── GlobalExceptionHandler.java
 ├── custom
 |    ├── ValidationException.java
 |    └── UnauthorizedException.java
 └── ErrorResponse.java
```

フォルダ名	役割
security/auth	JWTフィルタ・認証処理
security/config	Spring Securityの設定クラス
security/user	UserDetailsServiceのようなユーザー認証連携系 ユーザー名に対応するユーザー情報をDBから持ってくる 認証連携の橋渡し役
externalapi/{ドメイン}	ドメイン単位またはAPI単位で分ける 外部APIとのクライアントコードをまとめる場所
exception/handler	グローバルな例外処理 Controllerで発生した例外を共通的にキャッチして処理するクラスです。 @ControllerAdvice + @ExceptionHandler を使う
exception/custom	業務ロジックやエラーパターンに応じたカスタム例外を定義する
exception/ErrorResponse	例外レスポンス用のDTO

フォルダ構成詳細（service, repository, model）

```
util/
├── date/
|   └── DateUtil.java
├── string/
|   └── StringUtil.java

config/
├── mapper/
|   └── MyBatisConfig.java
├── global/
|   ├── AppProperties.java
|   └── GlobalCorsConfig.java
```

フォルダ名	役割
util/date	共通処理（日付関連の処理） 他のクラスに依存しない独立したクラス
util/string	共通処理（文字列操作） 他のクラスに依存しない独立したクラス
config/mapper	MyBatisなどマッピング設定 SQLとJavaオブジェクトの設定 データをどのようにDBと結びつけるか仕組みの設定
config/global/AppProperties	アプリケーション共通設定（定数・プロパティ）をまとめて管理 @ConfigurationProperties を使って、application.yml の内容をバインドできる
GlobalCorsConfig	CORS（クロスオリジンリクエスト）対応の設定を行います フロントエンド（React/Angularなど）とAPIサーバーが別ドメインの場合に必要

ショートカット

インポート編成：ctl + shift + o
　インポート記述の自動化（不要なものは削除してくれる）
クイック・アウトライン（開く）：ctl + o
クイック・アウトライン（閉じる）：esc