Section

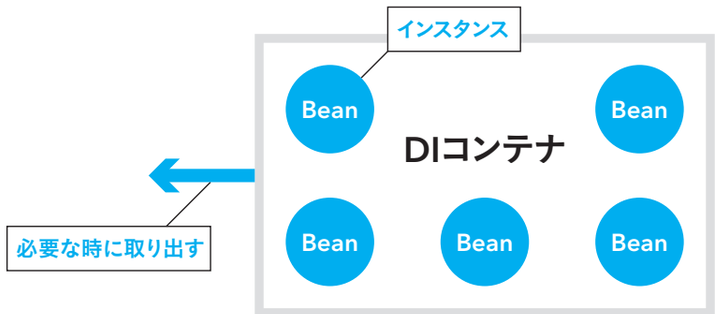
3-3 DIについて知ろう（インスタンス生成）

ここではDIについて深堀します。引き続き見慣れない言葉が出てきますが「そうなんだ」位の気持ちで学習してください。まずは「Bean」という言葉から説明します。

3-3-1 Beanとは？

Spring Frameworkは、「DIコンテナ」というJavaインスタンスを生成する機能を持っています。アプリケーションの起動時に必要な設定を読み込み（コンポーネントスキャン）、インスタンスを生成してDIコンテナに保持します。DIコンテナで管理されているインスタンスのことを「Bean」と呼びます。簡単に考えると「Bean」を必要なときに取り出して処理をさせるのがSpringの使用方法になります（**図3.18**）。

図3.18 Bean



3-3-2 Bean定義とは？

Spring Frameworkに「このクラスをBeanにします」と指示することを「Beanを定義する」と言います。Beanを定義する主要な方法には、下記の3つがあります。

① クラスにアノテーションを付加する
② Java Configクラスにメソッドを作成する
③ XML設定ファイルに記述する

本書では、Springアプリケーション開発で近年よく利用されている①と②を説明します。

①については、「**3-2 DIについて知ろう**」で説明していますので、ここでは②の「Java Configクラスにメソッドを作成する」についてプログラムを作成しながら説明します。

3-3-3 Java Configを使用したプログラム

インターフェース「BusinessLogic」、インターフェースを実装した「TestLogicImpl」クラスと「SampleLogicImpl」クラスを各々作成後、「Java Config」クラスを作成し、DIの動きを確認できるプログラムを作成します。

01 プロジェクトの作成

eclipseを起動し、メニューの左上から「ファイル」→「新規」→「Springスターター・プロジェクト」を選択します。「Springスターター・プロジェクト」が見つからない場合は、「その他」を選択し「Springスターター・プロジェクト」を検索してください。「新規Springスターター・プロジェクト」画面で、以下のように入力後「次へ」ボタンを押します。

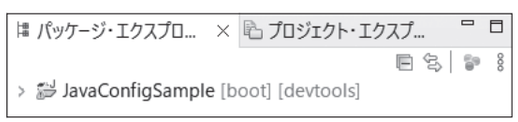
設定内容

名前	JavaConfigSample
タイプ	Gradle-Groovy
パッケージング	Jar
Javaバージョン	21
言語	Java

※ 他はデフォルト設定

依存関係で「Spring Boot DevTools（開発ツール）」を選択後「完了」ボタンを押し、プロジェクトが作成されます（**図3.19**）。

図3.19 プロジェクト作成



02 「使われる側」インターフェースと実装クラスの作成

「JavaConfigSample」の「src/main/java」フォルダを選択し、マウスを右クリックし、「新規」→「インターフェース」を選択します。インターフェース設定画面にて以下の「設定内容」を記述後、「完了」ボタンを押します。

記述例

```
private SomeService someService;

@Autowired
public void setSomeService(SomeSerivice someService) {
    this.someService = someService;
}
```

コンストラクタインジェクション

コンストラクタインジェクションの主な特徴と記述例は以下のとおりです（**表3.6**）。

表3.6　コンストラクタインジェクションの特徴

概要	コンストラクタを通じて依存性を注入する方法
記述方法	コンストラクタに@Autowiredアノテーションを付与する
特徴	不変性※が保たれ、テスト時にモック化が容易になる

※ 不変性とは、あるオブジェクトが一度作成された後、その状態やデータが変更されない特性を指します。つまり、オブジェクトの内容は固定され、後から変更することができません。

記述例

```
private final SomeService someService;

@Autowired
public SomeClass(SomeService someService) {
    this.someService = someService;
}
```

Spring 4.3以降、コンストラクタが1つだけの場合、「@Autowired」を省略することができます。これにより、コンストラクタインジェクションがさらにシンプルになります。

```
private final SomeService someService;

@Autowired  ←  省略可能
public SomeClass(SomeService someService) {
    this.someService = someService;
}
```


Column　インジェクションの推奨方法

インジェクションの方法で推奨されるのは「コンストラクタインジェクション」です。

推奨される理由はフィールドに「final」修飾子を付与して不変性を保証する場合、「コンストラクタインジェクション」のみが使用できるためです。なお、Javaでは「final」修飾子を付与したフィールドは、その値が変更不可能になります。

これらのフィールドの値は、宣言時かコンストラクタ内でのみ設定できます。

初めから「コンストラクタインジェクション」の方法を教えれば良いだろうと思った方がいると思いますが、「フィールドインジェクション」の方が「DI」のイメージをしやすいので、本書では「フィールドインジェクション」から説明させて頂きました。「フィールドインジェクション」は非推奨です。

3-4-2　各インジェクションを利用したプログラムの作成

「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」、「Lombok」との連携を用いた「インジェクション」の動きを確認できるプログラムを作成します。

01　プロジェクトの作成

eclipseを起動し、メニューの左上から「ファイル」→「新規」→「Springスターター・プロジェクト」を選択します。「新規Springスターター・プロジェクト」画面で、以下のように入力後「次へ」ボタンを押します。

設定内容

名前	InjectionSample
タイプ	Gradle-Groovy
パッケージング	Jar
Javaバージョン	21
言語	Java

※ 他はデフォルト設定

依存関係で「Spring Boot DevTools（開発者ツール）」、「Lombok（開発者ツール）」を選択後「完了」ボタンを押し、プロジェクトが作成されます（**図3.22**）。

Section 4-1

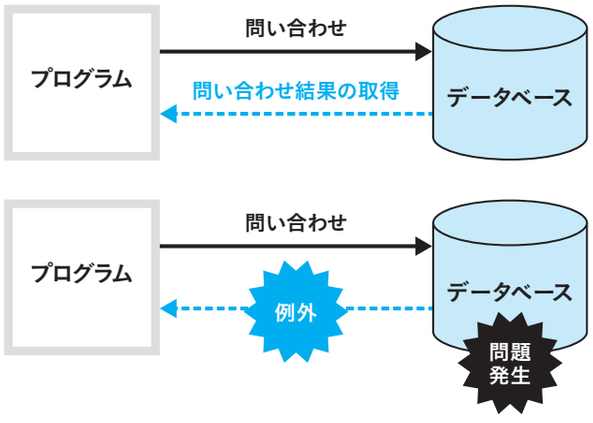
AOP（アスペクト指向プログラミング）の基礎を知ろう

「3-1 Spring Frameworkのコア機能」で「Aspect Oriented Programming：アスペクト指向プログラミング）」略して「AOP」について簡単に説明しました。ここではAOPについて、もう少し詳しくデータベースへのアクセス処理プログラムを例に説明します。

4-1-1 AOPの例（データベースアクセス）

データベースアクセス処理には例外発生時の対応処理を必ず含める必要があります。例外処理を記述しないと、プログラムが停止しますし、Javaの場合は例外処理をプログラムに含めないとコンパイルもできません（**図4.1**）。

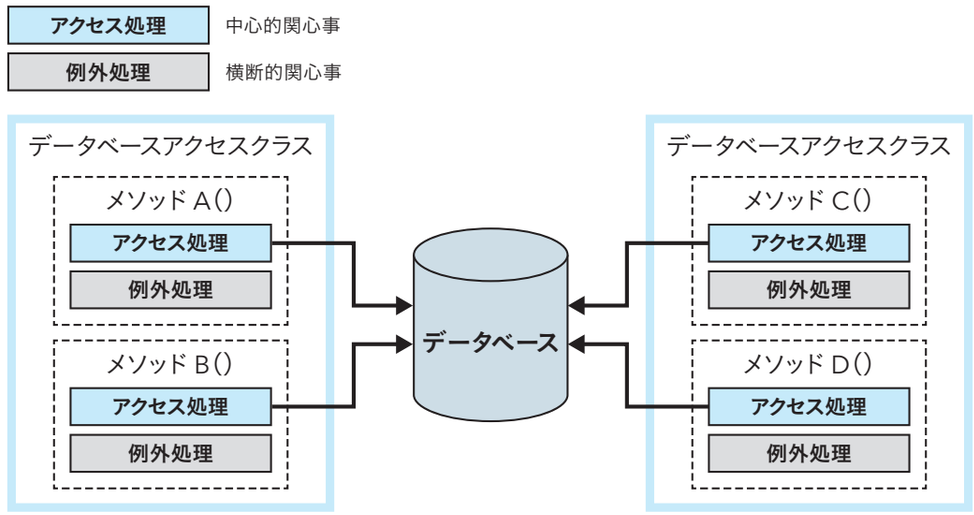
図4.1 データベースへのアクセス処理



多数のデータベースアクセス処理を作成すると、例外処理の内容はいつも同じですが、例外処理は必要なため常に作成しなければなりません。例外処理を含めるとプログラムコードは増え、複雑になってしまいます。「実現したいプログラム」は「データベースへのアクセス処理」であり、「例外処理」は「実現したいプログラムに付随する」プログラムになります。

「実現したいプログラム＝中心的関心事」、「付随するプログラム＝横断的関心事」を分離してプログラムを作成できないでしょうか（**図4.2**）。

図4.2 データベースへのアクセス処理での「中心的関心事」と「横断的関心事」



4-1-2 AOPの固有用語

Spring Frameworkが提供しているAOP機能を利用することで「中心的関心事」と「横断的関心事」を分離してプログラムを簡単に作成できます。

具体的な使用方法の説明に入る前にAOPの固有用語について説明します（**表4.1**、**図4.3**）。

表4.1 AOP固有用語

用語	内容
Advice **「アドバイス」**	特定のタイミング（例えば、メソッドが呼び出される前や後）で実行されるコードです。つまり「横断的関心事」を具体的にコードで表現したもの（メソッド）です
JoinPoint **「ジョインポイント」**	アドバイスが実行される具体的な場所です
Pointcut **「ポイントカット」**	どのジョインポイント（場所）でアドバイス（コード）が実行されるかを指定するための表現やパターン、つまりは条件です
Aspect **「アスペクト」**	アドバイス（何をするか）とポイントカット（いつ実行するか）を組み合わせたもの（クラス）です。つまり「横断的関心事」を「どのように」かつ「どこで」実行するかを定義したものです
Interceptor **「インターセプタ」**	インターセプタは、特定の操作（通常はメソッド呼び出し）を「捕捉」して、その前後で何らかの処理を行うオブジェクトです
Target **「ターゲット」**	ターゲットは、アスペクトが適用される対象です

Section

5-3 Spring MVCを使ってみよう

これから「Spring MVC」を使用して、Webアプリケーションを作成していきますが、もしWebの知識について自信がないという方は、「2-2 Webアプリケーション作成の必須知識を確認しよう」を参照してから、プログラムの作成に進んでください。

5-3-1 Spring MVCのプログラムの作成

「リクエスト」で送られる「URL」に対応する「メソッド」のことを「リクエストハンドラメソッド」と呼びます。「コントローラ」に「リクエストハンドラメソッド」を作成後、「ビュー」を作成して「ブラウザ」で「Hello View!!!」と表示するプログラムを作成します。

01 プロジェクトの作成

eclipseを起動し、メニューの左上から「ファイル」→「新規」→「Springスターター・プロジェクト」を選択します。「新規Springスターター・プロジェクト」画面で、以下のように入力して「次へ」ボタンを押します。

設定内容

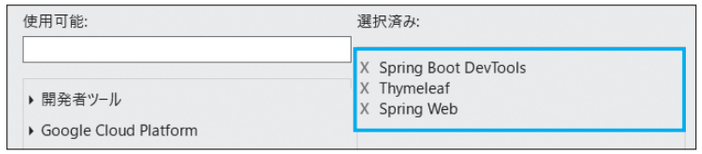
名前	SpringMVCViewSample
タイプ	Gradle-Groovy
パッケージング	Jar
Javaバージョン	21
言語	Java

※ 他はデフォルト設定

依存関係で以下を選択して、「完了」ボタンを押します（**図5.7**）。なお、選択した「Spring Web（Web）」が「Spring MVC」になります。

- Spring Boot DevTools（開発者ツール）
- Thymeleaf（テンプレートエンジン）
- Spring Web（Web）

図5.7 依存関係

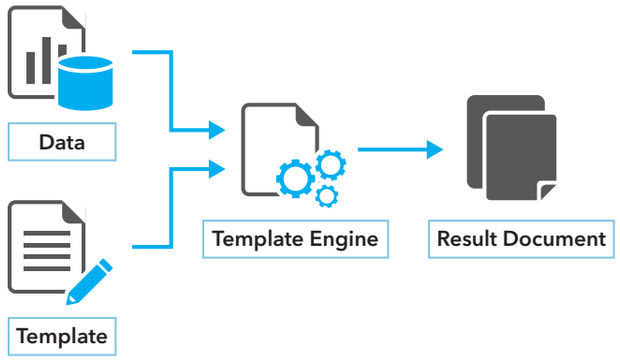


Column Thymeleaf（タイムリーフ）とは？

テンプレートエンジンは、データと特定の形式（テンプレート）を組み合わせて、最終的な表示内容（ビュー）を作成するツールです（**図5.B**）。

Thymeleaf（タイムリーフ）はそのようなテンプレートエンジンの一つで、Spring Bootで使用が推奨されています。簡単に言えば、ThymeleafはSpring Bootと一緒に使うと、データを画面に表示する作業を簡単にしてくれます。

図5.B テンプレートエンジン



Column Spring Web MVC（Spring MVC）とは？

Springが提供するWebアプリケーションを開発するためのフレームワークです。このフレームワークは、MVC（Model-View-Controller）という設計パターンに基づいています。

02 コントローラの作成

実際に私達が作成する部分の「コントローラ」を作成します。「SpringMVCViewSample」の「src/

Section

7-2 複数のリクエストパラメータを送ろう

「ビュー」で入力した複数の値をサーバーに送信し、「Form」クラスを利用して複数値を受け取ってみましょう。「@RequestParam」は便利ですが、渡す値が増える程、リクエストハンドラメソッドで「@RequestParam」が付与された「引数」が増えるので煩雑です。

7-2-1 「@RequestParam」で複数値を受け取る

先ほど作成した「RequestParamSample」プロジェクトを使用します。

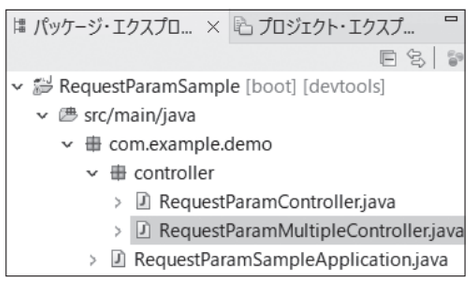
01 コントローラとビューの作成

コントローラの作成

「コントローラ」を作成します。「src/main/java」→「com.example.demo」フォルダを選択し、マウスを右クリックし、「新規」→「クラス」を選択します。

パッケージを「com.example.demo.controller」名前を「RequestParamMultipleController」としてクラスを作成します（**図7.11**）。

図7.11 コントローラの作成



「RequestParamMultipleController」クラスの内容は**リスト7.7**になります。

リスト7.7 RequestParamMultipleController

```
001: package com.example.demo.controller;
002:
003: import org.springframework.stereotype.Controller;
004: import org.springframework.web.bind.annotation.GetMapping;
005:
006: @Controller
007: public class RequestParamMultipleController {
008:
009:     // GET かつ [url：/multiple]
010:     @GetMapping("multiple")
011:     public String showView() {
012:         // 戻り値は「ビュー名」を返す
013:         return "entry";
014:     }
015: }
```

クライアントからURL「http://localhost:8080/multiple」がGETメソッドで送信されると、RequestParamMultipleControllerクラスのshowViewメソッドが呼ばれ、13行目で戻り値として「ビュー名：entry」を返します。

ビューの作成

「showView」メソッドの戻り値「ビュー名：entry」に対する「entry.html」を作成し「resources/templates」フォルダに配置します。「src/mai/resources」→「templates」フォルダを選択し、マウスを右クリックし、「新規」→「その他」を選択します。「HTMLファイル」を選択し「次へ」ボタンを押し、ファイル名に「entry.html」と入力後「完了」ボタンを押します。

entry.htmlの内容は**リスト7.8**のようになります。

リスト7.8 entry.html

```
001: <!DOCTYPE html>
002: <html xmlns:th="http://www.thymeleaf.org">
003: <head>
004:     <meta charset="UTF-8">
005:     <title>入力画面</title>
006: </head>
007: <body>
008:     <form th:action="@{/confirm}" method="post">
009:         <div>
010:             <label for="name">名前：</label>
011:             <input type="text" name="name">
012:         </div>
013:         <div>
014:             <label for="age">年齢：</label>
015:             <input type="number" name="age" min="1" max="100">歳
```

Section

8-1 入力チェックについて知ろう

「7章 サーバーにデータを送信する方法を学ぼう」はイメージできましたでしょうか。「ビュー」で入力を行う時に、疑問に思う内容として「数値を入力してもらいたいのに、文字列を入力された時などはどうなるのか?」があります。この章では「ビュー」で入力した値に対して「入力チェック」を行う「バリデーション」機能について説明します。

8-1-1 バリデーションとバリデータとは?

バリデーション

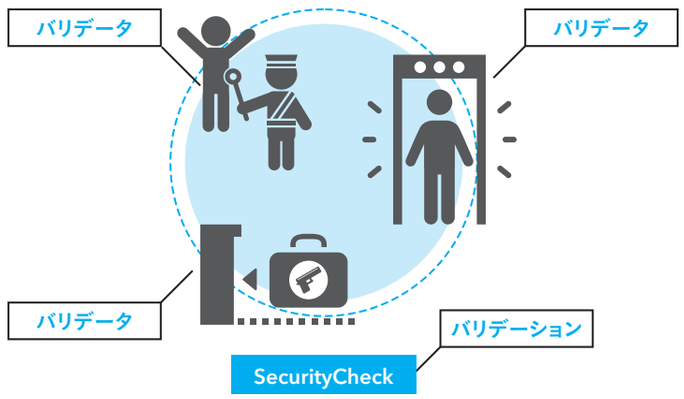
「バリデーション」はユーザーが入力したデータ（例：名前、メールアドレス、パスワードなど）が「適切な形式」であるかどうかを確認する方法です。現実世界で例えると「空港のセキュリティチェック」のようなものです。セキュリティチェックは、乗客が飛行機に乗る前に、持ち物が安全かどうかを確認します。

バリデータ

「バリデータ」はプログラム内でデータが正しい形式になっているかをチェックする機能やコードのことを指します。つまりバリデーションを行う機能、またはプログラムのことです。

現実世界で例えると空港のセキュリティチェックを行う「スタッフや機械」のようなものです（**図8.1**）。

図8.1 バリデーションとバリデータのイメージ



バリデーションの種類

バリデーションを大きく分けると以下の2つに分かれます。

- 単項目チェック
- 相関項目チェック（相関チェック）

8-1-2 単項目チェックとは?

「単項目チェック」とは、入力項目1つ1つに対して、設定する「入力チェック」機能です。使用方法はFormクラスなどのフィールドに「アノテーション」を付与します。

「入力チェック」にはいくつかの種類の「アノテーション」があります。主にJava EEの「Bean Validation」やHibernateフレームワークの「Hibernate Validator」がよく使用されます。これらのアノテーションを使用すると、例えば「テキストボックスに数値しか入力できない」ように制限したり、「メールアドレスが正しい形式であるかどうか」をチェックしたりできます。

また、数値の入力フィールドに文字列が入力された場合のような「型変換チェック」は、特別なアノテーションを使わずに「入力チェック」機能を有効にするだけで自動的に行われます。

表8.1に「Bean Validation」に定義されている検証アノテーションの一覧を示します。これらのアノテーションは、「javax.validation.constraints」パッケージに定義されています。

表8.1 Bean Validation

アノテーション	説明	使用例
@NotNull	値がnullでないことを確認します	@NotNull private String name;
@NotEmpty	文字列が空でないこと、またはコレクションが空でないことを確認します	@NotEmpty private String name;
@NotBlank	文字列が空白でないことを確認します	@NotBlank private String name;
@Min	数値が指定した最小値以上であることを確認します	@Min(18) private int age;
@Max	数値が指定した最大値以下であることを確認します	@Max(100) private int age;
@Size	文字列、配列、またはコレクションのサイズが指定した範囲内であることを確認します	@Size(min=1, max=10) private String name;
@Pattern	文字列が指定した正規表現に一致することを確認します	@Pattern(regexp="^[a-zA-Z]+$") private String name;
@Email	文字列が有効なメールアドレス形式であることを確認します	@Email private String email;

Section 9-1

MyBatisについて知ろう

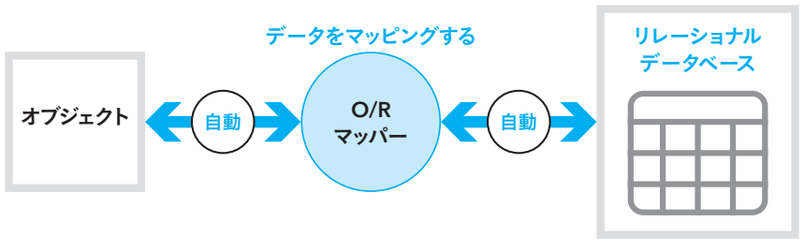
現在のJavaプログラム開発では、データベースとのアクセス処理には「O/Rマッパー」というフレームワークを使用する開発が多いです。ここではO/Rマッパーについて簡単に説明し、Springと相性が良いO/Rマッパー「MyBatis」を使用してプログラムを作成していきます。

9-1-1 O/Rマッパーとは？

「O/Rマッパー」を簡単に説明すると、アプリケーションで扱う「O：オブジェクト」と「R：リレーショナルデータベース」とのデータをマッピングするものです。

もう少し詳細に説明すると「O/Rマッパー」は、あらかじめ設定された「O：オブジェクト」と「R：リレーショナルデータベース」との対応関係の情報に基づき、インスタンスのデータを対応するテーブルへ書き出したり、データベースから値を読み込んでインスタンスに代入したりといった操作を自動的に行なってくれます（**図9.1**）。

図9.1 O/Rマッパーのイメージ



9-1-2 MyBatisとは？

「MyBatis」は、「O/Rマッパー」の一つです。オブジェクトとRDB（リレーショナルデータベース）の「マッピング」を行います。MyBatisは無料で使用できるオープンソースソフトウェアで、Apache License[注1]というライセンスの下で配布されています。

MyBatisの特徴は、SQLを直接書くことができる点です。SQLを「XML」ファイルやJavaの「アノテーション」を利用して書くことができます。

また、MyBatisはSpringと連携することで、より効率的にアプリケーション開発をすることができます。

（**注1**） Apache Licenseは、商用・非商用を問わず自由に使用、変更、再配布することができます。

9-1-3 MyBatisの使用方法

MyBatisの使用方法には、「アノテーション」を使用する方法と「マッパーファイル（XMLファイル）」を使用する2つの方法があります。具体的な使用方法は後述するため、ここでは、それぞれの使用方法について簡単に説明します。

アノテーションを使用する方法

Javaのインターフェースにアノテーションを使用して、SQLを直接記述します（**リスト9.1**）。

リスト9.1 アノテーションを使用した例

```
001: public interface BookMapper {
002:     @Select("SELECT * FROM books WHERE id = #{id}")
003:     Book getBookById(int id);
004: }
```

マッパーファイルを使用する方法

XMLファイルを使用してSQLを記述します（**リスト9.2**）。このXMLファイルを「マッパーファイル」と呼びます。

マッパーファイル内では、SQLとその結果をどのようにJavaのオブジェクトにマッピングするかを定義するため、「マッパーファイル」と呼ばれます。

リスト9.2 マッパーファイルを使用した例

```
001: <mapper namespace="com.example.BookMapper">
002:     <select id="getBookById" parameterType="int" resultType="com.example.Book">
003:         SELECT * FROM books WHERE id = #{id}
004:     </select>
005: </mapper>
```

「アノテーション」を使用する方法は、簡単なSQLの場合には便利ですが、複雑なSQLや再利用が必要な場合は、「マッパーファイル」を使用する方法が推奨されます。本書では、「マッパーファイル」を使用する方法を説明します。

Section

11-1 「Domain Object」と「Repository」を作成しよう

現時点では、データベースのテーブル構造だけがわかっている状態です。色々な作成方法がありますが、ビギナーの方に私がお勧めする作成方法は、「使われる側」のクラスから作成する方法です。

11-1-1 今回作成するコンポーネント

「ToDoアプリ」で作成する「コンポーネント」の作成状況を**表11.1**に示します。**表11.1**の中のNo.6～No.9が今回作成する部分です。

表11.1 作成予定コンポーネント

No	層	コンポーネント	作成物名称	備考
1	アプリケーション層	View	ー	見た目、画面です
2	アプリケーション層	Controller	ToDoController	制御の役割を担います
3	アプリケーション層	Form	ToDoForm	「画面のフォーム」を表現します
4	ドメイン層	Service	ToDoService	インターフェースで作成します
5	ドメイン層	ServiceImpl	ToDoServiceImpl	「Service」を実装します
6	ドメイン層	Domain Object	ToDo	ここでは「Entity」と同様です
7	ドメイン層	Repository	ToDoMapper	インターフェースで作成します（今回はMyBatisのマッパーファイルを使用する方法で作成するので名前は「××Mapper」とします）
8	インフラストラクチャ層	RepositoryImpl	ー	O/Rマッパーにより自動作成されます
9	インフラストラクチャ層	O/R Mapper	ー	MyBatis

11-1-2 Domain Object：エンティティの作成

テーブル構造がわかっていることから、まずはドメイン層のコンポーネント「Domain Object」を作成します。「Domain Object」は、業務処理を実行する上で必要な概念やルールを表現する広い概念であり、その中で「識別子が同一であれば同一」とみなすコンポーネントを「Entity：エンティティ」と呼びます。まずは「todos：すること」テーブルの1レコードに対応するエンティティを作成しましょう。

「webapp」の「src/main/java」フォルダを選択し、マウスを右クリックし、「新規」→「クラス」を選択します。クラス設定画面にて以下の「設定内容」を記述後、「完了」ボタンを押します。

設定内容

パッケージ	com.example.webapp.entity
名前	ToDo

※ 他はデフォルト設定

「ToDo」クラスの内容は**リスト11.1**のようになります。

リスト11.1 ToDo

```
001: package com.example.webapp.entity;
002:
003: import java.time.LocalDateTime;
004:
005: import lombok.AllArgsConstructor;
006: import lombok.Data;
007: import lombok.NoArgsConstructor;
008:
009: /**
010:  * すること：エンティティ
011:  */
012: @Data
013: @NoArgsConstructor
014: @AllArgsConstructor
015: public class ToDo {
016:     /** することID */
017:     private Integer id;
018:     /** すること */
019:     private String todo;
020:     /** すること詳細 */
021:     private String detail;
022:     /** 作成日時 */
023:     private LocalDateTime createdAt;
024:     /** 更新日時 */
025:     private LocalDateTime updatedAt;
026: }
```

「ToDo」クラスはschema.sqlで記述した「todos」テーブルの列に対応するフィールドを持った「Entity」です。簡単に言うと「todos」テーブルの1行に対応するクラスです。

今回作成するアプリケーションでは、O/Rマッパー「MyBatis」を使用します。MyBatisは、結果データをJavaオブジェクトにマッピングする際に、そのオブジェクトのデフォルトコンスト

Section

12-2 トランザクション管理を知ろう

ここでは「4-3-1 トランザクション管理」で軽く説明した「@Transactional」アノテーションについて使用方法を含め説明します。

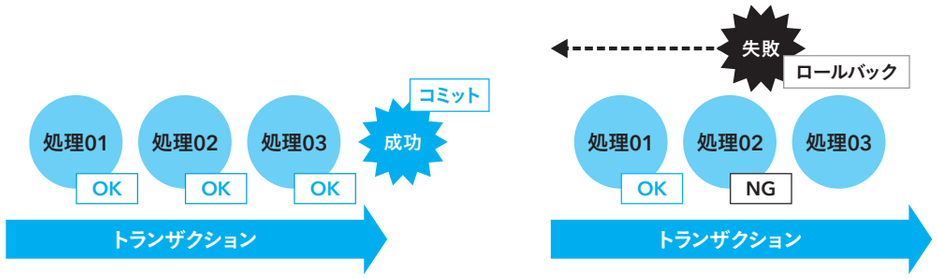
12-2-1 トランザクションとは？

「トランザクション」とはデータベースやコンピュータプログラムで使われる概念で、複数の処理を１つにまとめたものです（**図12.1**）。トランザクションは成功か失敗のどちらかしかありません。

処理の途中で失敗した場合はトランザクションの実行前に戻ります。このことを「ロールバック」といいます。

処理がすべて成功した場合は処理が確定されます。このことを「コミット」といいます。トランザクションには中途半端に「成功」する、中途半端に「失敗」するなどはありません。

図12.1 トランザクション

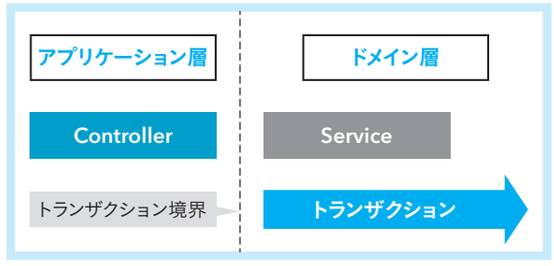


12-2-2 トランザクション境界とは？

トランザクションは開始と終了を明示的に指定する必要があり、開始から終了までの範囲を「トランザクション境界」といいます（**図12.2**）。結論から言うとトランザクション境界は「Service」の実装クラスに設定します。

MVCモデルにおいて「サービス処理」は「Model」に属します。「Service」は「Model」の一部であり、「サービス処理」の入口（開始）と捉えることができます。このことからトランザクション境界は「Service」の実装クラスに指定します。

図12.2 トランザクション境界



12-2-3 トランザクションの管理方法

「トランザクションの管理方法」は「Spring Framework」から提供されている「@Transactional」アノテーションを使用します。

使用方法は簡単です。クラスやメソッドに「@Transactional」アノテーションを付与することでトランザクションが管理され、トランザクションの「開始、コミット、ロールバック」が自動で行われます。

ロールバック発生条件は、非検査例外（RuntimeException及びそのサブクラス）が発生した場合です。検査例外（Exception及び、そのサブクラスでRuntimeException以外）が発生した場合は、「ロールバック」されず「コミット」されますので注意してください。

クラスに@Transactionalを付与する

クラスに「@Transactional」を付与することで、クラス内の「すべてのメソッド」にトランザクション制御をかけることができます（@Transactionalアノテーションは「public」メソッドのみに適用されます）。

クラスに@Transactionalを付与する場合の適用範囲とメリット・デメリットを以下に示します。

- 適用範囲
  クラスに付与した場合、そのクラス内の全てのメソッドがトランザクション管理されます
- メリット
  クラス内の全メソッドに一括でトランザクション管理を適用できるため、コードがシンプルになります
- デメリット
  トランザクションを必要としないメソッドまでトランザクション管理が適用される可能性があります。これはパフォーマンスに影響を与える可能性があります

Section

15-1 Spring Securityの概要

この章からは少し難易度が上がります。「認証（Authentication）」とは、ユーザーが自身の身元を証明する方法のことです。簡単に言えば、これは「ログイン処理」に相当します。Springには「ログイン処理」を容易に実装できる機能が備わっています。

15-1-1 Spring Securityとは？

Spring Securityは、認証や認可、およびその他多くのセキュリティ対策を簡易に実装できるSpringプロジェクトが提供するフレームワークです。多機能であるため、ビギナーの方にはやや敷居が高いかもしれません。本書ではSpring Securityの基本的な部分について説明します（**図15.1**）。より深い理解を求める方は、他の書籍を参照して学習してください。

図15.1 Spring Security

認証と認可

Spring Securityが提供する「認証」と「認可」について、**表15.1**に示します。

表15.1 認証と認可

言葉	説明
認証（Authentication）	ユーザーが自分の身元を証明する方法。簡単に言うと「ログイン」のことです
認可（Authorization）	認証されたユーザーが特定のリソースにアクセスできるかどうかを決定する方法。簡単に言うと「権限」のことです

15-1-2 メニュー画面の作成

Spring Securityの説明に入る前に、後に作成する「ログイン画面」で「認証」が成功した場合、「メニュー画面」に遷移させたいため、まずは「メニュー画面」を作成します。メニュー画面には、「ToDo一覧へのリンク」を設置します。

01 Controllerの作成

メニュー用のControllerを作成します。「webapp」の「src/main/java」フォルダを選択し、マウスを右クリックし、「新規」→「クラス」を選択します。クラス設定画面にて以下の「設定内容」を記述後、「完了」ボタンを押します。

設定内容

パッケージ	com.example.webapp.controller
名前	MenuController

※ 他はデフォルト設定

「MenuController」クラスの内容は**リスト15.1**のようになります。

リスト15.1 MenuController

```
001: package com.example.webapp.controller;
002:
003: import org.springframework.stereotype.Controller;
004: import org.springframework.web.bind.annotation.GetMapping;
005: import org.springframework.web.bind.annotation.RequestMapping;
006:
007: /**
008:  * Menu：コントローラ
009:  */
010: @Controller
011: @RequestMapping("/")
012: public class MenuController {
013:
014:     /**
015:      * メニュー画面を表示する
016:      */
017:     @GetMapping
018:     public String showMenu() {
019:         // templatesフォルダ配下のmenu.htmlに遷移
020:         return "menu";
021:     }
022: }
```