フレームワーク基礎実習

04. Spring MVC入門

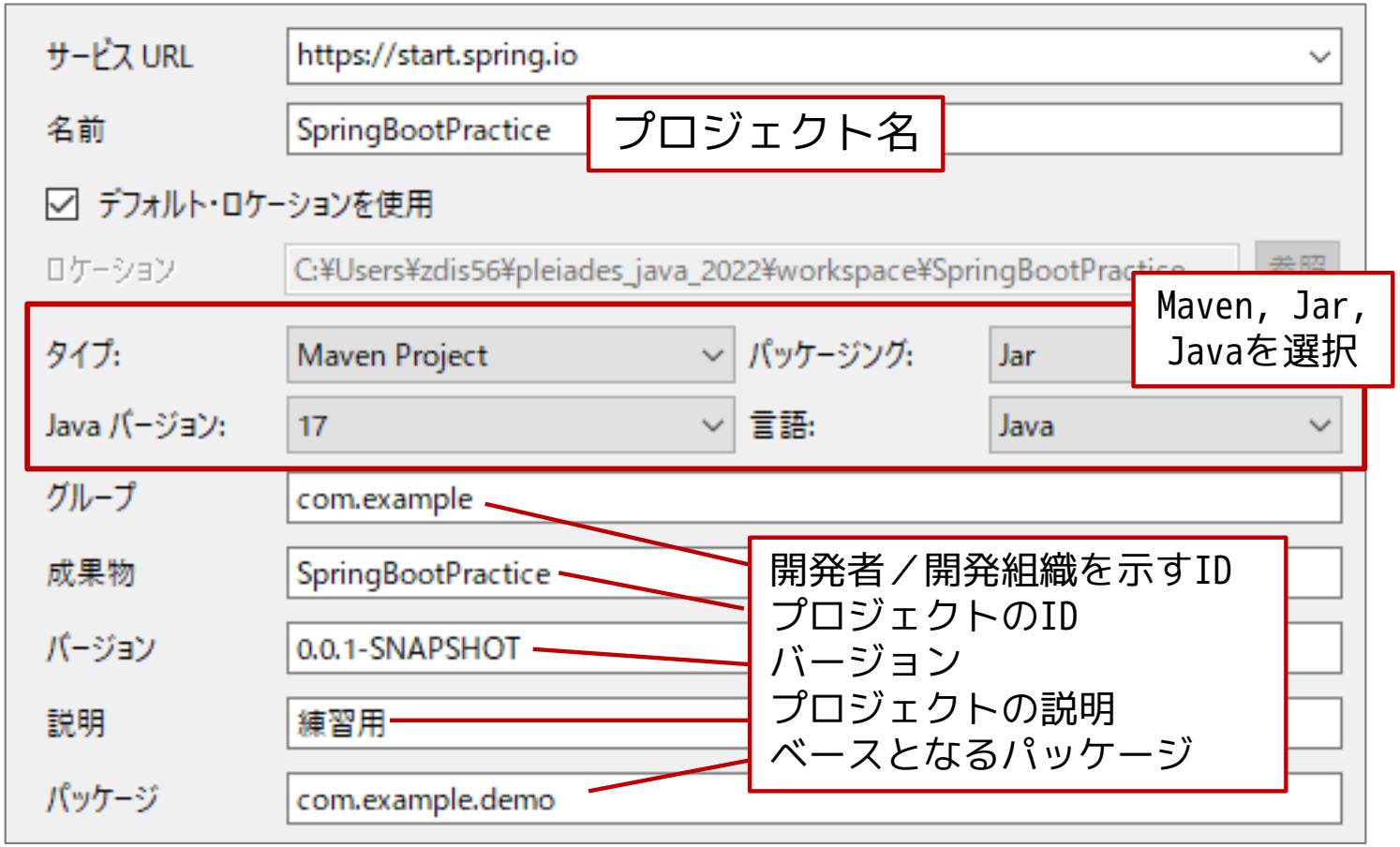
株式会社ジードライブ

今回学ぶこと

- Spring BootによるWebプロジェクトの作成
  - Webアプリケーションの実行と停止
- Spring MVCの概要
- コントローラー

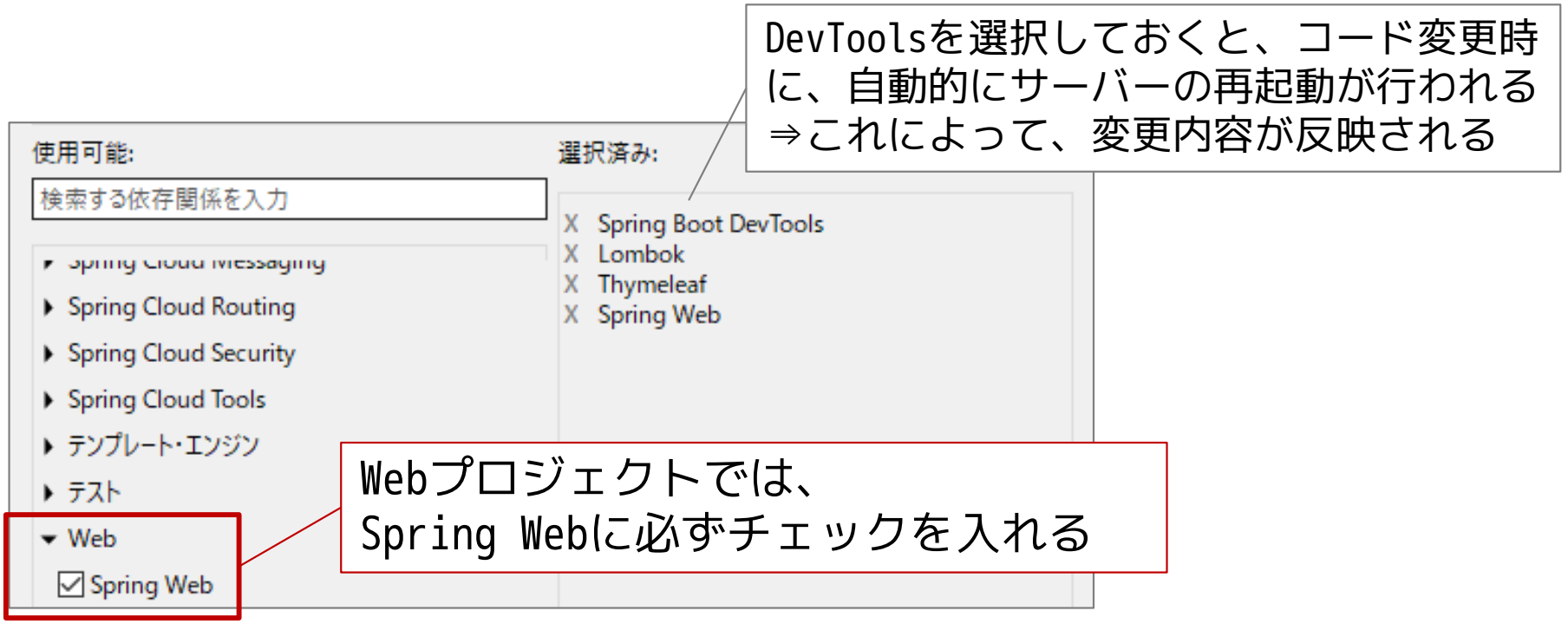
Webプロジェクトの作成

- ファイル ⇒ 新規 ⇒ Springスターター・プロジェクト



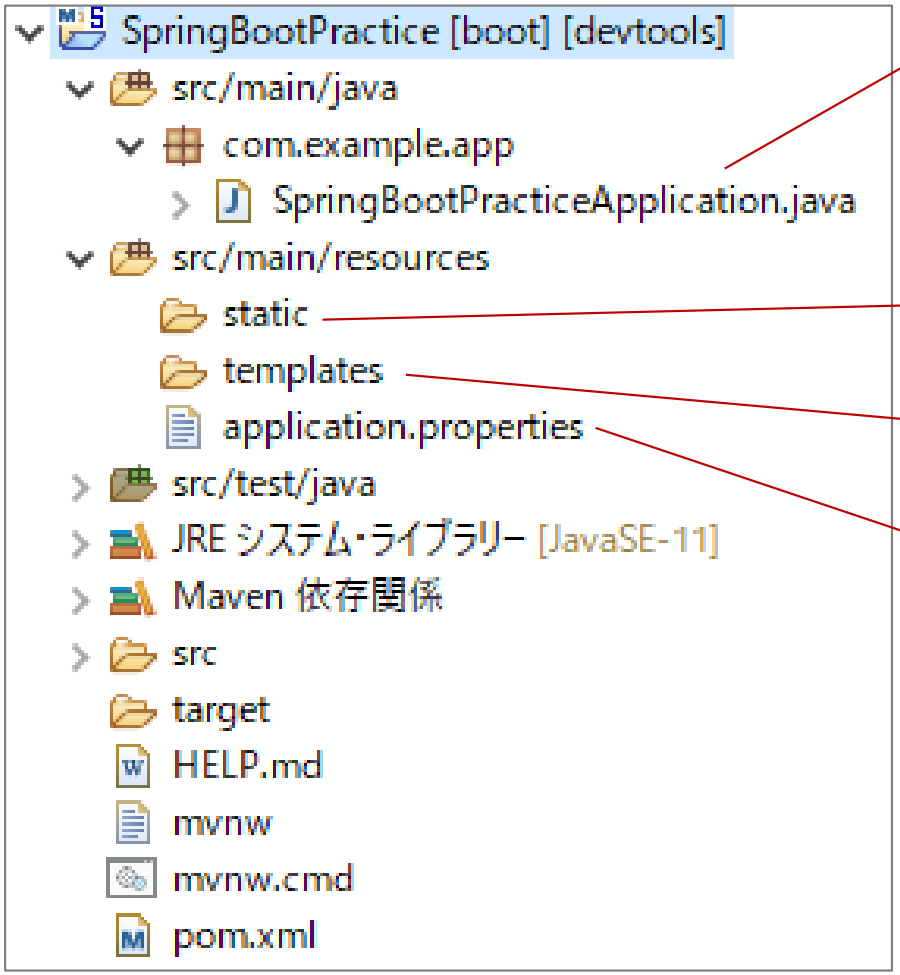
Webプロジェクトの作成

- Webプロジェクトの場合、依存関係として「Web ⇒ Spring Web」を選択する
  - 訓練では、併せて「テンプレート・エンジン ⇒ Thymeleaf」を選択する



Webプロジェクトの構成

- プロジェクト作成後は以下のような構成になる



mainメソッドが含まれており、これを実行することで、Tomcatが起動する
※ ファイル名は、プロジェクト名に応じて異なる。名称変更可。

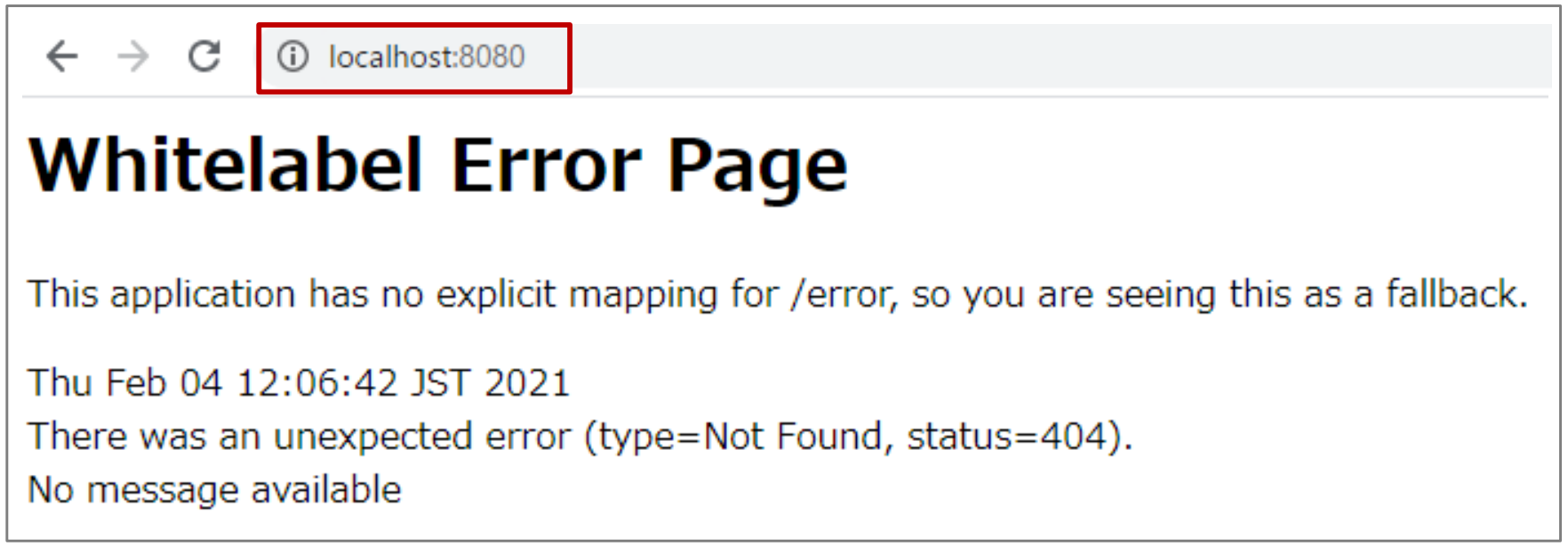
静的ファイル(画像，CSS，JS)を格納

ビューファイル(Thymeleaf)を格納

データベース接続等の設定ファイル

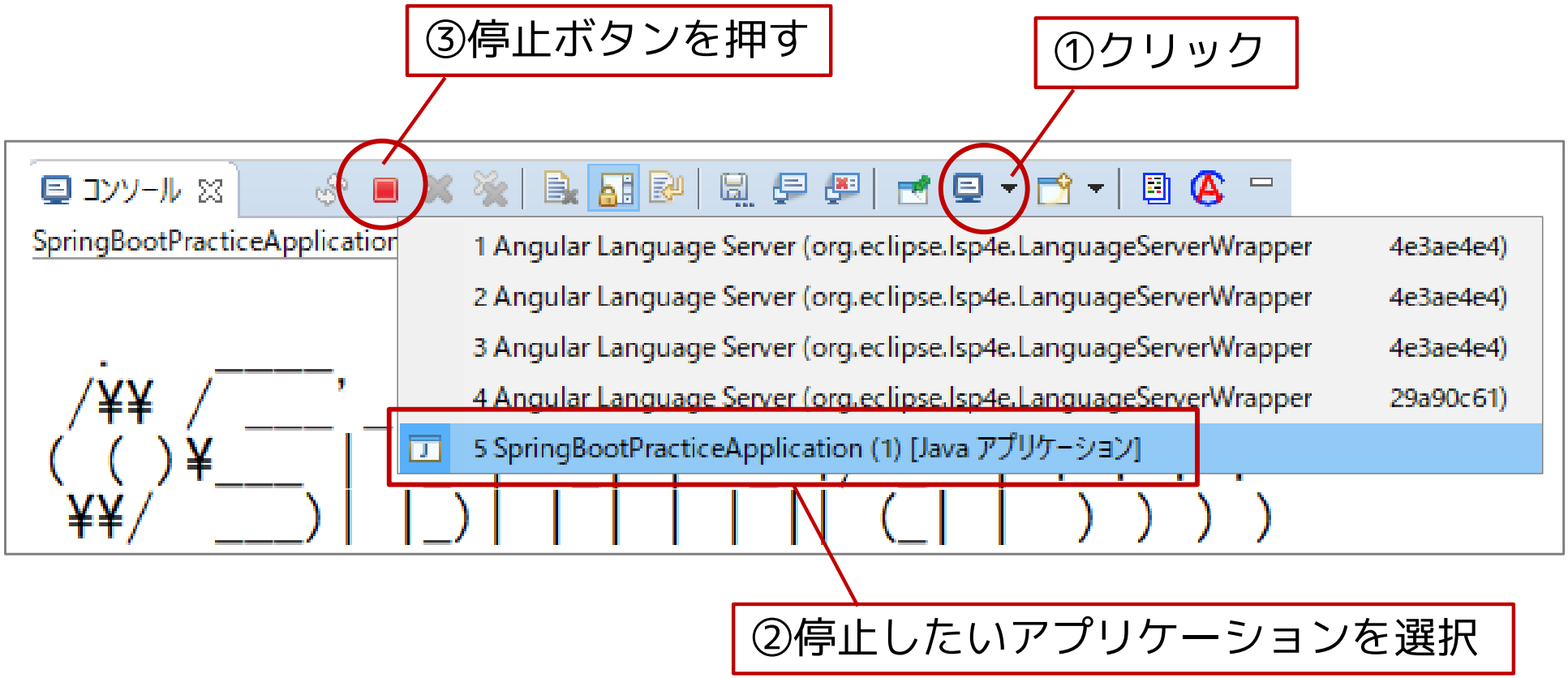
Webアプリケーションの実行

- プロジェクト作成時に生成される〇〇Application.javaのmainメソッドを実行する
  - 前頁の例では、SpringBootPracticeApplication.javaを右クリックし、「実行 ⇒ Javaアプリケーションの実行」
- ブラウザを開き、http://localhost:8080 を訪問すると、ページが表示されるようになる



アプリケーションの停止

- 停止したいアプリケーションのコンソールを開き、停止ボタンを押す



アプリケーションの停止

- コンソールで停止できない場合、コマンドプロンプトで対応する

① ポート番号8080を使用しているアプリケーションを調べる

```
netstat -aon | find "8080"
```

② 以下のように表示される

```
TCP    0.0.0.0:8080   0.0.0.0:0   LISTENING  11628
```

③で使用する番号

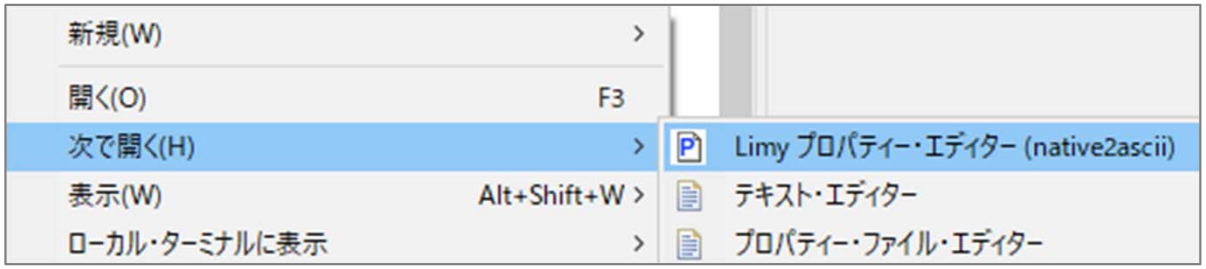
③ taskkillコマンドで終了させる

```
taskkill /F /pid 11628
```

②で表示された番号

プロジェクトの設定

- プロジェクトの設定は、application.propertiesで行う
  - application.properties を開く際は、右クリックし、
    次で開く ⇒ Limy プロパティー・エディター を選択する



application.propertiesの記述例

```
# ポート番号、コンテキストルート
server.port=8888
server.servlet.context-path=/practice

# DevToolsによるTomcatの自動再起動を無効にする
spring.devtools.restart.enabled=false
```

行頭に # を付けるとコメント行になる

http://localhost:8888/practice
がトップページのURLになる

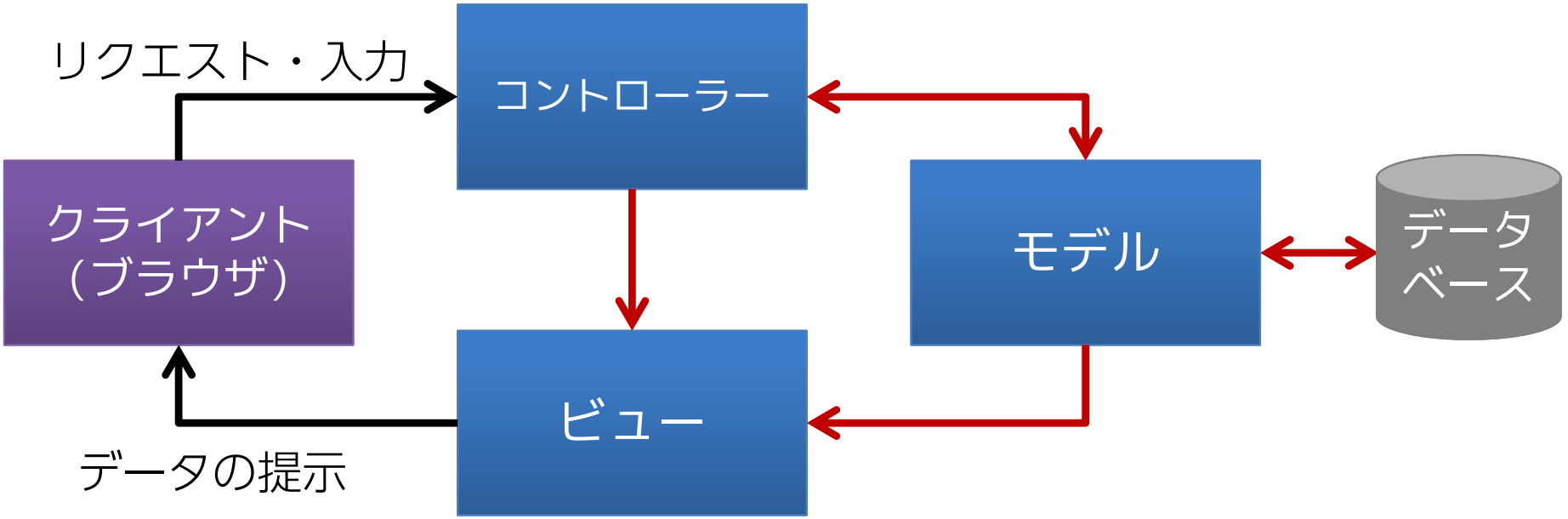
練習問題

- 練習04-1

Spring MVCの概要

MVCモデル

- システムをモデル(Model)、ビュー(View)、コントローラー(Controller) の3つの要素に分割して構成するパターンで、Webシステムの開発にも利用される

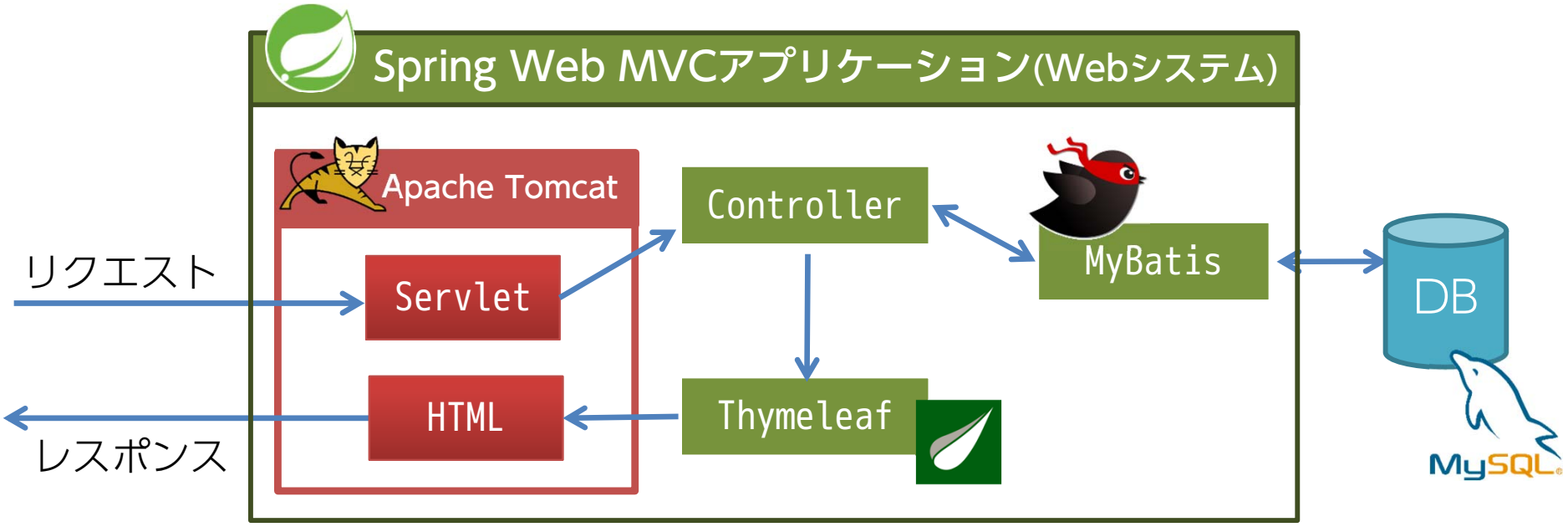


MVCモデル

- コントローラー(Controller)
  - クライアントからの入力を受け付け、内部の制御(モデルやビューの呼び出し)を担当する
- モデル(Model)
  - コントローラーからの指示を受け、データの処理を行う
  - データベースとの連携やコントローラー⇒ビュー間でのデータの受け渡し等に利用される
- ビュー(View)
  - 画面を生成し、クライアントへデータの提示する

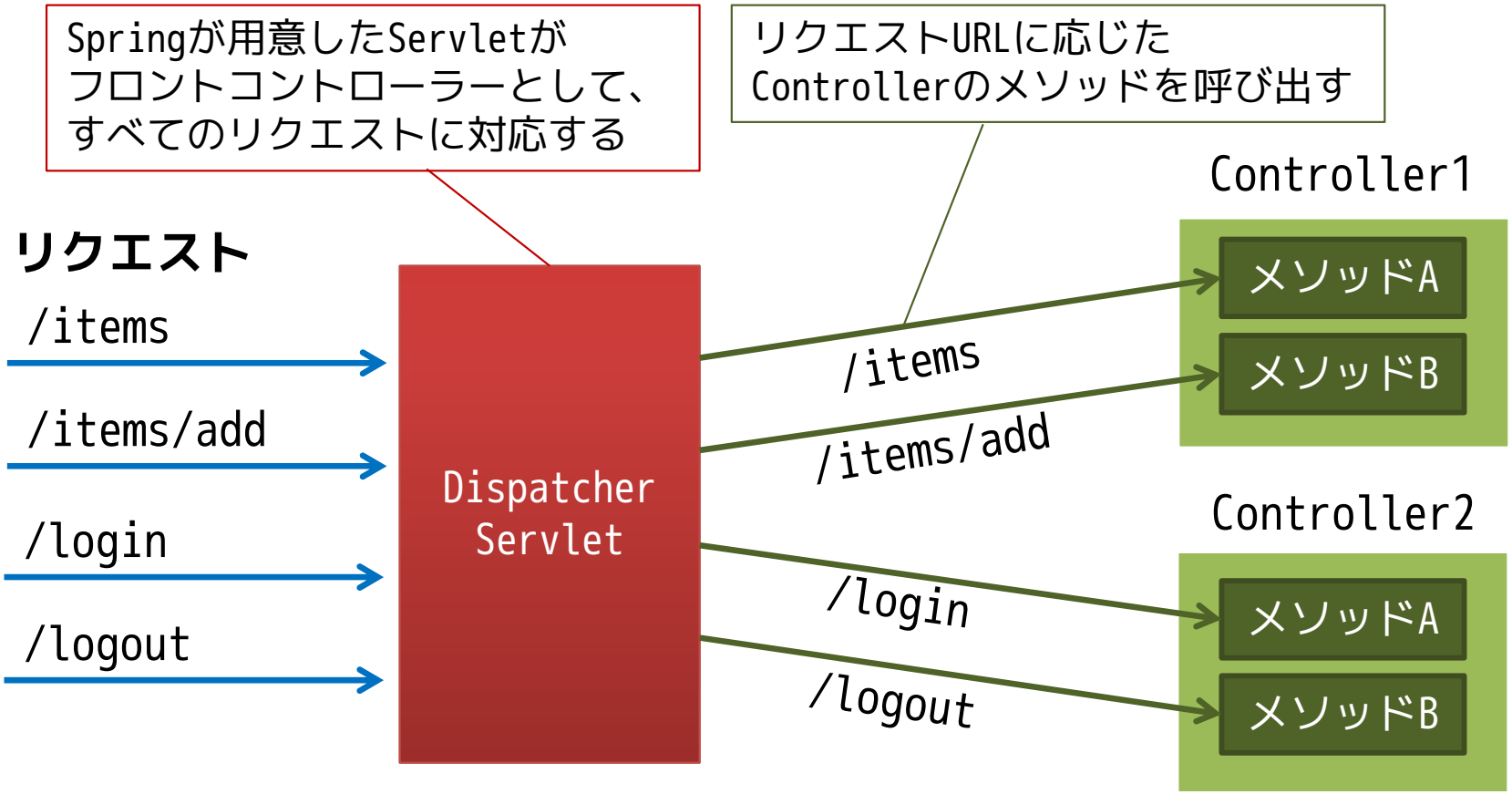
Spring MVC

- Spring Web MVC (Spring MVC)を導入することで、MVC形式のWebシステムを構築することができる
  - 依存関係に「Spring Web」を加えることで導入できる
  - 訓練で開発するシステムでは、以下の図のように、MyBatisがModelを、ThymeleafがViewを担う



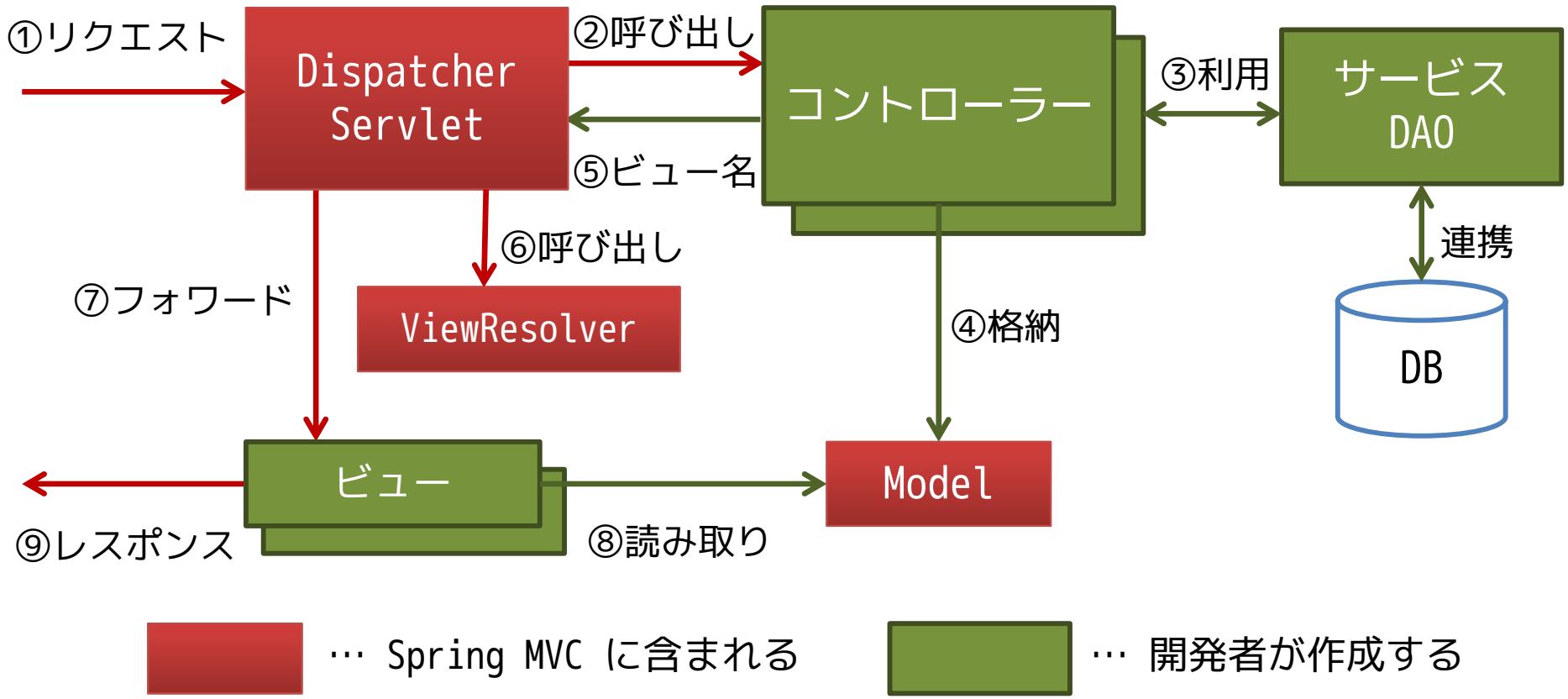
Spring MVCの動作概要

- Spring MVCは１つのServletがすべてのリクエストを引き受けるフロントコントローラー型を採用している



Spring MVC の動作概要

- リクエストからレスポンスまでの流れ



コントローラー

コントローラー

- @Controllerアノテーションを付与されたクラスで、Springによってオブジェクトの生成・管理が行われる
  - このアノテーションが付いていると、フロントコントローラーから呼び出される
- URLに対応するメソッドを実装することができる
  - @RequestMapping, @GetMapping, @PostMapping等のアノテーションで、URLとのマッピングを指定する
  - Modelオブジェクトを経由して、ビュー(Thymeleaf)にデータを渡すことができる
  - 戻り値としてビュー名を指定する

コントローラーの基本構造

- コントローラーの例

```
@Controller
@RequiredArgsConstructor
public class ItemController {

  private final ItemService service;

  @GetMapping("/items")
  public String list(Model model) {
    List<Item> itemList = service.getAllItems();
    model.addAttribute("items", itemList);
    return "list";
  }

}
```

クラスに@Controllerアノテーションを付与する

@GetMappingで、メソッドと
URLとのマッピングを指定する

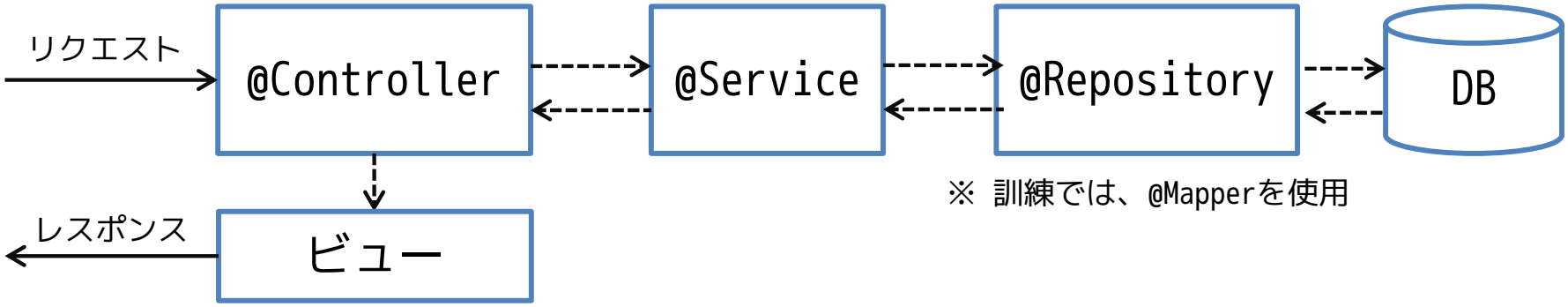
Modelオブジェクトを経由して
テンプレートにデータを渡す

戻り値としてビュー名を指定する

@Controller

- @Componentを継承しており、Springによって生成・管理される
  - @Controller以外にも、以下のような@Componentを継承するアノテーションが存在し、Webシステム開発に利用される

アノテーション	説明
@Controller	コントローラーに付与する
@Service	コントローラーが使用するサービスに付与する
@Repository	データベースと連携するクラス(DAO)に付与する



@Serviceの利用

- コントローラー内で@Serviceアノテーションが付与されたクラスのオブジェクトを利用するには、以下のような手順をとる

① @Serviceアノテーションを付与したクラスを作成する

```
@Service
public class ItemService {
  public String getItemName(int index) {
    String[] items = {"りんご", "みかん", "バナナ"};
    return items[index];
  }
  public int getItemPrice(int index) {
    int[] price = {380, 450, 230};
    return price[index];
  }
}
```

@Serviceの利用

② SpringのDI機能を使い、コントローラーに注入する

```
@Controller
public class ItemController {
  @Autowired
  private ItemService service;
  ...
}
```

この型と互換性がある型(継承元クラスや実装元インターフェースの型)でもよい

または、以下のように記述する

```
@Controller
@RequiredArgsConstructor
public class ItemController {
  private final ItemService service;
  ...
}
```

@GetMapping

- リクエストに対応するメソッドに、アノテーションを付与することで、URLの指定を行う
  - メソッドの戻り値はビュー名(String型)になる

localhost:8080/**items** というURLにリクエストがあった場合、list()メソッドが呼び出される

```
@GetMapping("/items")
public String list(Model model) {

    ...

    return "list";
}
```

listという名前のビュー(list.html等)にフォワード(処理の続きを委託)する

@GetMapping

- { }を使い、配列でURLを指定することで、複数のURLにまとめて対応することができる

localhost:8080/**items** または、
localhost:8080/**items/home** というURLにリクエストがあった場合、list()メソッドが呼び出される

```
@GetMapping({"/items", "/items/home"})
public String list(Model model) {

    ...

    return "list";
}
```

@RequestMapping

- コントローラーに対し、@RequestMappingを付与することで、ベースとなるURLの指定を行うこともできる

```
@Controller
@RequestMapping("/items")
public class ItemController {

    @GetMapping
    public String list() { … }

    @GetMapping("/details")
    public String showDetails { … }

}
```

ベースとなるURL

/items に対応

/items/details に対応

ビューへのデータの受け渡し

- Model#addAttribute()で、ビューにデータを渡すことができる
  - Modelはメソッドの引数として指定することができる

```
@GetMapping("/item/show")
public String show(Model model) {
  model.addAttribute("name", "りんご");
  model.addAttribute("price", 100);
  return "items/show";
}
```

org.springframework.uiパッケージからインポートする

参照時の名前

ビューに渡すデータ

ビュー(Thymeleaf)で表示する場合の記述例 (Thymeleafについては次章で学習)

```
<p>商品名：[[${name}]]</p>
<p>値段：[[${price}]]円</p>
```

メソッドの戻り値

- ビューの名前を文字列として指定する

```
@GetMapping("/item/show")
public String show(Model model) {
  model.addAttribute("name", "りんご");
  model.addAttribute("price", 100);
  return "items/show";
}
```

Thymeleafを利用する場合、
src/main/resources/templates/**items/show.html** に
処理の続き(画面表示用のHTMLを生成する処理)を任せることになる
⇒ 処理の続きを任せることを「**フォワード**」と呼ぶ

練習問題

- 練習04-2