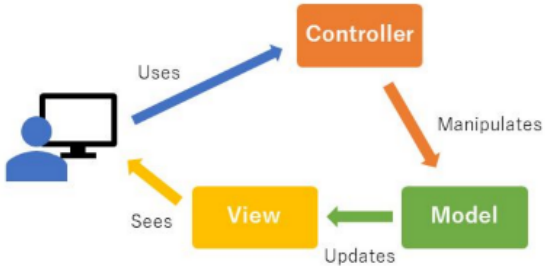
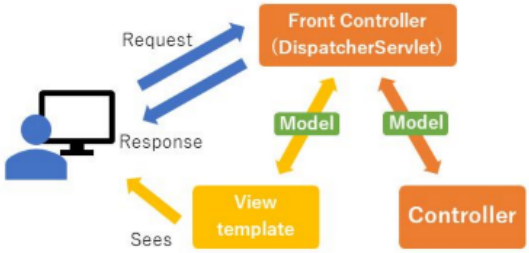
6-4.Spring MVC

Spring MVC

Spring MVCは、Webアプリケーションを作成するためのフレームワークです。
これは、MVC としての機能を提供します。MVC はソフトウェア設計の１つで、機能を「 Model（モデル）」「 View（ ビュー）」「 Controller（ コントローラ）」 の３つに分割して、それらが連携して処理をします。通常、MVCは以下のような図で構成されます。



SpringMVCでは、MVCの中でも「FrontController（フロントコントローラ）パターン」に分類されています。
これは、中央のフロントコントローラが処理の中継を行い、管理するパターンです。
図にすると、以下のような構成になります。



フロントコントローラは、SpringMVCが管理するコントローラです。開発する上では、その処理内容を意識する必要がありません。メインとして意識する必要があるのは、基本的なModel-View-Controllerの部分です。

- **Model**
  モデル は、アプリケーションの動的なデータ構造です。
  Springでは、ごく普通の **Javaオブジェクト（POJO：PlainOldJavaObject）** や **Entity** などが、これに該当します。

- **View（template）**
  ビュー は、図や表などの画面情報を表現します。
  Springでは前ページのような **Thymeleaf** を使った **HTMLテンプレートファイル** が、これに該当します。

- **Controller**
  コントローラ は、入力を受け入れて処理を行います。Springでは **@Controllerアノテーション** を付けたクラスが、これに該当します。

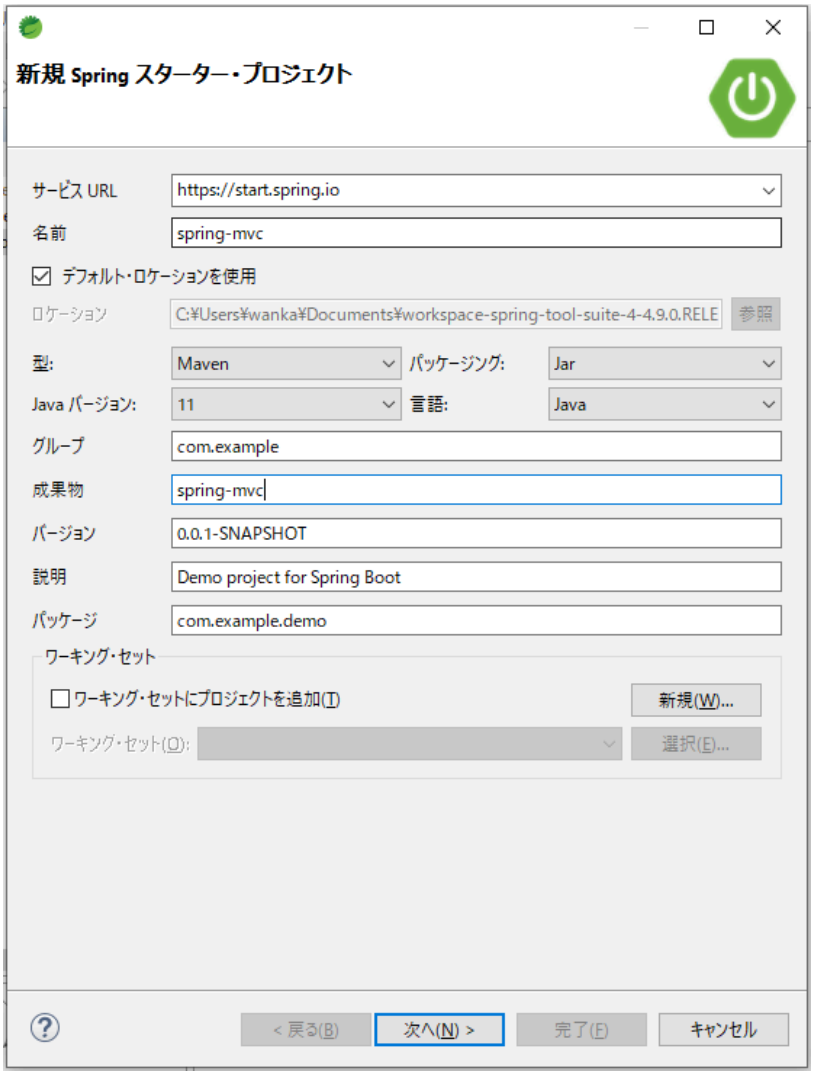
モデルを作成する

モデルを使って、値を送信したらその値を表示するアプリケーションを作成します。

?

プロジェクトを作成

1. プロジェクト名を `spring-mvc` とし、 **次へ** を押下する



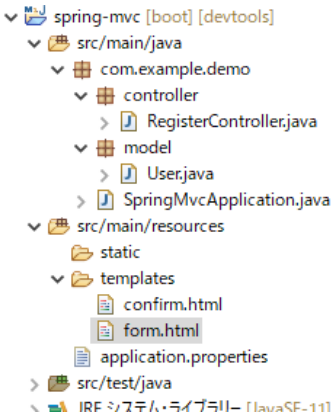
2. 依存関係を下記に設定してください。

   開発ツール→ `Spring Boot DevTools` と `Lombok`
   テンプレートエンジン→ `Thymeleaf`
   Web→ `Spring Web`

   選択したら完了を押下する。

3. 画像を参考に下記ファイルを作成してください

- RegisterController.java
- User.java
- confirm.html
- form.html



モデルクラスの作成

今回は `User.java` をモデルクラスとします。
User.java に下記コードを記述してください。

User.java

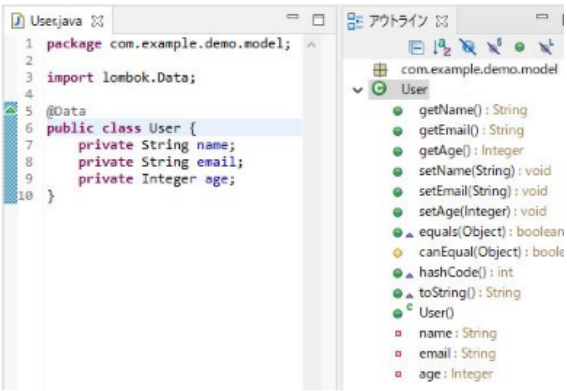
```
package com.example.demo.model;

import lombok.Data;

@Data
public class User {
        private String name;
        private String email;
        private Integer age;
}
```

@Dataとは？

モデルには、 **Getter** や **Setter** が必要になります。
この定型コードを補ってくれるのが、プロジェクト作成時に依存関係で設定した **Lombok** です。
Lombok で生成される定型コードは、右側の[アウトライン]で確認できます。



※アウトラインが表示されていない場合
[ウィンドウ]タブ > ビューの表示 > アウトライン で表示されます。

`@Data` アノテーションは **Lombok** に定義されているアノテーションで、 **Getter** や **Setter** を自動生成してくれる非常に便利なものになります。

もし、 **Lombok** を使わない場合は、以下のように Getter や Setter が必要になります。

```
package com.example.demo.model;
public class User {
    private String name;
    private String email;
    private Integer age;

    public String getName() {
        return this.name;
    }

    public String getEmail() {
        return this.email;
    }

    public Integer getAge() {
        return this.age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

コントローラークラスの作成

RegisterController.javaを、以下のように編集します。

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.example.demo.model.User;

@Controller
public class RegisterController {
        @GetMapping("/form")
        private String readForm(@ModelAttribute User user) {
                return "form";
        }

        @PostMapping("/form")
        private String confirm(@ModelAttribute User user) {
                return "confirm";
        }
}
```

@GetMapping と @PostMapping

@RequestMapping と同様に、アクセスされると該当するアノテーションが付けられているメソッドが処理されます。
@RequestMapping と違うところは、アノテーションの名称で **Get** か **Post** かが判別できるので、アノテーションの引数はアドレスのみの指定で済む点です。

@ModelAttribute

モデル属性に **バインド** します。
バインド とは、日本語で「結びつける」「関連付ける」などの意味です。
引数を付けた場合はその名前でバインドされます。
ちなみに、下記の2つは、同じ動作になります

```
@ModelAttribute User user
@ModelAttribute("user") User user
```

`@ModelAttribute` を使うと、リクエストと一致するものがモデルへ流し込まれます。
これは **データバインディング** と呼ばれ、解析や変換に対処する必要がなくなり、
ModelAndView の **addObject** メソッド などで、モデルに登録する必要がないということです。

View の作成

form.html を、以下のように編集します。

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
        <meta charset="UTF-8">
        <title>Title</title>
</head>
<body>
        <form th:action="@{/form}" th:object="${user}" method="post">
                <label for="name">氏名：</label>
                <input type="text" th:field="*{name}"><br>
                <label for="email"> E-Mail：</label>
                <input type="email" th:field="*{email}"><br>
                <label for="age">年齢：</label>
                <input type="number" th:field="*{age}"><br>
                <button>送信</button>
        </form>
</body>
</html>
```

th:object と th:field

th:object は フォームにバインドするオブジェクトを設定します。
今回は、コントローラ側で用意した **user** オブジェクトを設定しています。 `@ModelAttribute User user`

`*{フィールド名}` は選択変数式で、 **th:object** が付いたタグ内では、オブジェクト名を省略できます。 **th:object** を使用しない場合、以下のように `${オブジェクト名.フィールド名}` を指定する必要があります。

```
<input type="number" th:field="${user.age}">
```

th:field は タグの`id`・`name`・`value` 属性をHTMLに出力する機能です。
以下の2つは、同じ動きをします。

```
<input type="number" th:field="*{age}">
<input type="number" id="age" name="age" th:value="*{age}">
```

---

confirm.html

confirm.htmlを、以下のように編集します。

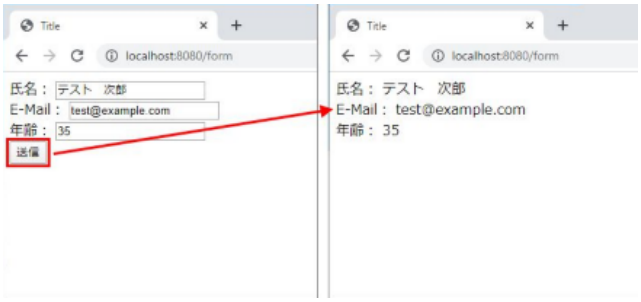
```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
        <meta charset="UTF-8">
        <title>Title</title>
</head>
<body>
        <span th:inline="text">
                氏名：[[${user.name}]]<br>
                E-Mail：[[${user.email}]]<br>
                年齢：[[${user.age}]]<br>
        </span>
</body>
</html>
```

th:inline

タグ内のテキストを展開します。 `[[ ]]` で囲むと、その値を表示できます。

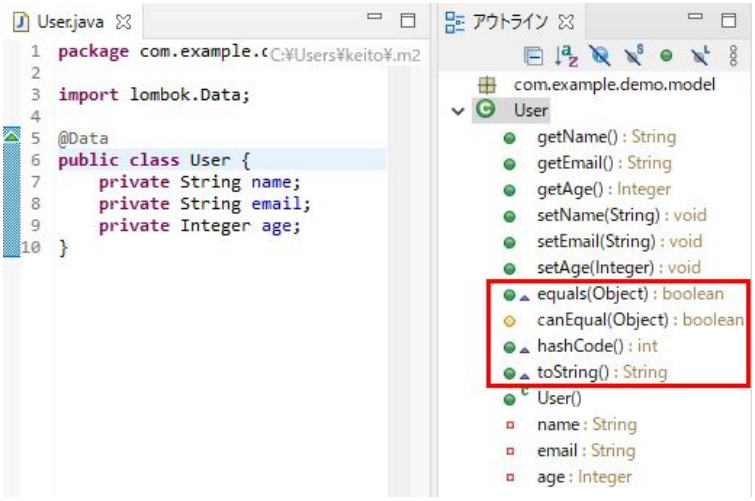
さて、一通り書き写せたところでで実行してみましょう！
プロジェクトを右クリック > [実行] > [Spring Boot アプリケーション] を選択。

ブラウザで http://localhost:8080/form にアクセスします。
表示された画面で、文字を入力。[送信]ボタンを押すと、その値が **confirm.html** に表示されます。

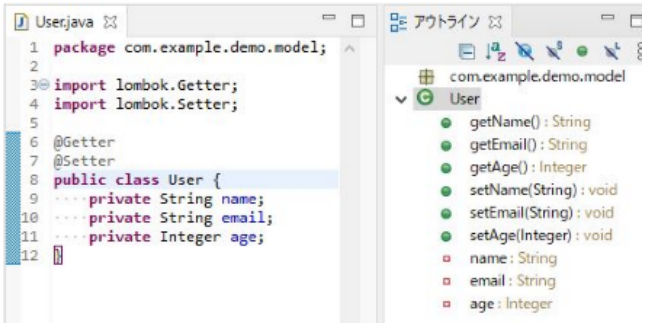


振り返り

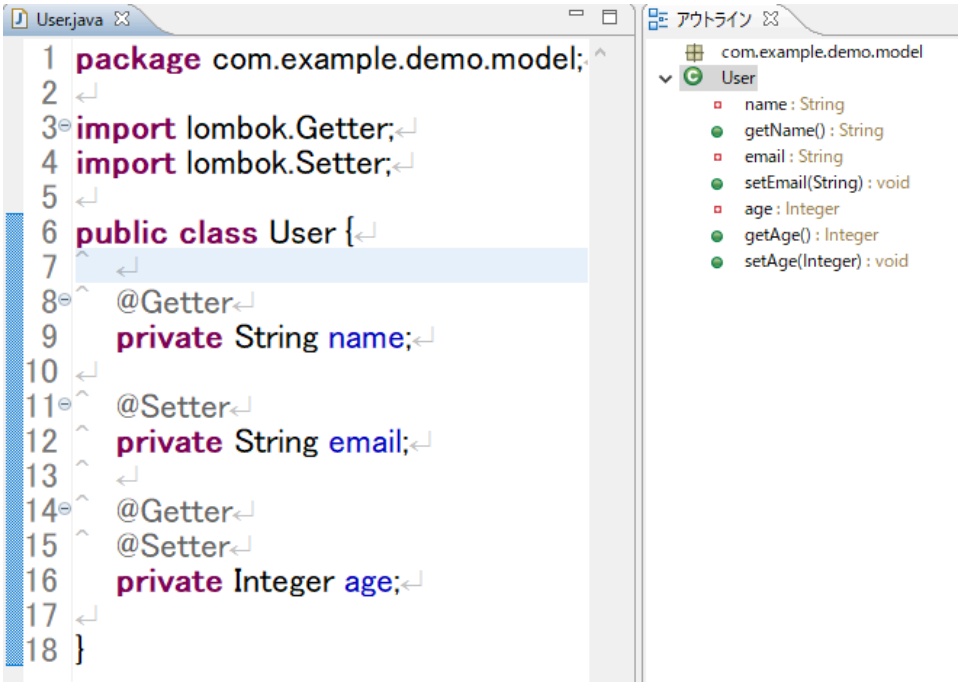
モデルの作成で **Lombok（ロンボック）** を使用しましたが、この機能について振り返ってみます。
Lombok の **@Dataアノテーション** を使うと、 **Getter** や **Setter** 以外にも、 **equals()、canEqueal()、hashcode()、toString()** などのコードも自動生成されます。



もし、 **Getter** と **Setter** だけにしたい場合は、 **@Getter** と **@Setter** アノテーションを使用すると実現できます。



もし、一部のフィールドのみ **Getter** や、 **Setter** を設定したい場合は 該当するフィールドにアノテーションを使用すると実現できます。



以上のように、 **Setter** , **Getter** の自動生成を調整することができ、
臨機応変に対応することが出来ます。

最終更新日時: 2022年 09月 10日(土曜日) 09:32