5-1.MVCモデル

はじめに

MVCモデルとは、 **Model** 、 **View** 、 **Controller** の３つの単語の頭文字から取られたものです。

要素	役割	ファイルクラス
Model (モデル)	アプリケーションの主たる処理（計算処理など）やデータの格納などを行う	java
View (ビュー)	ユーザに対して画面の表示を行う	jsp
Controller (コントローラー)	ユーザからの要求を受け取り、処理の実行をモデルに依頼し、 その結果の表示をビューに依頼する	servlet

実は、Java上級のスキルチェックコードは、このMVCモデルで作られていたのです。
なので、すでに身をもってMVCモデルを体感しています。

> **習ってなくてもできた？**

このMVCモデルというのはシステムを作る上での単なる、 **概念（考え方）** の領域だからです。
この考え方を、 **エンジニア全員** が **共通認識** として持つことで、いわゆる **話が早くなります** 。
極端な例にはなりますが、あなたがModelを作ってと言われれば、Modelの役割の担うコーディングをすればOKなのです。
もっともメジャーな概念が **MVCモデル** ということですね。

この章で、さらにMVCモデルという考え方に慣れて行きましょう！

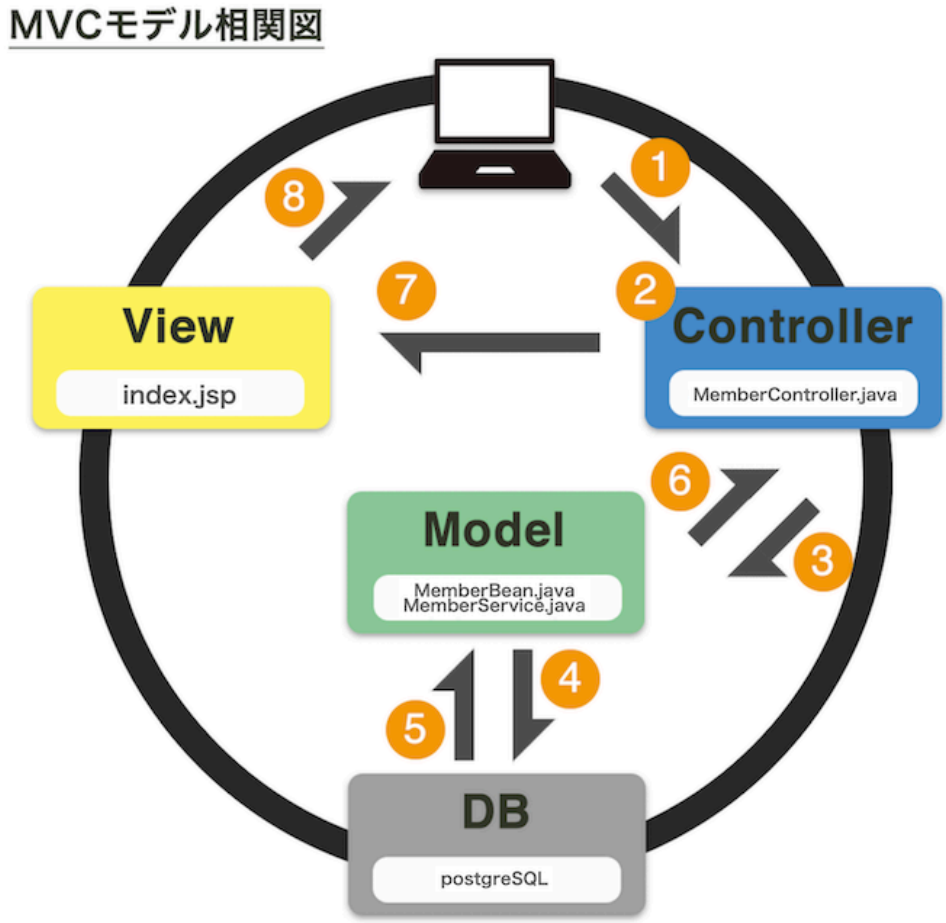
※ 以下、説明に出てくるソースは、スキルチェックを例にしております

Step1 : MVCモデルの概要

基本的な動作、業務上の処理を行う部分を **Model** が受け持ち、
ユーザインタフェース部分を **View** が受け持ち、
これらのやり取りを制御するのが **Controller** です。
と言われてもパッとしないですね。

スキルチェックで使ったソースも含めて、MVCの相関図を以下の画像にまとめてみました。
プログラムの大きな流れも①～⑧まであるので、順を追って確認してみると、さらにイメージがつきやすいと思います。

?



シーケンス	処理内容
①	ユーザーの要求（入力やボタン操作等）
②	①の要求に沿って処理（パラメーターの取得）
③	要求ごとの処理（入力チェックや呼び出す処理の選定）
④	ServiceよりSQLを実行
⑤	実行結果を返却しBeanへ格納
⑥	③の要求に基づいて結果を返す
⑦	jspに結果を返す（③ or ⑥の結果）
⑧	受け取った結果を元に画面へ表示

しっくりきたでしょうか....?
一度、MVCモデルを触っているので、この構造の理解については問題ないと思います。

では、なぜMVCモデルでシステムを構築しなくてはならないのでしょうか？
その大きな理由として、２つのことが考えられます。

> **理由①：生産性の向上**

ModelをViewやControllerから密接な結びつきがないよう完全に切り離すことにより、
異なるWebアプリケーションでも使いまわすことが可能です。
同じようなロジックを、ゼロから記述しなくて済みます。

例えば、３か月費やして完成させた社員情報システム処理があったとします。
他の案件で似たようなものを作ってほしいとお願いされた時、
そのModelを流用し、ちょっと手を加えるだけでクリアできてしまう。なんていうことが可能になります。

> **理由②：メンテナンス性の向上**

MVCモデルは、Model、View、Controllerを分割することで、それぞれの独立性を高めます。
独立性が高まれば、修正や追加機能といった要求に対して容易かつ柔軟に対応することが可能です。
そのため、メンテナンス性の向上が図れます。

概要としては、以上になります。
それでは、Model、View、Controllerそれぞれについて、詳しく見て行きましょう。

Step2 : Model

`MemberBean` として一つファイルを設けたかと思います。
実はファイル名を `Bean` としたことにも意味があります。

`Bean` とは、ViewやControllerとは関係のないものであり、Java言語で作成された普通のクラスのことです。
それぞれの実際の部品化（コンポーネント化）したもの1個単位を単にBeanと呼ぶこともあります。
そのような名前の由来は、Javaのコーヒー豆という意味でBean（豆）という言葉を掛け合わせたものです。

総称して `JavaBeans` と呼んでいます。

スキルチェックで使ったプロジェクトの`MemberBean` の中身は以下のようになっていました。

```
public class MemberBean {

    private String Id;
    private String PassWord;
    private String Name;
    private String Comment;
    private String Login_Time;

    public void setId(String Id) {
        this.Id = Id;
    }

    public String getId() {
        return Id;
    }

    public void setPassWord(String PassWord) {
        this.PassWord = PassWord;
    }

    public String getPassWord() {
        return PassWord;
    }

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

    public String getName() {
        return Name;
    }

    public void setLogin_Time(String Login_Time) {
        this.Login_Time = Login_Time;
    }

    public String getLogin_Time() {
        return Login_Time;
    }

    public void setComment(String Comment) {
        this.Comment = Comment;
    }

    public String getComment() {
        return Comment;
    }
}
```

> **解説**

`JavaBeans` の仕様は、あらかじめ規約が定められています。
例えば、JSPから利用されるJavaBeans（Bean）として成立するためには、
外部からアクセスされるフィールドに対して、アクセサメソッドを持つことです。

アクセサメソッドとは、フィールドの値に外部からアクセスし、格納や取得を行うためのメソッドのことを指します。

セッターメソッド

フィールドの値を設定するためのアクセサメソッド、「setフィールド名」という形で記述します。
今回だと、setIdメソッド等。

ゲッターメソッド

フィールドの値を取得するためのアクセサメソッド、「getフィールド名」という形で記述します。
今回だと、getIdメソッド等。

MemberBeanクラス は、 Id PassWord Name Comment Login_Time というフィールドがありますが、
それぞれのフィールドには値を設定または取得するための
setId() 、 getId() というアクセサメソッドを定義しています。
このようにして、セッター、ゲッターを設けることが、 JavaBeans（Bean） の
決まりごとの一つとして定められています。

ビジネスロジック

MVCの中には、ビジネスロジックという部類があります。
スキルチェックだと、 MemberService がそれに該当します。

ビジネスロジック とは、実際の **業務にあたる部分** です。
MemberService の場合、
「特定のユーザを探し、ヒットすればユーザー情報を提供し、その時間を記録する」という処理が流れているわけですが、
これは本来、人が仕事として論理的に（業務のルールに従って）裁かなくてはなりません。

このようにして、 **人の業務を担うロジック** ことを **ビジネスロジック** と呼んでいます。
ファイル名に関しても、 ○○○○○Service とすることが基本です。

以下の解説は、スキルチェックで作成したMemberService.javaを見ながらお読み下さい。

解説

MemberServiceクラスは、 search() メソッドにより、
ユーザID（ id ） と **パスワード（ password ）** を条件にして、
合致するものが、任意のデータベースのテーブルにあるかないかを判別する処理を行っています。

合致するものがあれば MemberData にテーブルの値を格納し、
なければ、 MemberData を null で返す。というようなロジックになっています。

このようなクラス（ビジネスロジック）も **Model** という枠の中に入ってきます。
理由としては、 **ビジネスロジックとして再利用可能な部品** なので、
ViewやControllerとは関係のないもの、つまり JavaBeans という位置づけになるからです。

補足

- null **による初期化**
  Bean のような、 **インスタンスを生成する必要があるクラスオブジェクトを最終的にViewへ返すようなロジック** においては、
  Beanを null で初期化しておくことが多いです。
  また、 null で初期化することにより、プログラム実行時に余計なメモリを使用することを避けています。
  （検索結果として表示データが無い場合に、インスタンスを生成していてもメモリを無駄に消費することを回避する意図があります。

加えて、今回は社員情報という1つの情報を丸っとViewへ返す作りとしているため、
View側で最低限の表示ロジックを組むにあたっての判断材料（ `if (MemberBean != null)` の部分）としています。
（※詳細は **Step3 : View** にて解説します。

Step3 : View

以前の章で`JSP` について学びましたが、
MVCモデルにおいての `JSP` の役割は、 **ブラウザに表示するためのもの** です。
そのため、JSPには表示するための簡単なロジックが少しあるにせよ、
制御ロジックやビジネスロジックなどは無いようにします。

以下の解説はスキルチェックで作成した`index.jsp`を見ながらお読み下さい。

> **解説**

まず、`MemberController` からの指示を、`getAttribute`を使用して受け取ります。

`MemberController` から渡ってきたデータが `MemberBean` に値が格納されるわけですが、
その値を使って、条件分岐します。条件は二つです。

```
if (MemberBean != null) {
    //省略

} else { %>
    IDもしくはパスワードが間違ってます
<% }
```

> **条件1 : nullではない**

nullではない つまり、検索条件に合致したデータが存在し、テーブルのデータを取得できているはずなので、
getメソッドを用いて取得データの値を表示します。

> **条件2 : nullだった場合**

nullだった つまり、テーブルに条件と合致したデータがなかったので、
その旨を表す文言を表示する。
今回の場合は、 `IDもしくはパスワードが間違ってます` としてます。

このようにして、MVC構造においての `JSP` はとってもシンプルなものになります。

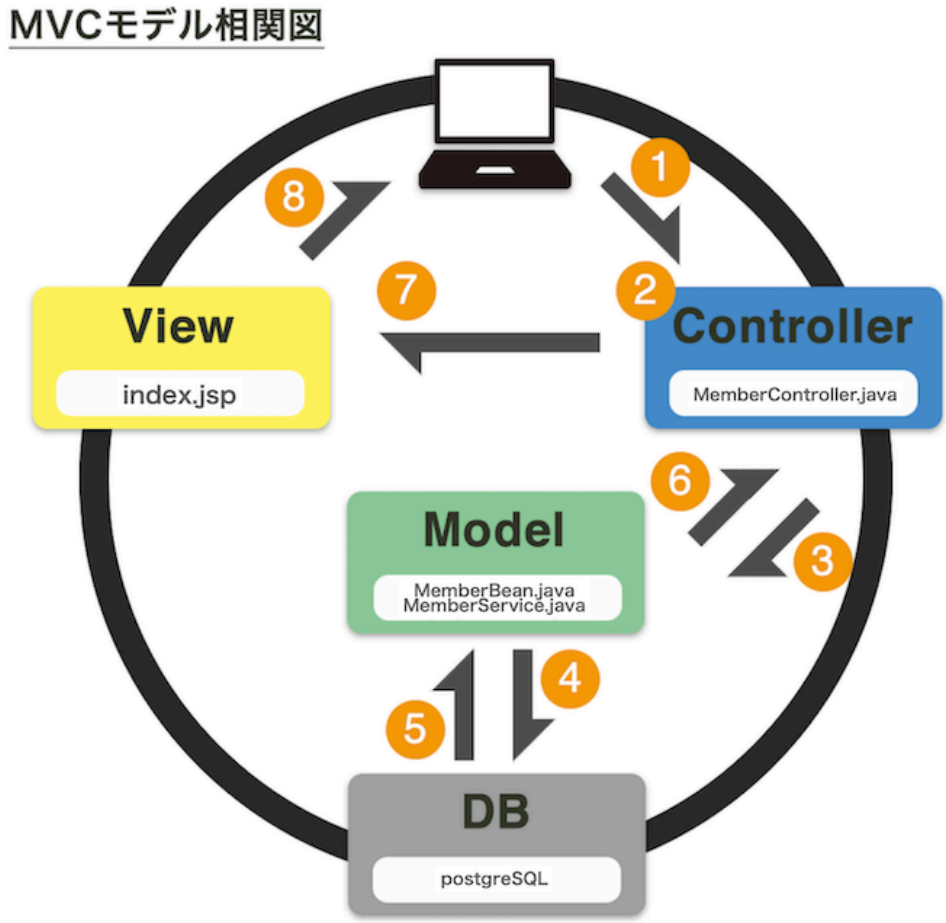
Step4 : Controller

最後は、 **Controller** についてです。

以下の解説は、スキルチェックで作成した`MemberService.java`を見ながらお読み下さい。

> **解説**

図とシーケンス表をもとに、順を追ってみていきましょう。



シーケンス	処理内容
①	ユーザーの要求（入力やボタン操作等）
②	①の要求に沿って処理（パラメーターの取得）
③	要求ごとの処理（入力チェックや呼び出す処理の選定）
④	ServiceよりSQLを実行
⑤	実行結果を返却しBeanへ格納
⑥	③の要求に基づいて結果を返す
⑦	jspに結果を返す（③ or ⑥の結果）
⑧	受け取った結果を元に画面へ表示

> **シーケンス1：ユーザーの要求（入力やボタン操作等）**

ユーザーが画面にて、任意の情報を入力する

> **シーケンス2：①の要求に沿って処理**

`MemberController` が任意の情報をキャッチ

ユーザーからの入力情報を、`getParameter`を使用してパラメーターとして受け取ります。

シーケンス3：要求ごとの処理（入力チェックや呼び出す処理の選定）

`MemberController` が 任意のユーザー情報をもとに社員情報の照会をする必要があるため、`MemberService` の `search` メソッドを呼び出す。

シーケンス4：ServiceよりSQLを実行

ビジネスロジックである、 `MemberService` が仕事を開始する。
まずは、SQLに問い合わせる。
(問題④～⑥　SQLの設定と実行)

シーケンス5：実行結果を返却しBeanへ格納

ヒットしたので、SQLの値を `MemberData` に見繕って `MemberController` に返してあげる。

`search` メソッドの戻り値が `MemberData` というのもポイントです。

```
//問題⑤にてMemberDataにデータ格納
  ↓

//searchメソッドの戻り値
return MemberData;
```

シーケンス6：③の要求に基づいて結果を返す

`MemberController.java`に戻ってきます。

`searchメソッド` の戻り値が `MemberData` (SQLの結果が格納されている変数)なので、
それを `MemberBean`に格納します。

シーケンス7：jspに結果を返す（③ or ⑥の結果）

JSP結果を返す値を、`setAttribute`により準備。
キーは `"MemberBean"` として、
値は、 SQLの値が格納されている `MemberBean` となる。

渡したいJSPを指定する。

```
RequestDispatcher dispatcher = context.getRequestDispatcher("/index.jsp");
```

シーケンス8：受け取った結果を元に画面へ表示

よって、index.jspが受け取った値を表示することが可能になる。

このように、MVCモデルにおいての `Controller` は、 **ユーザー** 、 **Model**、 **View** の仲介役になっています。
まさに、システムを **Controller** しているわけですね。

Step5：総括

基本はMVC

MVCモデルのつながりについて一通りは説明してきましたが、実際にプログラミングしていくと大変難しいと思います。
また、最初のTOPページがhtmlであったり、JSPであったりすると少し難易度が上がりますし、
複数ページが遷移する場合などは、とても難しく感じると思われます。

しかし、複数ページが遷移する場合でも基本的な **MVCモデルのつながりは同じ** です。
プログラムの流れ を理解し、それぞれの **役割を把握する** ことで、
どこにどんな処理が必要なのか が見えてきます。

課題

提出課題はありませんので、一通り学習が終わったら次の章に進んで下さい。

最終更新日時: 2022年 10月 10日(月曜日) 15:01