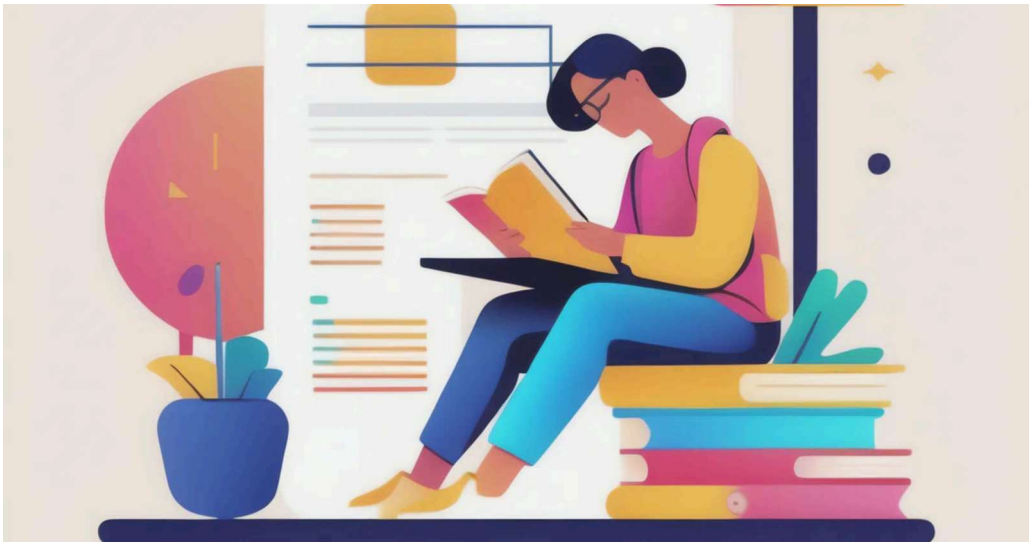

MVCモデルによるWebアプリケーション開発（No６.積立投資）-EE8

office・M
2024年1月12日 10:20

¥1,000 •••

JavaEE8（JSP・Servlet）の環境でWebアプリケーションの開発方法を学ぶ講座をシリーズで提供しています。今回は部品として作成されたクラスファイル（積立投資のシミュレーションをhtml形式で出力するBean）を活用する方法を学びます。チーム開発での役割分担を想定しています。

2024年9月よりECLIPSEのバージョンを最新版（Version: 2024-06 (4.32.0)）に変更しました。

目次

3．積立・利率・期間の入力画面のJSPファイルを作成します

4．シミュレーション結果の出力画面のJSPファイルを作成します

実行確認

1．InternalServerErrorとなります

2．実行用Beanを配置します

3．再度実行します

ソースコード例と提供Bean

1．AccumulationSimBean.class

2．AccumulationServlet.java

3．accumulationForm.jsp

4．accumulationResult.jsp

連絡先

開発概要

JavaEE8アプリケーション・サーバ（Tomcat）を利用して開発を行うということを前提に実装を進めていきます。開発ツールには統合開発環境のEclipseを用いることにします。

題材として積立投資のシミュレーションを行うWebアプリケーションを作成します。入力画面に積立金額、想定利回り、積立総月数を入力すると、運用収益のシミュレーション結果を表示する画面に遷移します。運用収益を計算するBeanは提供します。このBeanは、参照先に示す「金融庁　資産運用シミュレーション」や「カシオの計算サイト」と同じ結果を返すようオリジナルで作成したものです。

（参照先）
金融庁　資産運用シミュレーション
※金融庁サイトでは結果が1円ずれますが許容範囲です
https://www.fsa.go.jp/policy/nisa2/moneyplan_sim/index.html

（参照先）
カシオの計算サイト
※実行結果とぴったり同じになります
https://keisan.casio.jp/exec/system/1254841870

JavaEE8環境ですので使用言語は当然Java（JSP・Servlet）となり、設計技法にはMVCモデルを使います。MVCモデルとはWebアプリケーションの構成をModel（業務ロジック）ーView（表示）ー

Controller（制御）に分割して設計する技法です。３つのモデルに役割分担することで部品化が促され、ひいてはチーム開発に貢献します。

今回、業務ロジックであるBeanは作成しません。他のチームメンバーが作成したという想定で提供されたBeanを活用することにします。

実装していく過程でEclipseの利用方法が不明な場合は、別途メール等でお知らせください。この講座では、作成済みの仕様と設計書から「**JavaEE8環境でのMVCモデルの実装の仕組みを学ぶ**」ことを目的とします。

開発方針

Webアプリケーションの開発環境には図１のようにEclipse2024を用います。

図1．開発環境 Eclipse 2024

Spring等のフレームワークは使用しません。これは、MVCモデルにおけるhttpプロトコルの処理の実装を直に学んでいただきたいためです。

開発に必要な仕様と最小限の設計ドキュメントは提示します。この情報をもとに、まずは、自分で試行錯誤しながら実装してみてください。

設計仕様

【積立投資シミュレーションの仕様】
以下の条件を実現するWebアプリケーションの作成を行います。
（条件）
・urlを指定してサイトにアクセスすると３つの入力欄（積立、年率、期間）と「シミュレーションボタン」が配置された画面を表示します。
・積立とは毎月積立てする金額、年率とは１年間の想定利回り、期間は積立の総月数を表しています。

・積立は円単位で、年率は%の実数値を小数点以下2桁で、期間は月単位で入力します。
・「シミュレーションボタン」をクリックすると画面が遷移して運用収益の履歴を表示します。
・運用収益は月複利で四捨五入による元加を行い非課税で算出しています。

【Model（処理ロジック）】
AccumulationSimBean.classが提供されます。
（仕様）
・引数で積立金額(円)と想定利回り(実数値)と積立総月数(月)を受け取る
・月毎の運用収益を計算しhtml形式で変数に保持する
・運用収益の変数はメソッドで外部に公開される

外部設計

接続urlは**http://localhost:8080/money/Accumulation**とします。よってEclipseの動的Webプロジェクトの名前は**money**となり、サーブレットのurlパターンは**Accumulation**となります。

Eclipseのメニューバーより
ファイル→新規→動的Webプロジェクト→「money」プロジェクトを作成する → 図1.1の内容で設定する

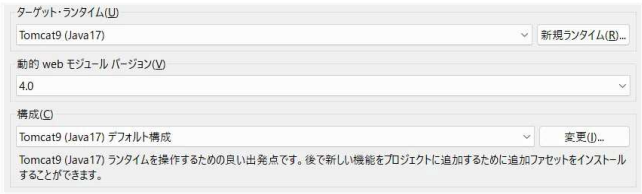

図1.1 動的Webプロジェクトの設定

※作成済みであればこの処理は必要ありません

---

urlにアクセスすると図２のような、起動画面が表示されます。積立(積立金額)と年率(想定利回り)と期間(積立総月数)を入力して「シミュレーションボタン」をクリックすると、図３のように運用収益のシミュレーション結果の表示画面に遷移します。

図２．起動画面

図３．運用収益のシミュレーション画面

最終的には「マネーシミュレーションWebアプリ」のメニュー画面からクリッカブルマップを利用して起動できるように構成します。

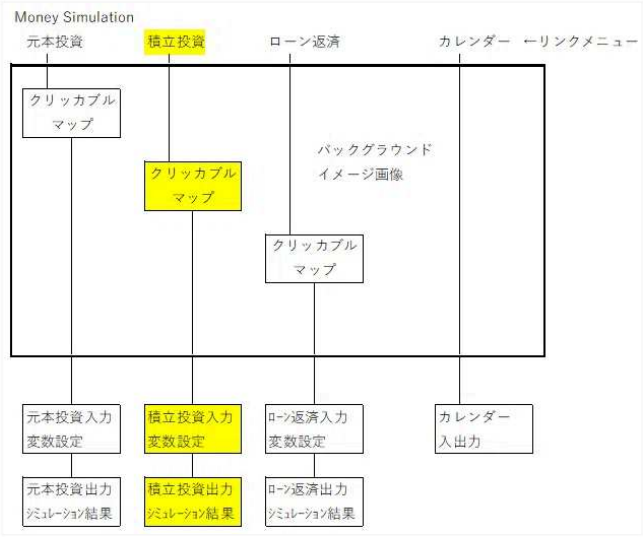

図3.1 マネーシミュレーションWebアプリ概念図

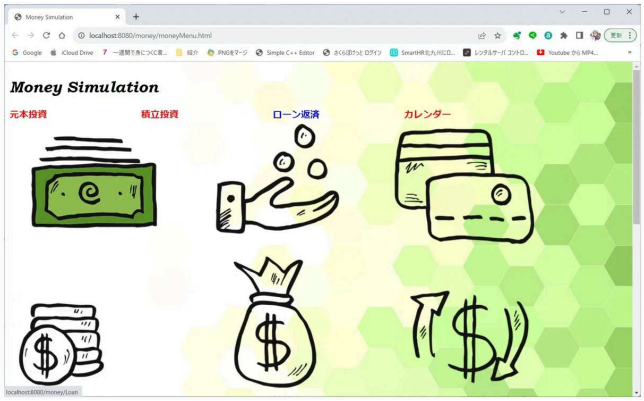

図3.2 マネーシミュレーションWebアプリの例

内部設計

クラス連携図は図４のようになります。

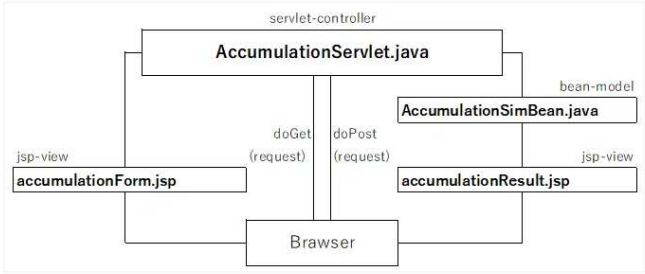

図４．MVCモデル図

提供される積立投資計算ロジック（AccumulationSimBean.class）の使い方とクラス図は以下のようになります。

（使い方）
・AccumulationSimBeanクラスを**積立金額**と**想定利回り**と**積立期間**の３つの引数をもつコンストラクタで実体化します。
・simulationメソッドを実行することでsimフィールドに運用収益の履歴がhtml文字列で表組されます。
・利用側のクラスはgetSim()メソッドで運用収益の履歴を取得します。

（クラス図）

図4.1 AccumulationSimBeanクラス図

実装手順

１．積立投資計算ロジック用JavaBeansクラスを使用する準備をします

(1)クラスファイルの準備をします
以下Windows環境を想定しています。
事前に「提供Beanフォルダ」を作成しておきます（例　c:￥提供Bean）。
提供Beanフォルダ内に**AccumulationSimBean.class**を保存しておきます。
その際**パッケージの階層**に従ってください。
（例　c:￥提供Bean￥jp￥ict￥aso￥model￥AccumulationSimBean.class）

(2)AccumulationSimBean.classをEclipseのビルド・パスに追加します

Eclipseパッケージ・エクスプローラより
moneyプロジェクトを右クリック→ビルド・パス→ビルド・パスの構成→「ライブラリ」タブ→「クラスパス」クリック→外部クラス・フォルダーの追加→「提供Bean」を指定します→最後に「適用して閉じる」をクリック

※図５のように一度設定されていれば再度設定する必要はありません。

図５．Javaのビルドパス追加画面

２．リクエストコントロール用Servletクラスを作成します

Eclipseパッケージ・エクスプローラより
moneyプロジェクトを右クリック→新規→その他→Web→サーブレット
→以下の内容で作成する

・AccumulationServlet.java
　　パッケージ：jp.ict.aso.controller
　　クラス名：AccumulationServlet
　　ソースコード：考えましょう！　※アノテーションは**/Accumulation**

３．積立・利率・期間の入力画面のJSPファイルを作成します

Eclipseパッケージ・エクスプローラより
moneyプロジェクトを右クリック→新規→その他→Web→JSPファイル
→以下の内容で作成する

・accumulationForm.jsp
　　保存場所：money/src/main/webapp/**WEB-INF/jsp** ← **注意！**
　　ファイル名：accumulationForm.jsp
　　ソースコード：考えましょう！

４．シミュレーション結果の出力画面のJSPファイルを作成します

Eclipseパッケージ・エクスプローラより
moneyプロジェクトを右クリック→新規→その他→Web→JSPファイル
→以下の内容で作成する

・accumulationResult.jsp
　保存場所：money/src/main/webapp/**WEB-INF/jsp ← 注意！**
　ファイル名：accumulationResult.jsp
　ソースコード：考えましょう！

実行確認

サーブレットクラス（AccumulationServlet.java）を実行します。しかし、、、

１．InternalServerErrorとなります

実行用Beanのデプロイ（配置）が必要です。この設定を行わないと図６のような実行時エラーになります。

Eclipseパッケージ・エクスプローラより
AccumulationServlet.javaを右クリック→実行→サーバーで実行
→図６のように**エラー**となる

図６．実行時エラー画面

２．実行用Beanを配置します

AccumulationSimBean.classをEclipseの実行時のクラスパスに追加します。

Eclipse**プロジェクト・エクスプローラ**より
（**パッケージ・エクスプローラではありません！**）
moneyプロジェクトを展開→buildを展開→classesを展開→jpを展開→ictを展開→asoを展開→modelがなければ作成

図７のようにmodelフォルダの位置へAccumulationSimBean.classをドラッグ＆ドロップすることで実行時クラスパスにBeanが追加されます。
※Eclipseを終了させると、再度追加が必要な場合があります。

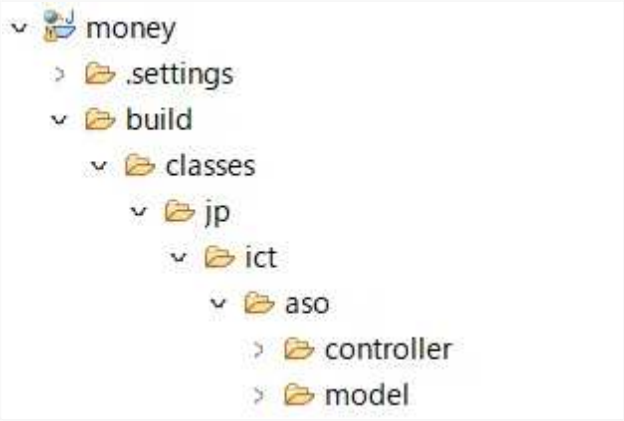

図７．実行時クラスパスの位置

３．再度実行します

Tomcatサーバを再起動したのち、再度サーブレットクラス（AccumulationServlet.java）を実行します。

Eclipseパッケージ・エクスプローラより
AccumulationServlet.javaを右クリック→実行→サーバーで実行
→図８のように実行される

図８．起動画面

ソースコード例と提供Bean

以下に各プログラムのソースコードの例（本文内では「考えましょう！」になっている部分）を示しますので、実装の参考にしてください。

また、提供BeanとしてAccumulationSImBean.classを次のセクションに置きますので、ダウンロードして使用してください。

１．AccumulationSimBean.class

２．AccumulationServlet.java

```java
package jp.ict.aso.controller;

import java.io.IOException;

@WebServlet("/Accumulation")
public class AccumulationServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

    // フォワード
      RequestDispatcher dispatcher =
        request.getRequestDispatcher
            ("/WEB-INF/jsp/accumulationForm.jsp");
      dispatcher.forward(request, response);
  }
  protected void doPost(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

    // リクエストパラメータの取得
    request.setCharacterEncoding("UTF-8");
    String stumitate = request.getParameter("tumitate");
    String sriritu = request.getParameter("riritu");
    String skikan = request.getParameter("kikan");

    int tumitate = Integer.parseInt(stumitate);
    double riritu = Double.parseDouble(sriritu);
    int kikan = Integer.parseInt(skikan);

```

図９．AccumulationServlet.java（１）

```
41      // 入力値をプロパティに設定
42      AccumulationSimBean asb = new AccumulationSimBean(tumitate,ririt,kikan);
43      asb.simulation();
44
45      // リクエストスコープに保存
46      request.setAttribute("result", asb);
47
48      // フォワード
49      RequestDispatcher dispatcher =
50          request.getRequestDispatcher
51              ("/WEB-INF/jsp/accumulationResult.jsp");
52      dispatcher.forward(request, response);
53    }
54 }
```

図10．AccumulationServlet.java（２）

３．accumulationForm.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8">
6  <title>投資</title>
7  </head>
8  <BODY  background="images/money_back1920.jpg" >
9
10 <h1>投資シミュレーション</h1>
11 <h2>金融庁　積立投資</h2>
12 <h3>利息元加処理：月複利
13 元加方式：四捨五入
14 課税処理：非課税</h3>
15 <hr>
16
17         <form method="POST" action="Accumulation">
18 <p>積立：<input type="number" name="tumitate" value=50000> (円) </p>
19 <p>年率：<input type="number" name="ririt" step=0.001 value=0.03> (%の実数値) </p>
20 <p>期間：<input type="number" name="kikan" value=12> (月) </p>
21 <p>※積立とは毎月の積立金額</p><p>※年率とは想定利回り</p>
22 <p>期間とは積立総月数</p>
23
24 <input type="submit" value="シミュレーション">
25
26         </form>
27
28 </body>
29 </html>
```

図11．accumulationForm.jsp

４．accumulationResult.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8" import="jp.ict.aso.model.*" %>
3  <%
4  AccumulationSimBean asb=(AccumulationSimBean)request.getAttribute("result");
5  %>
6  <!DOCTYPE html>
7  <html>
8  <head>
9  <meta charset="UTF-8">
10 <title>投資</title>
11 </head>
12 <BODY  background="images/money_back1920.jpg" >
13
14 <h1>投資シミュレーション</h1>
15 <h2>金融庁　積立投資</h2>
16 <h3>利息元加処理：月複利
17 元加方式：四捨五入
18 課税処理：非課税</h3>
19 <hr>
20
21 <p>
22 シミュレーション結果：<%=asb.getSim() %><br>
23 </p>
24
25 <a href="Accumulation">戻る</a>
26 </body>
27 </html>
```

図12．accumulationResult.jsp

連絡先

質問や不明な点がある場合は以下のアドレスにメールをください。24時間以内に返信いたします。

info@slowlife.halfmoon.jp