4-3-3.簡単なサーブレットプログラム・web.xml・動作確認

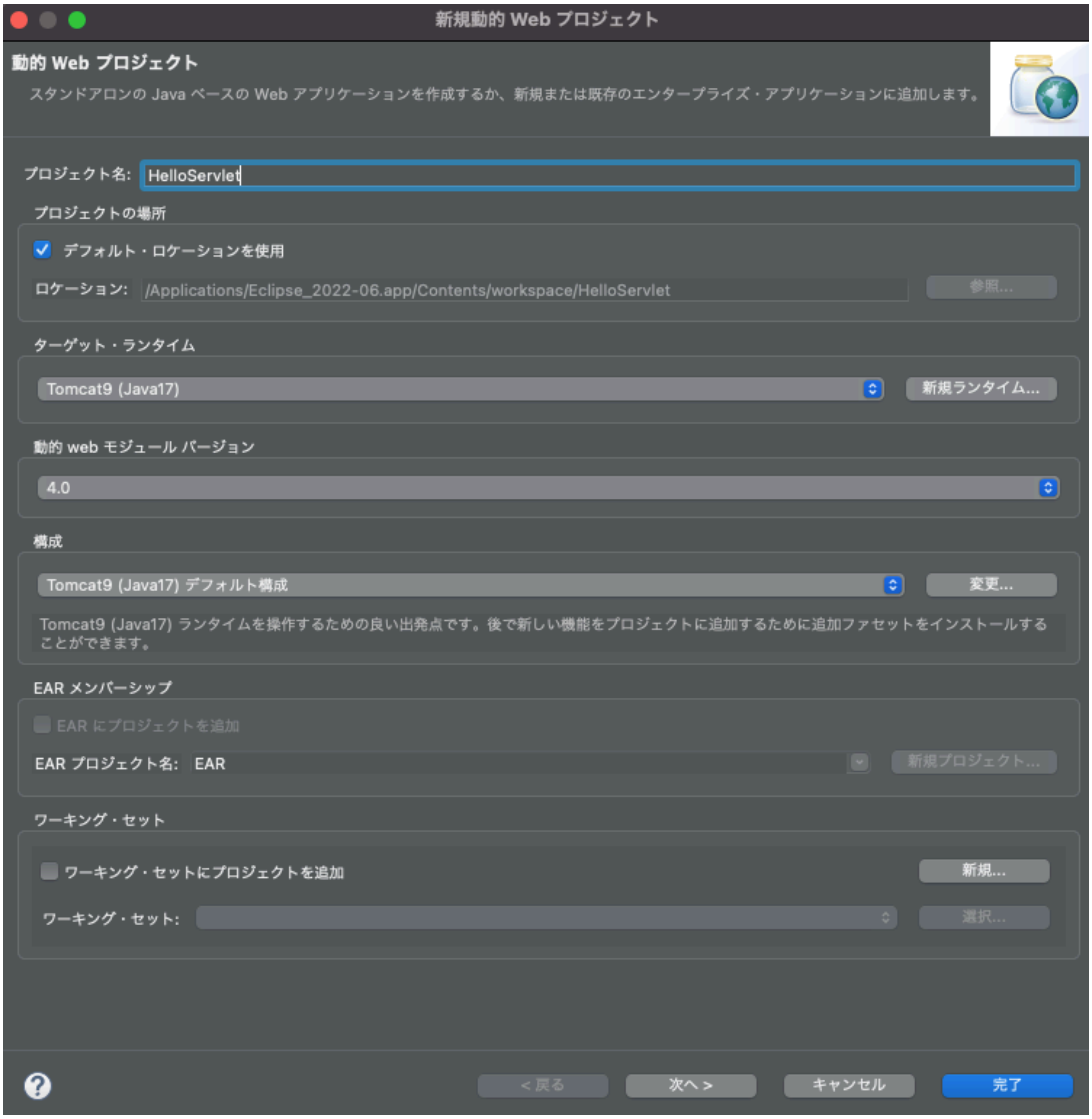
■簡単なサーブレットプログラム

まずは手始めに「HelloServlet」と表示するプログラムを作成してみましょう！
そこそこの手順を踏みますが、覚えてしまえば簡単です！

Step1: プロジェクトの作成

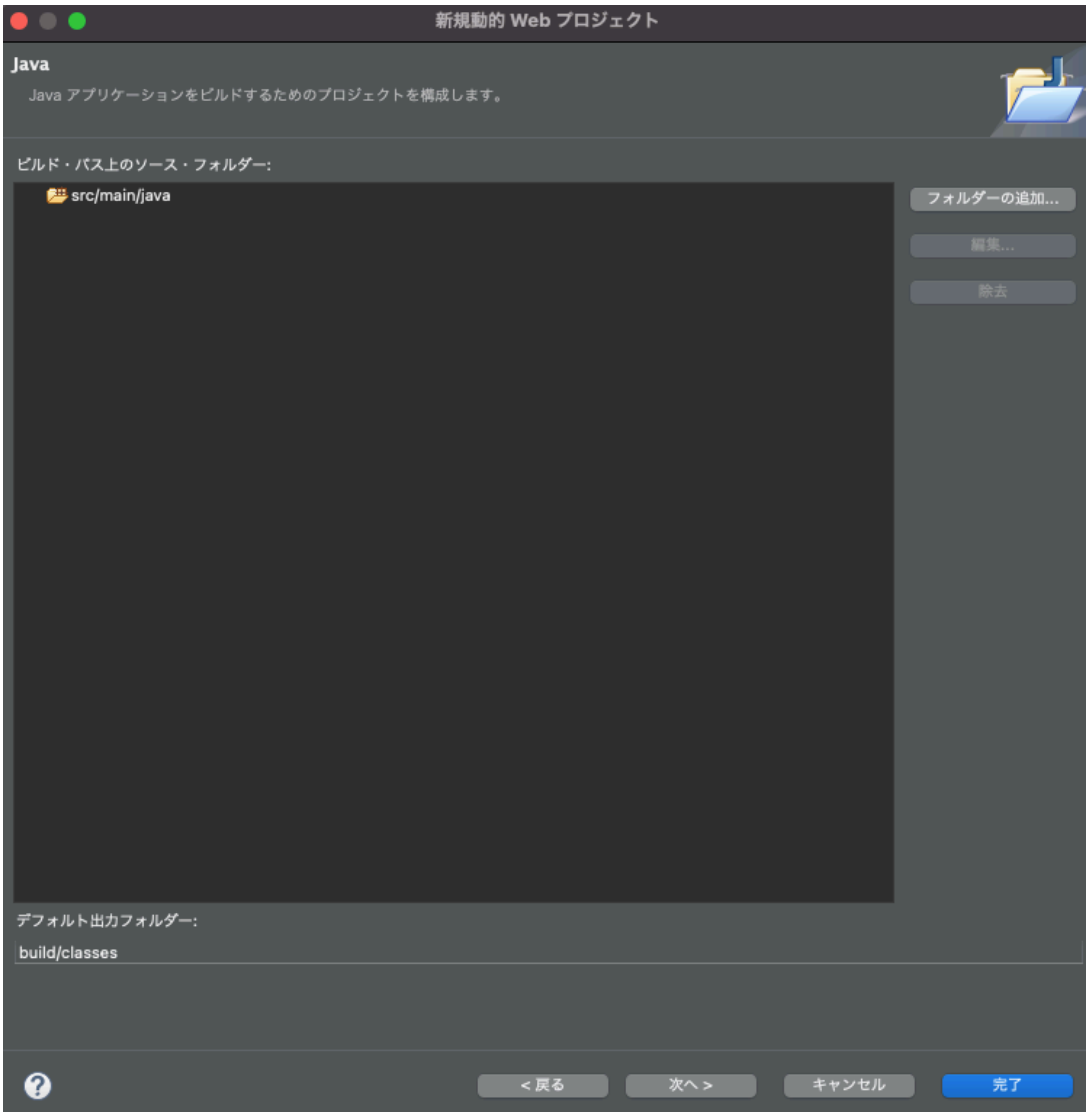
1.メニューの《ファイル》→《新規》→《その他》→《動的webプロジェクト》を選択します。

プロジェクト名を「HelloServlet」にして「次へ」ボタンを押します。

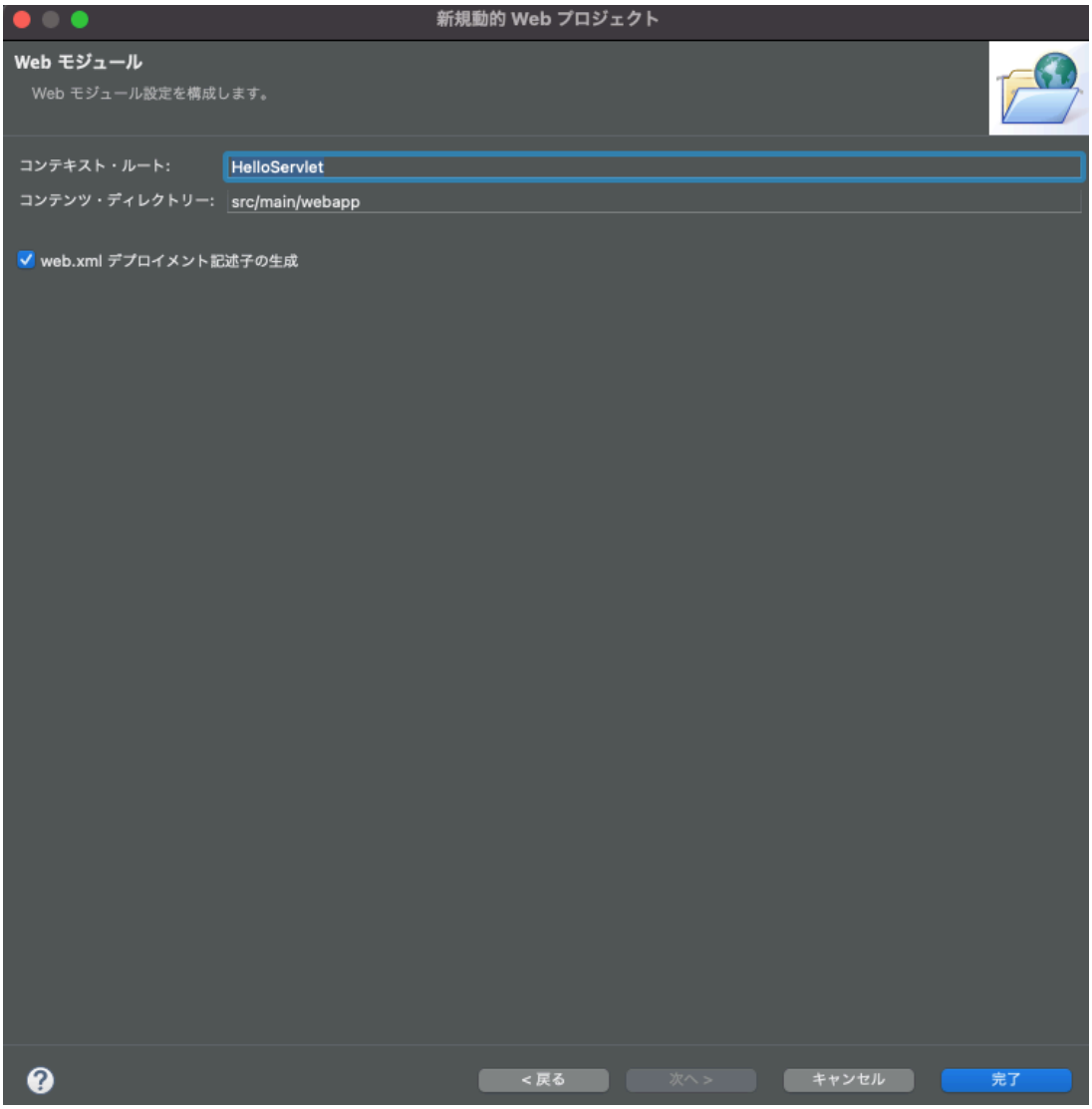


次は、デフォルトのまま次へボタンをクリックします。

?



「web.xml デプロイメント記述子の生成」にチェックを入れて完了を押しましょう。

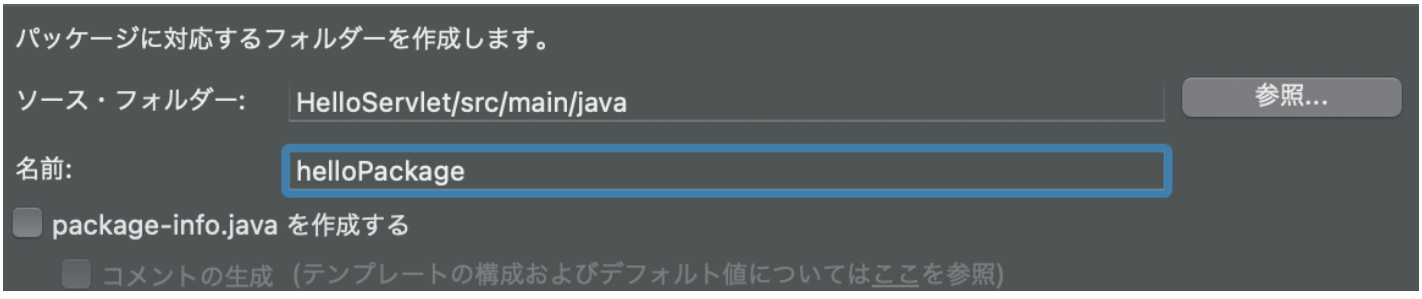


Step2: パッケージの作成

《HelloServlet》→《src/main/java》で右クリック。

《新規》→《パッケージ》を選択します。

パッケージ名は「helloPackage」としましょう。



Step3: クラス（Javaファイル）の作成

こちらは今までと同じ作業で、「HelloServlet」クラスを作成しましょう。

Step4: HelloServlet.java　に「HelloServlet」と表示するロジックを記述

```
package helloPackage;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

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

        response.setContentType("text/html; charset=Windows-31J");
        PrintWriter out = response.getWriter();
        out.println("<body>HelloServlet</body>");

    }
}
```

Step5: web.xmlの編集

このままでは入力されたURLと実行されるファイルとの紐付けができないので、
紐づけるための処理を書いていきます。

《HelloServlet》→《src》→《main》→《webapp》→《WEB-INF》→《web.xml》
こちらを開いて、<web-app>タグの間に下記のコードを貼り付けて下さい。
（※「ここから～ここまで」の内容をコピーしてください。

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID"
version="4.0">

    <!-- ここから -->
    <servlet>
        <servlet-name>helloServlet</servlet-name>
        <servlet-class>helloPackage.HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>helloServlet</servlet-name>
        <url-pattern>/sample</url-pattern>
    </servlet-mapping>
    <!-- ここまで -->
</web-app>
```

Step6: 実行

それでは実行してみましょう！

動作確認を行います。

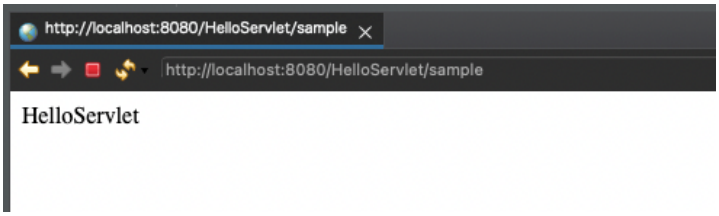
プロジェクトを右クリックし、
「実行」→「サーバーで実行」から実行します。

内部Webブラウザビューが自動で開きます。

URLの末にURLのパターン名（ここでは「 `sample` 」）を入力し、
http://localhost:8080/HelloServlet/sample

としてエンターを押すとコードの内容が表示されます。

サーバーが起動している間は、 Eclipseの内部Webブラウザでないブラウザでも表示可能です。
（ **Chrome**や **IE（Internet Explorer）** 、 **Firefox** などで開いてみましょう！



> **解説**

サーブレットプログラムもクラスライブラリを利用します。
今回のクラスは **HttpServletクラス** です。

今までと違うのは **HttpServletクラス** を継承し、必ず「 `doXXX()` 」メソッドを オーバーライド して利用するところです。

「 `doXXX()` 」メソッドは、全部で7種類ありますが、
Webサーバ（HTTPサーバ）とやり取りするものがブラウザの場合は、 `doGet()` メソッド と `doPost()` メソッド の2つのみです。

上記のプログラムの例では、doGet()メソッドを使っています。
（ `extends` しているHttpServletクラスのdoGet()メソッドをオーバーライドしています）。

```
// オーバーライドなので一字一句間違えず記述する
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    ...
}
```

次に、 `doGet` のメソッドと、中身の処理について簡単に見ていきます！

- 第1引数の **HttpServletRequestインタフェース** は、HTTPリクエストに対応
- 第2引数の **HttpServletResponseインタフェース** は、HTTPレスポンスに対応

■1行目: `response.setContentType("text/html; charset=Windows-31J");`
`setContentType()` メソッドでレスポンスの **MIMEタイプ** を指定します。
この値は、そのままHTTPヘッダーの「 `Content-Type` 」の値として指定されます。

・Content-Type と MIMEタイプ

テキストや画像、オーディオファイルなど、拡張子だけでは転送先のブラウザ上で判別が難しい情報を伝えるための設定になります。
`"text/html; charset=Windows-31J"` までが `Content-Type` としての扱いになります。

一部ですが、以下のような組み合わせになります。

Content-Type	MIME タイプ
text	text/html や text/css
image	image/png や image/jpeg

（上記はほんの一部ですが、記述形式はどれも同様です。

また「 `charset=XXX` 」の部分は、ここで指定された文字コードに変換されてページを表示します。
サンプルでは `Windows-31J` を指定していますが、
日本語のページを表示する場合には、文字コードを **「Windows-31J」** 又は **「EUC_JP」** に指定する必要があります。

試しに「 ～ （大文字の波線）」を **utf-8** で出力してみると、「 ？ 」と表示されてしまいます。
（想定していない文字になったり、文字化けが発生したりした場合には、 文字コードの設定を確認しましょう！

■2行目: `PrintWriter out = response.getWriter();`
responseより呼び出した `getWriter()` メソッドで、 **PrintWrtierクラス** のインスタンス（ `out` ）を生成します。

■3行目: `out.println("<body>HelloServlet</body>");`
PrintWrtierクラス では、 `print()`や `println()` が利用できます。
今までの標準出力（コンソール）に表示する `System.out.print()` と同様の感覚で、
PrintWrtierクラス のprint()メソッドやprintln()メソッドを扱うことができます。

web.xmlについて

・web.xml
Webアプリケーションに関する様々な設定をxml形式で定義します。

- アクセスされたURLに対して呼び出すクラス
- エラー時に実行する処理の指定

などの設定をします。

・配置箇所
web.xmlは **「WEB-INF」ディレクトリ直下** に配置し、Webアプリケーション毎に作成します。
（※「web.xml デプロイメント記述子の生成」にチェックして作成した場合は、デフォルトで作成されます。

・読み込みのタイミング
Webサーバ（HTTPサーバ）の起動時にWebアプリケーション毎にweb.xmlが読み込まれます。

・web.xmlの基本構造
「`XML宣言`」、「`DTD宣言`」、「`本体`」の順で構成されています。

【テンプレート】

```
<?xml version="1.0" encoding="ISO-8859-1"?>                    ･･･XML宣言
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"    ･･･DTD宣言
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  ･･･                                                          ･･･本体の記述
</web-app>
```

web.xmlは、Webサーバの起動時に読み込まれ構文チェックが行われます。
正しく記述されてない場合は、 **「致命的エラー」扱いとなりそのコンテナは動きません。**
正しく記述されていれば、どのような「URL」で、どのサーブレットを動かすかという定義が「本体」に必要となります。
（※サーブレットが上手く機能しない場合は、 web.xmlの設定や書き方に間違いがないかチェックしましょう

```
<web-app>

  <servlet>
    <servlet-name>【サーブレットの命名】</servlet-name>
    <servlet-class>【パッケージ名】.【サーブレット名】</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>【サーブレットの命名】</servlet-name>
    <url-pattern>/【URLのパターン名】</url-pattern>
  </servlet-mapping>

</web-app>
```

解説

・ `<web-app>` **タグ**
サーブレットの命名には「任意の名前」を付与
（ **※日本語は付けられません**

・命名規則
`<servlet>`タグ内と `<servlet-mapping>` タグ内の2つで実施し、
上記2つのタグにおけるサーブレットの命名は **同一** でなくてはなりません。
`<servlet>` タグには、 `<servlet-class>` タグがあり、動かしたいサーブレット名を記述します。
（パッケージ化されている場合は、 `helloPackage.HelloServlet` のようにパッケージ込みのサーブレット名を記述します。

・ `<servlet-mapping>` **タグ と** `<url-pattern>` **タグ**
このタグ内に記述された文字列で、ブラウザのURL（アドレス）などで指定した場合、
先ほどの動かしたいサーブレットが動作する仕組みとなります。

上記 `<servlet-mapping>` における設定は、 **必須** となり、
この設定を **サーブレットマッピング** と呼びます。

先程動かしたテストプロジェクトを例に取ると、以下のような流れで指定したサーブレットが処理されます。

①「～～/sample」というURLに遷移する

②web.xmlの<servlet-mapping>タグ内で、一致するurl-pattern「/sample」が見つかり、helloServletというサーブレットネームを付けている

③<servlet>タグ内に、一致するhelloServletというサーブレットネームが見つかる

④そのサーブレットネームには、helloPackage.HelloServletクラスが指定されている

⑤HelloServletクラスが実行される

初めのうちは数多くあるタグの組み合わせに慣れないかもしれませんが、
何度も作成して練習しましょう！

マッピングは「sample」などよりも、 **可読性** と **サーブレットの紐づけの親和性** が高まるような
その機能に合った意味を持つ文言を記述するようにしましょう！

エラーが表示される場合

サーバー実行時にエラーが表示される場合があります。

前述で「 **致命的なエラー** 」のように表現していますが、
具体的にどのような場合に発生するか見ていきましょう。

【web.xml: 記述ミスなし】

ユーザー情報を返すようなサーブレットと仮定しましょう。

```
...
<servlet>
    <servlet-name>userInfo</servlet-name>
    <servlet-class>UserInfoServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>userInfo</servlet-name>
    <url-pattern>/user</url-pattern>
</servlet-mapping>
```

上記の場合は、サーバーは正常に起動します。

しかし、記述ミスが存在する場合は、画像のようなエラーが表示され、サーバーの起動に失敗してしまいます。

【web.xml: 記述ミスあり その１】

```
...
<servlet>
    <!-- servlet-nameが異なる -->
    <servlet-name>userinfo</servlet-name>
    <servlet-class>UserInfoServlet</servlet-class>
</servlet>
<servlet-mapping>
    <!-- servlet-nameが異なる -->
    <servlet-name>userInfo</servlet-name>
    <url-pattern>/user</url-pattern>
</servlet-mapping>
```

【web.xml: 記述ミスあり その２】

```
...
<servlet>
    <servlet-name>userInfo</servlet-name>
    <servlet-class>UserInfoServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>userInfo</servlet-name>
    <!-- url-patternの「/（スラッシュ）」漏れ -->
    <url-pattern>user</url-pattern>
</servlet-mapping>
```

> **起動エラー**

> **NGパターン**

- `servlet-name` が合致しない
  - そもそも **文字が異なる**
  - **小文字と大文字** で異なる
- `url-pattern` に記述ミス
  - アクセス先URIを記述しているが、 **接頭の** `/（スラッシュ）` **が漏れている**

例に記述した `userinfo` と `userInfo` ですが、パッと見なかなか気づけないかもしれません。
また、 `url-pattern` もパスの記述ミスがあると同様のエラーダイアログが表示されます。

マッピングするサーブレット情報が1、2つくらいならまだいいのですが、
増えていくほどに探し当てるのは困難になります。

コンパイルエラーが出力されない分、
こういった細かなミスを探すのは骨が折れますので、記述内容はしっかり見直すように心がけましょう！

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 09月 10日(土曜日) 05:06