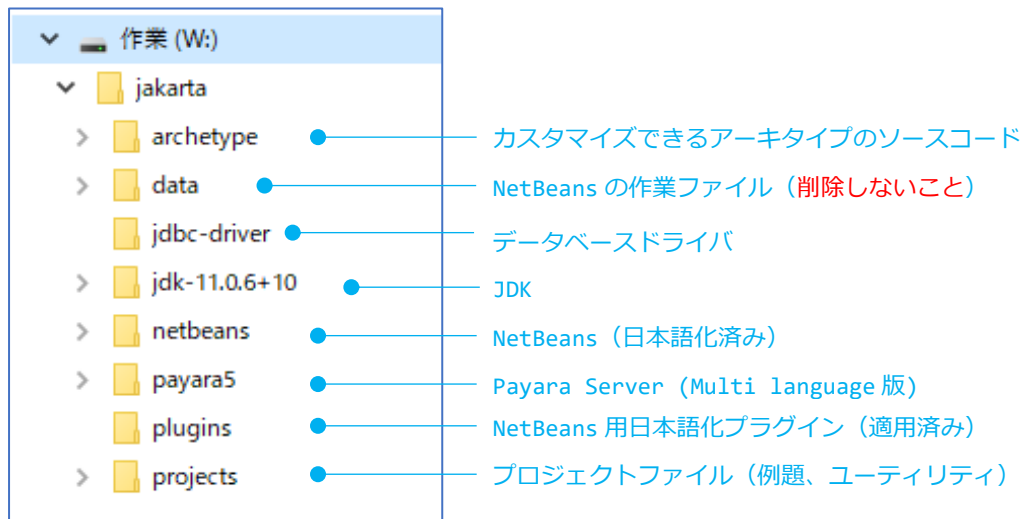


JET(JakartaEE Toolkit)ver0.90 の使い方

1. 展開場所とフォルダ名の制限

解凍したフォルダは、どこかのドライブの直下に置かなくてははいけません。また、フォルダ名を変更すると動作しないので、注意してください。

例えば、Wドライブに置く場合は、次のようになります。

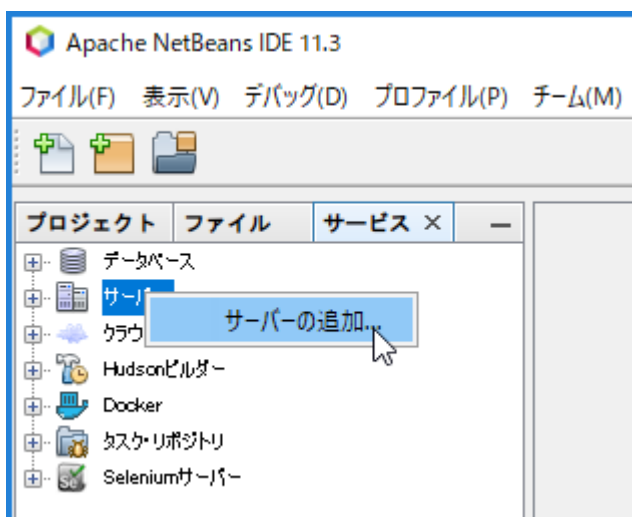


2. NetBeans の起動

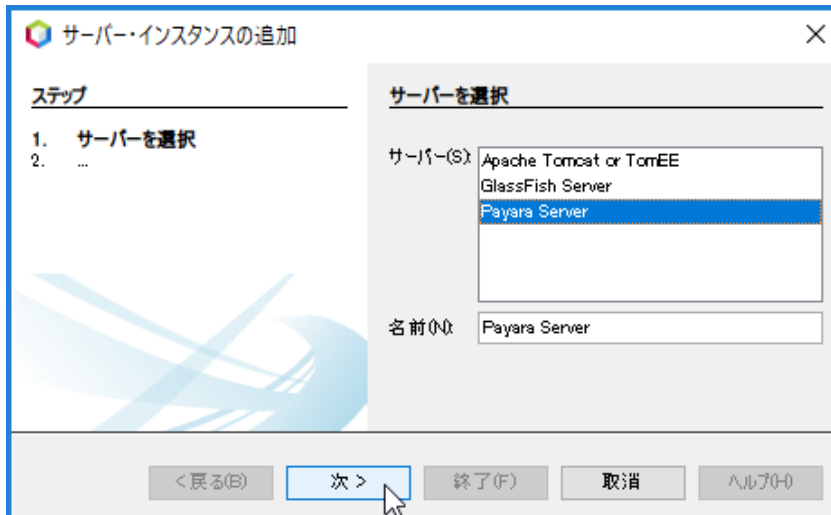
<ドライブ> / jakarta/netbeans/bin フォルダにある、netbeans64.exe (32 ビットシステムの場合は netbeans.exe) をダブルクリックすると起動します。ファイル名を右ボタンでクリックして、ショートカットを作り、デスクトップに置いておくと、すばやく起動できます。

3. Payara サーバーとの連携を設定する

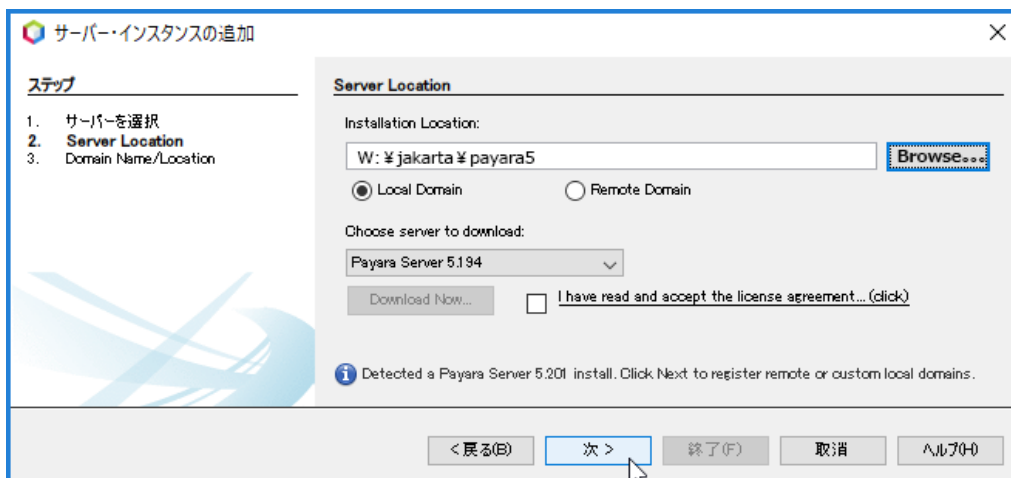
①起動した NetBeans で、[サービス] タブをクリックし、[サーバー] を右クリックして、[サーバーの追加] を選びます。



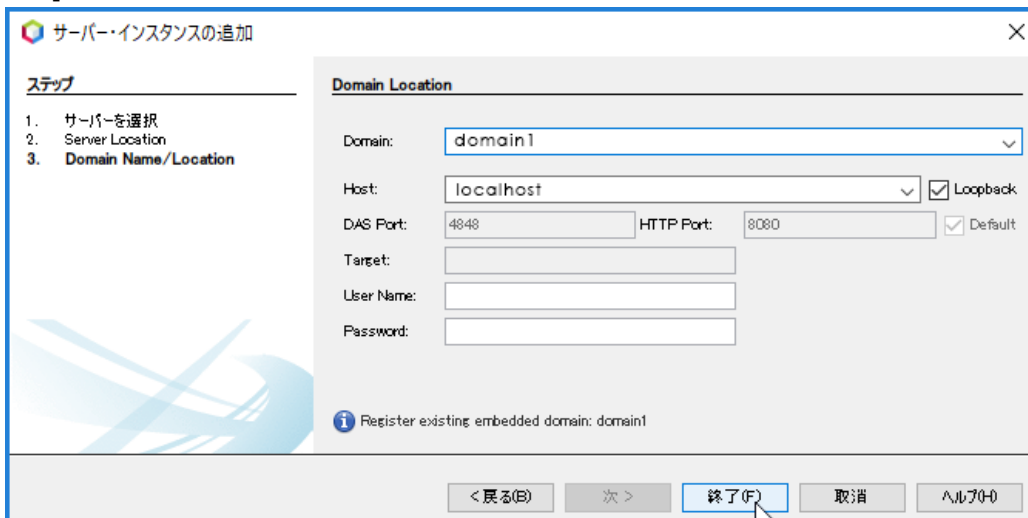
②ダイアログで、[Payara Server] を選んで [次へ] を押します



③ [Browse...] ボタンを押して、<ドライブ> /jakarta/payara5 を選択します。
次のような画面になるので、[次へ] を押します。



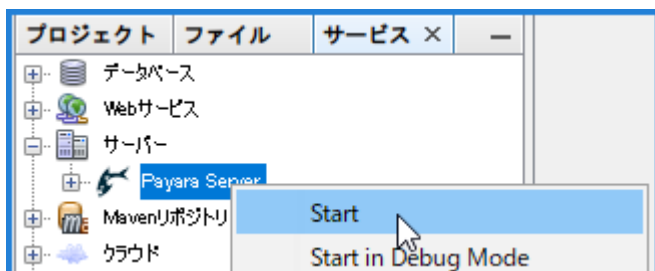
④ [終了]を押します



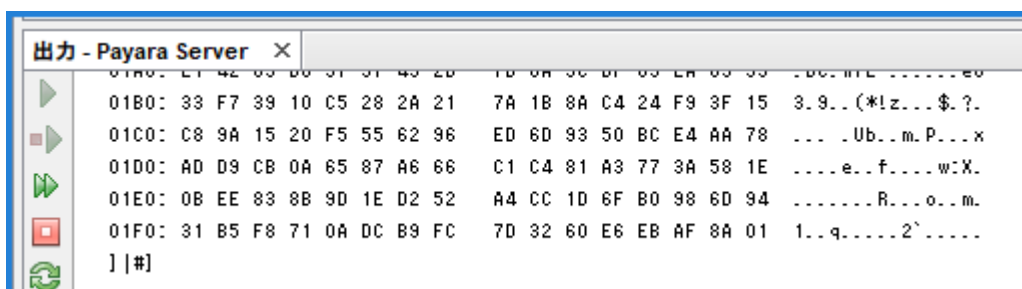
4. Payara サーバーを起動する

サーバーとして、Payara サーバーが表示されるので、起動します

- ① Payara Server を右ボタンでクリックして [Start] を選ぶ



- ② 下段にサーバーログが開いて、起動する様子が表示されます



5. 新規プロジェクトを作成して、実行する

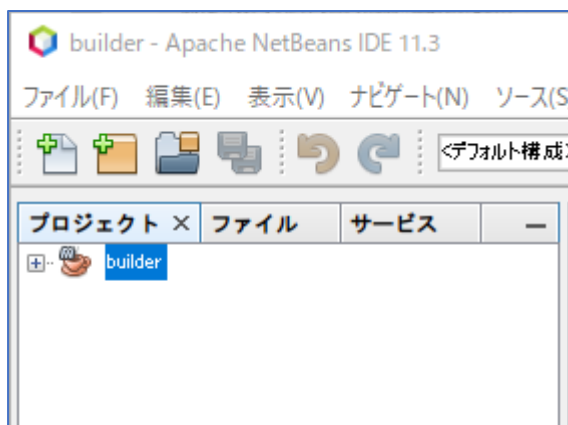
builder プロジェクトを使って JSF を使う新規プロジェクトを作成します。

※archetype を使う方法も使えますが、実はこちらの方が簡単です

- ① builder プロジェクトを開く

新規プロジェクトのひな型を自動作成する builder プロジェクトを開きます。

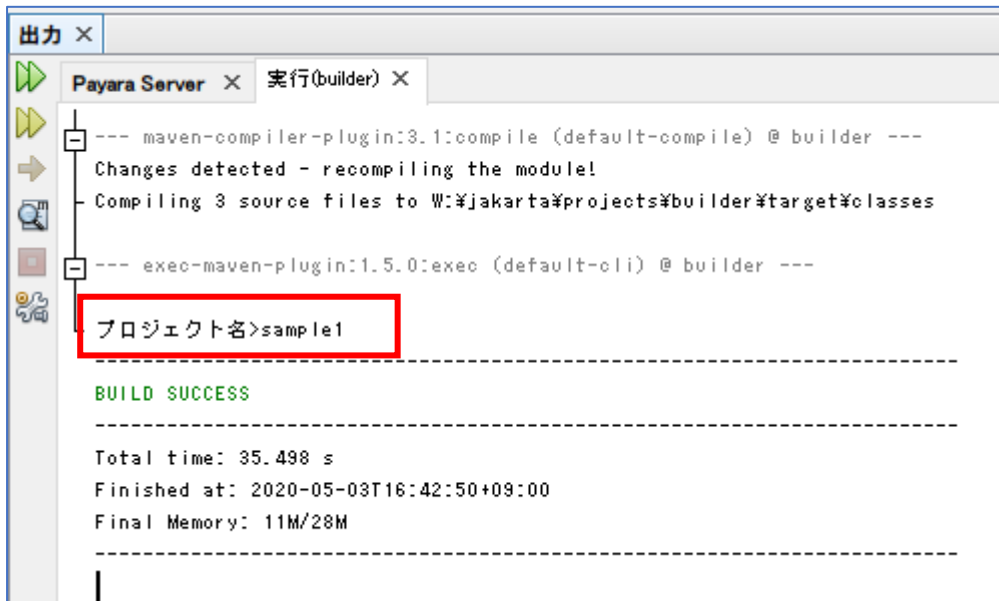
builder プロジェクトは、<ドライブ>/jakarta/projects フォルダにあるので、[ファイル]⇒[プロジェクトを開く]と選択して開いてください。



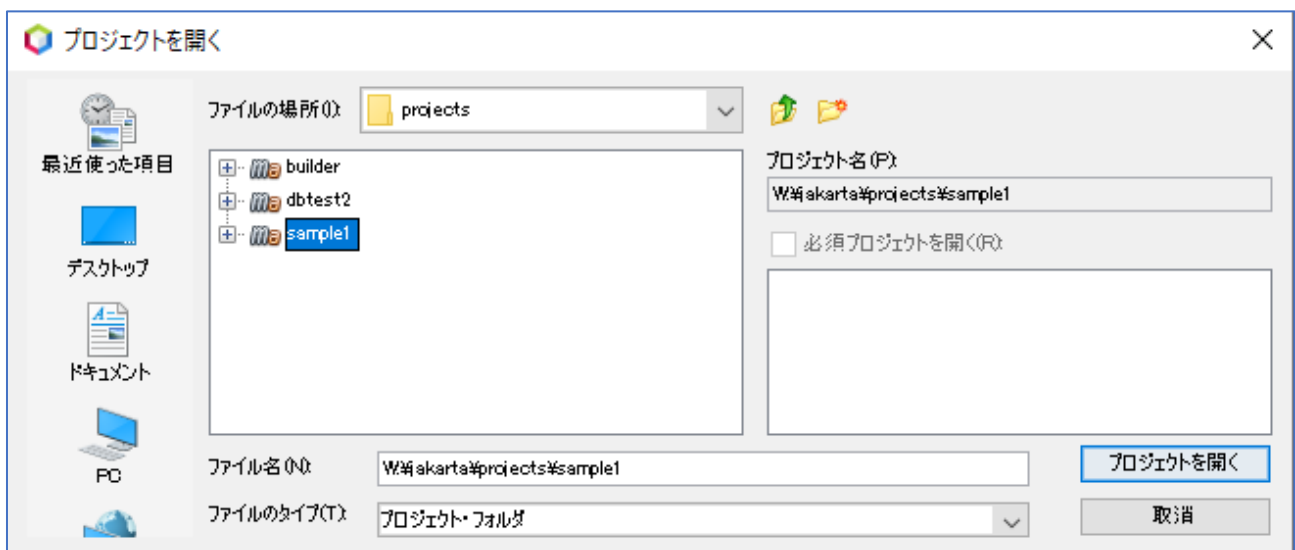
②builder プロジェクトを一度クリックしてから、実行ボタン (▶) を押して、実行します。

コンソールに**作成したいプロジェクト名**を入力するプロンプトが表示されるので、ここでは、**Sample1** と入力して Enter キーを押します。

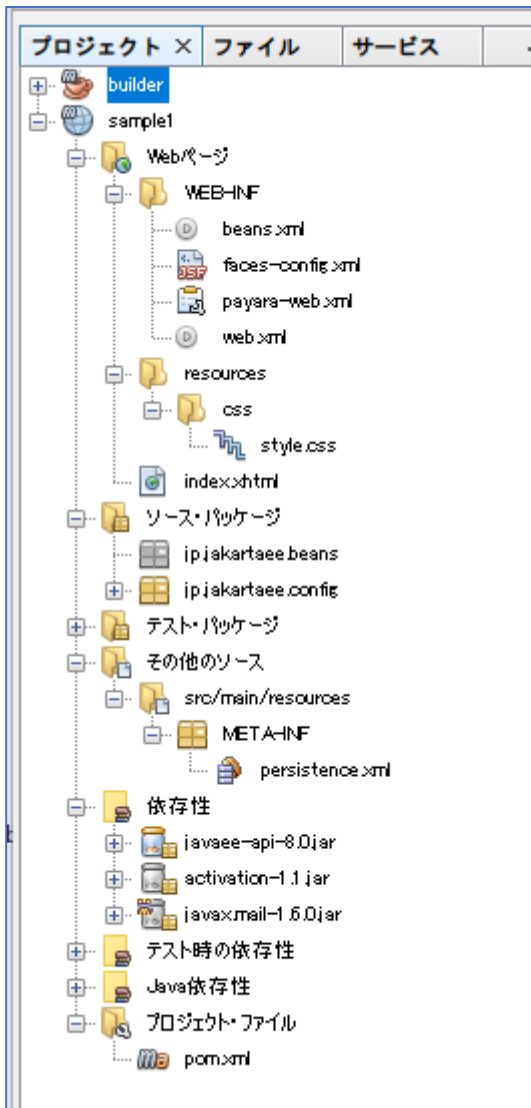
この操作で Sample1 プロジェクトが projects フォルダに自動作成されます。



③projects フォルダから、作成された Sample1 プロジェクトを開きます



④ Sample1 プロジェクトの内容を確認する



JSF フレームワークのための、設定ファイルは、WEB-INF フォルダ内に、すべて生成済です。また、リソースとして、CSS スタイルシート (style.css) も置いてあります。index.xhtml は次のようです。

```
index.xhtml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Trans
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.icp.org/isf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    Hello from Facelets
  </h:body>
</html>
```

ソースパッケージの `jp.jakartaee.config` には、JSF のバージョンを示すための Java クラスが入っています。JSF2.3 から CDI と連携するために必要になったクラスです。このまま置いておくだけで構いません。

`persistence.xml` は、`jdbc/mydb` というデータソースを参照します。汎用的な記述になっているので、通常は変更なしで使えます。



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
  xmlns="http://xmlns.icp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.icp.org/xml/ns/persistence
    http://xmlns.icp.org/xml/ns/persistence/persistence_2_2.xsd">
  <!-- Define Persistence Unit -->
  <persistence-unit name="sample1-PU" transaction-type="JTA">
    <jta-data-source>jdbc/mydb</jta-data-source>
    <exclude-unlisted-classes>>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Pom ファイルは次のようです。

jakartaEE の 8.0 を使う指定です。JDK は LTS であるバージョン 11 を使ってコンパイルします。

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>jp.jakartaee</groupId>
  <artifactId>sample1</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>
  <name>sample1</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <failOnMissingWebXml>>false</failOnMissingWebXml>
    <jakartaee>8.0</jakartaee>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>${jakartaee}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
```



```

    <artifactId>junit-jupiter</artifactId>
    <version>5.5.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.23.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>

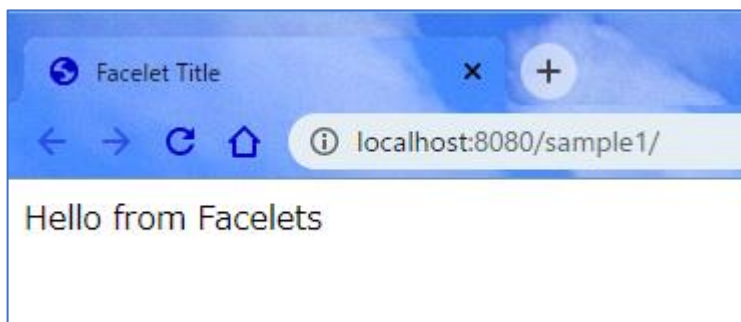
<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>11</source>
        <target>11</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.2.3</version>
      <configuration>
        <failOnMissingWebXml>>false</failOnMissingWebXml>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>

```

⑤プロジェクトを実行する

バックエンドのクラスはありませんが、JSF プロジェクトとして実行できます。プロジェクトを一度クリックしてから、ビルドボタン () を押し、さらに、(Payara サーバーが起動していることを確認して) 実行ボタン () を押してください。

次のようなウェブ画面が表示されます。



以上で、新規 JSF プロジェクトを作成して、実行できることがわかりました。後は、このファイルにクラスやウェブページを追加すると、実際の JSF プロジェクトを構築できます。

データベース設定について

1. 付属している Payara Server での JDBC 設定は次の通りです。

JDBC 接続プール	DB 製品	DatabasName (スキーマ名)	ユーザー	パスワード	JDBC リソース名
mydb_pool	MySQL	mydb	root	mysql8	jdbc/mydb
mydb_pg_pool	PostgrnSQL	mydb_pg	postgres	postgres	jdbc/mydb_pg
mydb_h2_pool	H2	mydb_h2	app	app	jdbc/mydb_h2

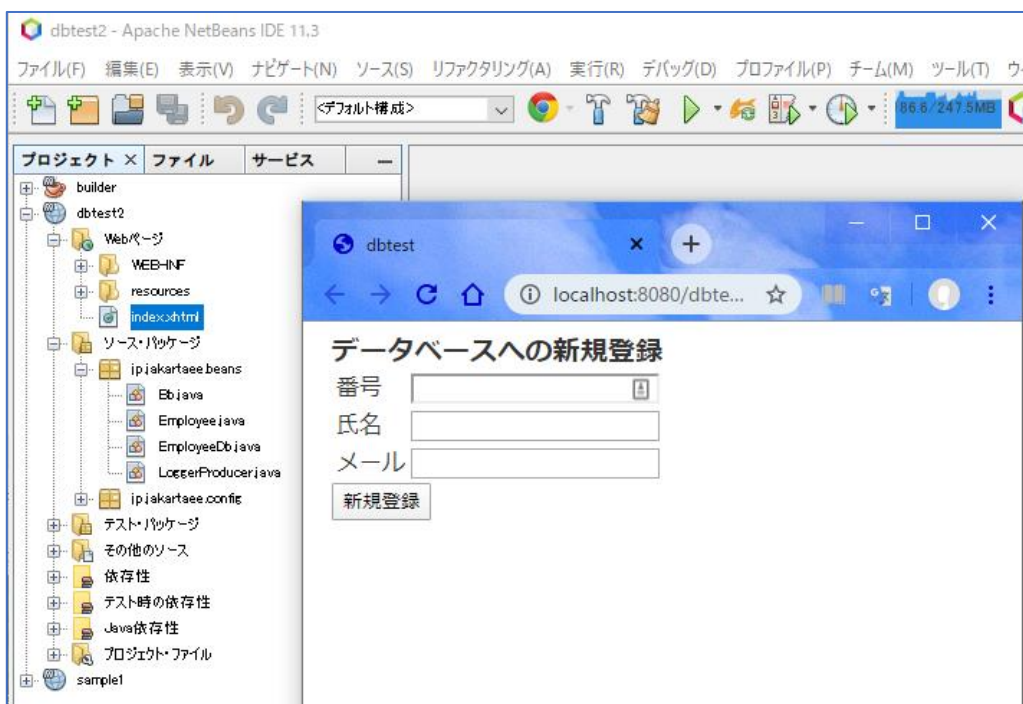
データベースを定義する際、Database 名 (スキーマ名)、ユーザー名、パスワードを、これと違うもので設定した場合は、Payara サーバーの管理コンソール (<http://localhost:4848>) の [JDBC 接続プール] 設定で、それぞれの接続プールを選択し、[追加プロパティ] タブを開いて書き換えてください。

2. 使用するデータベースは、Docker を使って準備します

具体的な方法は、Qiita ブログの「Jakarta EE+Payara server+NetBeans で開発しよう！(その2)」を参照してください。

3. データベースを起動したら、テストとして、dbtest2 プロジェクトを開いて実行します

次のような入力を行うウェブが開きます。記入して、新規登録ボタンを押すと、EMPLOYEE というテーブルが作成されて、レコードが書き込まれます。



結果は、NetBeans 上から確認できます。方法は、上記のブログに書いてあります。

(次ページの図を確認してください)

The screenshot shows the NetBeans IDE interface. On the left, the 'プロジェクト' (Project) view displays a tree structure of database connections. The 'mydb-mysql' connection is expanded to show a table named 'EMPLOYEE' with columns 'NUMBER', 'MAIL', and 'NAME'. The '出力' (Output) window at the bottom right shows the execution of a SQL query: 'SELECT * FROM EMPLOYEE LIMIT 100;'. The output displays a single row of data:

#	NUMBER	MAIL	NAME
1	12345	tanaka@mail.jp	田中ひろし

The output window also shows the following text: [1:1] 0.003秒で実行が成功しました。結果セットのフェッチに0.002かかりました。0.066後に実行が終了しました。no errorsが発生しました。

Maven archetype "jakarta-ee-jsf" について

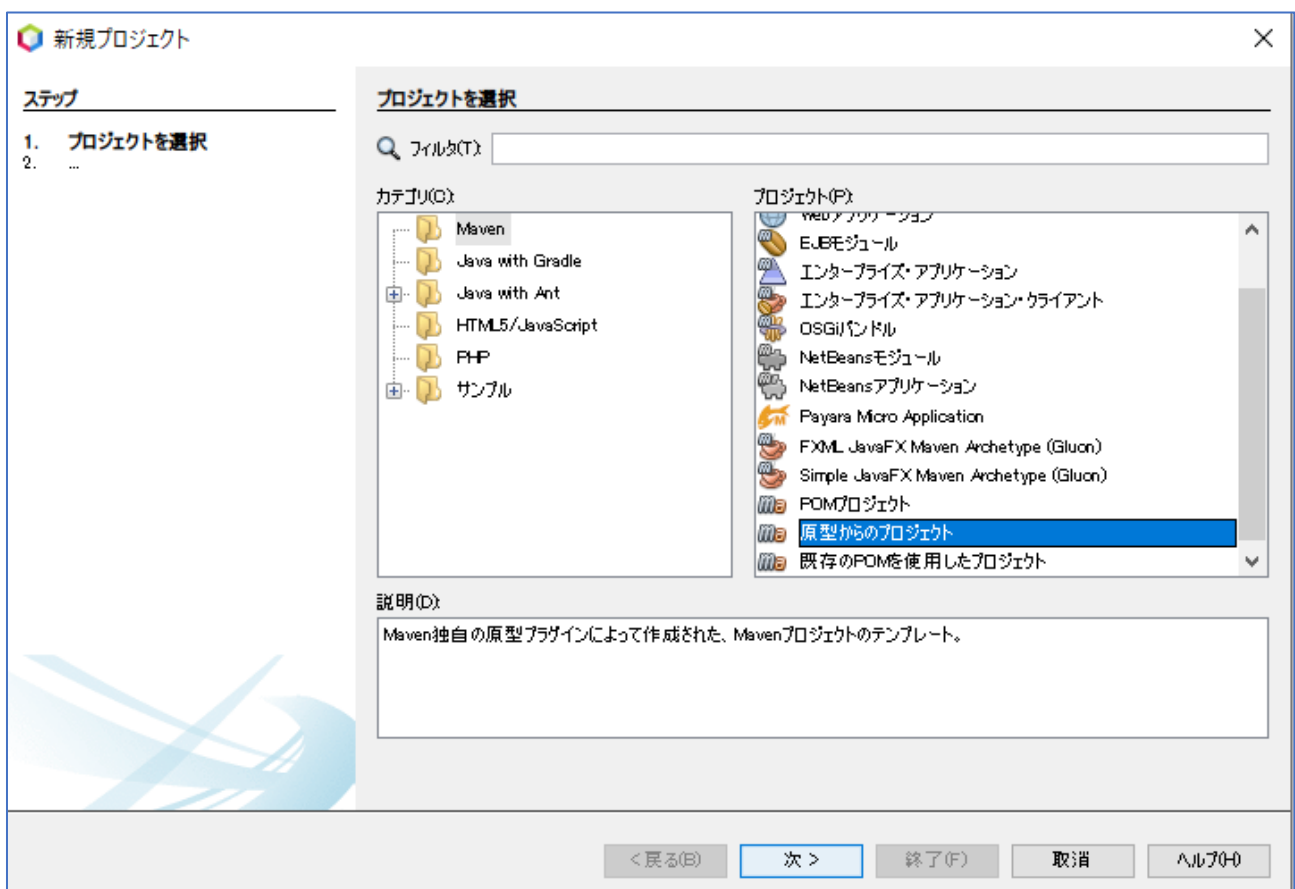
アーキタイプの作成と利用法については、別途、ブログ等でお知らせする予定です。ここでは、ある程度の関連知識があることを前提に説明します。

コマンドプロンプトを起動し、<ドライブ> /jakarta/archetype/jsf-maven-archetyp ディレクトリにて、`mvn clean install` コマンドを実行すると、ローカルリポジトリに、`jakarta-ee-jsf` というアーキタイプが登録されます。

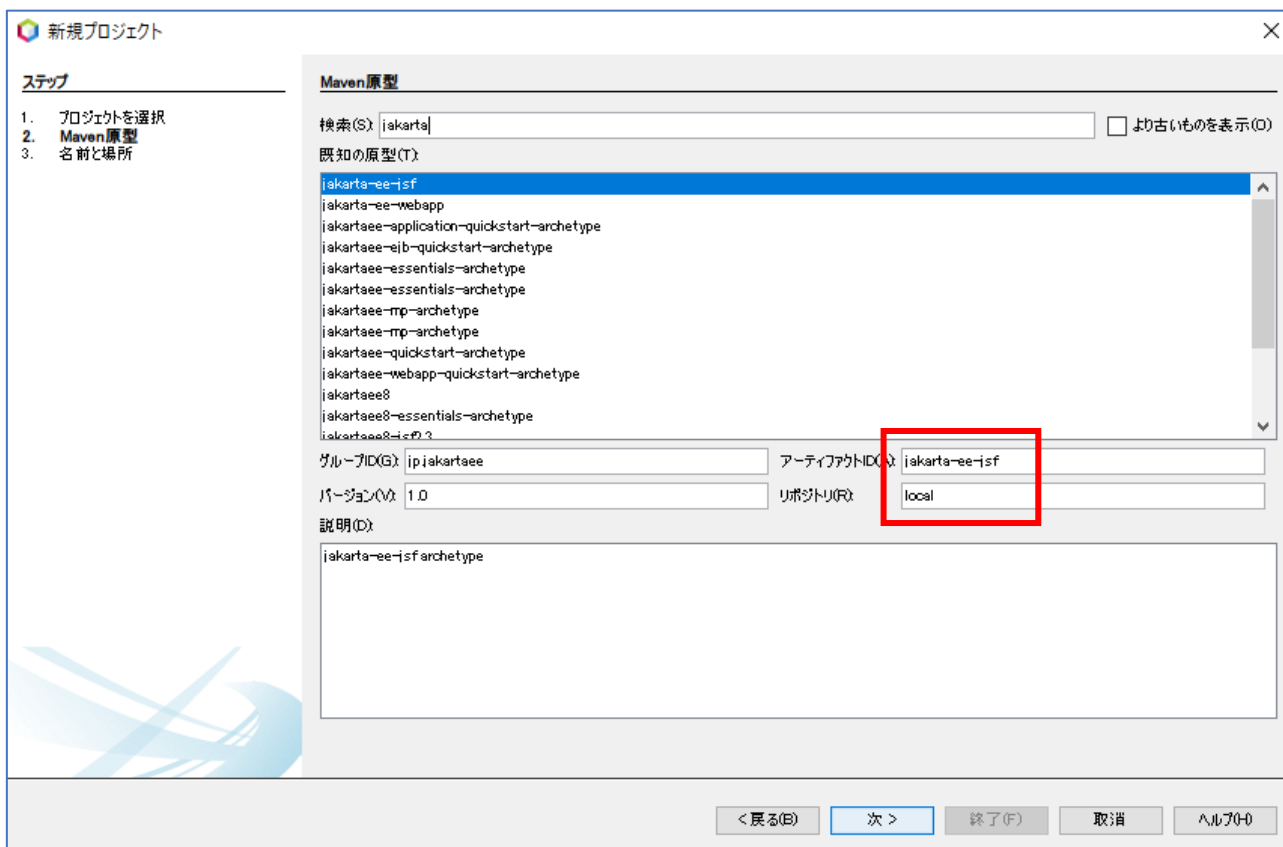
```
> w:  
> cd jakarta\archetype\jsf-maven-archetyp  
> mvn clean install
```

※実行するには、JDK と maven をインストールしておく必要があります

NetBeans から利用するには、[新規プロジェクト] ⇒ [新規プロジェクト] ⇒ [Maven] : [原型からのプロジェクト] を選択します。



アーキタイプを選択する画面になるので、検索欄に、jakarta と入れると、jakarta-ee-jsf が先頭に表示されます。builde プロジェクトで生成されるものとほぼ同じプロジェクトが生成されます。



注意

ローカルリポジトリは、デフォルトでは、Cドライブの ユーザー/<ユーザー名>/.m2/repository にあります。mvn clean install を実行すると、この中に archetype-catalog.xml が生成されて、ローカルリポジトリのアーキタイプとして認識されるようになります。