掲示板アプリケーション環境構築手順書

管理番号	
システムID	
システム名称	
改定日	
改訂者	

目次

1. はじめに

◇1-1.本書の目的

本書は、掲示板アプリケーションのソースコードをダウンロードし、ローカル環境で動作させるまでの手順をまとめたものです。
GitHubからのクローン、データベースの構築、必要な設定ファイルの編集Eclipseプロジェクトのインポート、アプリケーションの起動と動作確認までを、
一通りガイドしています。
この資料を使うことで、誰でもスムーズに開発環境を再現できることを目指しています。

◇1-2.対象読者

本書は、以下のような方を対象としています。
・Java開発の基本的な知識がある方
・Eclipse IDEの基本操作を理解している方
・データベース（MySQLなど）の基本的な操作ができる方
※特にStruts2フレームワークの利用経験は問いませんが、簡単な設定ファイル編集（db-config.propertiesやstruts.xmlなど）が発生しますので、
初学者の方は参考書籍などを併用して進めることを推奨します。

2. 動作環境

◇2-1.使用技術一覧（※本プロジェクトの動作確認済み構成）

掲示板アプリケーションは以下の技術を使用しています。

分類	技術名 / バージョン例
プログラミング言語	Java SE 17（Eclipse 2023-09標準 JDK使用）
フレームワーク	Apache Struts2 (2.5.26)
データベース	MySQL 8.0.32
ビルドツール	Apache Maven 3.8.x（pom.xml に準拠）
アプリケーションサーバ	Apache Tomcat 9.x
その他	JSTL 1.2、MySQL Connector/J 8.0.32（JDBCドライバ）

◇2-2.開発環境

本アプリケーションは以下の環境で開発・動作確認しています。	
項目	内容
IDE（統合開発環境）	Eclipse IDE 2023-09（Enterprise Java & Web Developers版）
OS	Windows 10 / 11
Java開発キット（JDK）	JDK 17（EclipseバンドルJDK使用）
データベース	MySQL Community Server 8.0.32
アプリケーションサーバ	Apache Tomcat 8.5.96

◇2-3.使用ライブラリ

本アプリケーションは以下の環境で開発・動作確認しています。			
項目	内容	使用用途	備考
HikariCP	com.zaxxer:HikariCP:3.4.5	高速・軽量なJDBCコネクションプールライブラリ	接続プールの性能・安定性が高く、多くの現場で採用
Struts2 Core	org.apache.struts:struts2-core:2.5.26	Struts2フレームワークの中核ライブラリ	Struts2の画面遷移、OGNL連携の基盤となる
Log4j API	org.apache.logging.log4j:log4j-api:2.14.1	ロギング用API（ログの出力・管理のインタフェース）	設定ファイル（log4j2.xml等）と連携して動作
Log4j Core	org.apache.logging.log4j:log4j-core:2.14.1	ロギング処理の実装部分（Log4jの本体）	APIとセットで使用、単体では動作しない
Servlet API	javax.servlet:javax.servlet-api:4.0.1（scope: provided）	サーブレット仕様のAPI、Tomcat等アプリサーバ提供	providedスコープのためWARに同梱されない
MySQL Connector/J	com.mysql:mysql-connector-j:8.0.32	MySQLデータベース用のJDBCドライバ	MySQL用のDB接続必須ライブラリ
OGNL	ognl:ognl:3.1.29	Struts2で使用される式言語ライブラリ（データ参照等）	Struts2の画面とアクションのデータバインドに必要
XmlPull	xmlpull:xmlpull:1.1.3.1	XMLパース用の軽量ライブラリ	Struts2の内部依存で使用
XPP3 Min	xpp3:xpp3_min:1.1.4c	XmlPull Parserの実装、XMLデータの読み込みに使用	XmlPullの実装ライブラリ、単体利用はほぼない

補足：

・pom.xml 内でMySQL Connector/J は8.0.32を指定しています。必ず同バージョンを使用してください。
・Tomcatは9.x系でも動作確認済ですが、安定運用を考慮し8.5.x系を**推奨**しています。
・JDKはEclipse 2023-09標準バンドル版（JDK 17）で動作確認しています。

重要メッセージ

本アプリケーションは、pom.xml で指定したバージョンと完全一致する環境でのみ動作確認をしています。
特にMySQL Connector/Jは8.0.32を指定しており、このバージョン以外は非推奨＆非サポートとします。
Maven依存関係の更新だけで必要なライブラリは自動的に揃いますが、環境設定（Tomcatのバージョンなど）が異なると再現性が失われる可能性があるため、
必ず推奨バージョンを使用してください。
詳細については本書の「10. トラブルシューティング」をご覧ください。

以上

3. 前提条件

3-1.Mavenプロジェクト前提

・本プロジェクトはMavenベースで管理されています。
pom.xml内で全ての依存ライブラリが管理されており、**手動でビルド・パスを追加/削除することは非推奨です。**

3-2.データベース設定 (db-config.properties)

・ファイルパス:/src/main/resources/db-config.properties
・内容:データベース接続用のユーザー名・パスワードDB名を記述。
・Maven更新時、このファイルはtarget/classesフォルダにコピーされ、実行時はこのパスから読み込まれます。
　環境に応じて内容を編集する必要があります(詳細は「ビルド＆実行方法」を参照)。
注意: 稀にMaven更新後もコピーされないケースがあり、その場合は「Javaのビルド・パス」の「ソース」タブに/src/main/resourcesを追加することで解決できます。

・HikariCPによる接続プール設定
本プロジェクトでは、JDBC接続に HikariCP を使用しています。
HikariCPの依存関係は pom.xml に以下のように記述されています:

```
<dependency>
<groupId>com.zaxxer</groupId>
<artifactId>HikariCP</artifactId>
<version>3.4.5</version>
</dependency>
```

接続先やユーザー名、パスワードは db-config.properties から読み込み、
DatabaseConnectionManager クラス内の HikariConfig で以下のように設定されます:
・maximumPoolSize = 10(接続プールの最大数)
・idleTimeout = 30000(アイドル接続のクローズ時間:30秒)
・maxLifetime = 1800000(接続の最大寿命:30分)
・driverClassName = com.mysql.cj.jdbc.Driver
また、startMonitoring() メソッドにより接続プールの状態(アクティブ接続数、アイドル接続数、総接続数、待機スレッド数)が
5秒ごとにログ出力され、運用中の監視が可能です。

3-3.その他の前提

・Javaバージョン: Java SE 8以降。
・アプリケーションサーバ Tomcat 8.x推奨(9.xでも動作確認済み)。
・データベース: MySQL 8.0.32推奨(pom.xmlで指定)。

以上

4. ソースコードの取得（概要）

本アプリケーションのソースコードは、GitHub（または任意のリポジトリ）から取得できます。

・ソース管理：Git

・プロジェクト形式：Mavenプロジェクト（Eclipse対応済）

リポジトリURL：https://github.com/juehara-crypto/Portfolio_Bulletinboard

4-1.GitHubからのクローン手順（概要）：

1.お使いの環境（Windows / macOS / Linux）にGitクライアントがインストールされていることを確認してください。

※ インストールされていない場合は、Git公式サイトからインストールしてください。

2.コマンドライン、もしくはGUIクライアント（例：GitHub Desktop、SourceTreeなど）からソースコードをクローンしてください。

コマンドライン例：

git clone https://github.com/juehara-crypto/Portfolio_Bulletinboard.git

GUIクライアントの場合は、クライアントの「Clone Repository」機能を使い、上記URLを指定してください。

補足：詳細な取得・インポート手順について

詳細な手順は「8. ビルド＆実行方法（詳細設定編）」で説明します。

以上

5. Eclipseのインポート手順（概要）

・開発環境：Eclipse IDE for Java Developers
・プロジェクト形式：Mavenプロジェクト

5-1. プロジェクトのインポート方法（概要）

1,Eclipseを起動し、[ファイル] → [インポート] → [既存のMavenプロジェクト] を選択します。
2.「ルートディレクトリ」に、クローンしたソースコードの保存先フォルダ（例C:\workspace\Bulletinboard）を指定します。
3.[プロジェクト] に表示されるプロジェクトを確認し、[完了] をクリックします。

5-2. ビルドパス設定（ライブラリの概要）

・Mavenプロジェクトなので、依存ライブラリはpom.xml に記述されています。
・インポート後、Eclipseの「プロジェクトを右クリック→ Maven → プロジェクトの更新（Update Project）」を実行することで、必要なライブラリが自動ダウンロードされます。

補足：詳細なインポートおよびビルドパス設定手順について
詳細な手順は「8. ビルド＆実行方法（詳細設定編）」で説明します。

以上

6.データベース設定（概要）

・本システムでは掲示板（bulletinboard）、スレッド（threads）、投稿（posts）、ユーザー（users）の4つの主要テーブルを使用し、それぞれ外部キーで関連付けています。
・各テーブルには論理削除用のフラグ（delete_flag, delete_day）を設けています。
・権限管理のため、usersテーブルにはauth_typeカラムを持っています。
・初期データ投入は、動作確認・テストのため管理者ユーザー、一般ユーザー、掲示板、スレッド、投稿の順に行います。

補足：詳細な手順について
詳細な手順は「8. ビルド＆実行方法（詳細設定編）」で説明します。

以上

7. アプリケーション設定

7-1. Struts2の設定ファイル説明

DB接続設定ファイルの説明（db-config.properties）

◇役割:
データベース接続のための設定ファイルで、JDBCドライバ・URL・ユーザー名・パスワードが記述されます。
場所:
/WEB-INF/classes/db-config.properties

デフォルト内容:
db.url=jdbc:mysql://localhost:3306/your_database_name
db.username=your_db_username
db.password=your_db_password

編集ポイント:
db.url: データベース名やホスト名が異なる場合、ここを修正。
db.username/db.password: 環境に応じたDBユーザー情報に書き換える。

例:
db.url=jdbc:mysql://192.168.1.100:3306/your_db_name
db.username=your_user
db.password=your_password

・web.xmlの説明

◇役割:
アプリケーション起動時に最初に読み込まれる設定ファイルで、
Struts2フレームワークの起動設定や初期画面（login.jsp）の指定も行います。
場所:
/WEB-INF/web.xml

◇ポイント:
・Struts2フィルターの設定:
<filter>
<filter-name>struts2</filter-name>
<filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>

◇初期画面（login.jsp）への遷移設定:
基本的にはindex.htmlなどから自動的に遷移するための設定やサーブレットコンテナの起動時に有効になるポイントです。

・struts.xml設定ファイルの説明

◇役割:
Struts2はリクエストURLに応じて適切なアクションクラスを呼び出し、
そのアクションの結果（result）によって次に表示する画面（JSPなど）を決定します。

◇初期動作の流れ
web.xmlではStruts2のフィルターが設定され、全てのリクエストはStruts2の制御下に入ります。

<filter>

<filter-name>struts2</filter-name>
<filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
これにより、初回アクセスではlogin.jspが表示されます。

◇アクションと画面遷移の例
ログインフォームでは：
<s:form action="login" method="post">
→ /login.action にPOST送信されます。

struts.xml:
<action name="login"
class="com.company.bulletinboard.action.login.LoginAction">
<result name="login">/login.jsp</result>
<result name="admin">/view/ManagementMenu.jsp</result>
<result name="user">/view/userPortal.jsp</result>
<result name="input">/login.jsp</result>
</action>
・アクション名loginの呼び出しに対し、認証処理を実施。
・結果（result）に応じて管理画面または利用者ポータルに遷移。

◇ルーティングとリダイレクト
Struts2はアクションからアクションへの内部転送やリダイレクトも可能です。
たとえば、CancelAction内でsuccessの場合にListActionにリダイレクトする場合は以下のように設定します：
<action name="cancel" class="com.company.bulletinboard.action.CancelAction" method="mainProc">
<result name="success" type="redirect">list</result>
</action>
このように、
同じアクション内で画面遷移するパターン
別アクションへリダイレクトして処理を引き継ぐパターン
を柔軟に使い分けることができます。

以上

7. アプリケーション設定

7-2.補足:ビルド・依存管理(pom.xml、.classpath)

この補足では、アプリケーションのビルドおよび依存管理に関わる重要な設定ファイルであるpom.xml と .classpath、
そしてそれらを支えるビルドツールMaven について説明します。

・Maven とは？

◇役割:
Maven(メイヴン)は、Java のプロジェクト管理ツールであり、
以下のような役割を持っています。
・必要なライブラリ(依存関係)の自動ダウンロード・管理
※ダウンロードされたライブラリは、通常~/.m2/repository (Maven のローカルリポジトリ)に保存されます。
・プロジェクトのビルド(コンパイル、テスト、パッケージングなど)の自動化
・複数人、複数環境での開発をスムーズにする共通ルールの提供

特に、Struts2 や Spring のようなフレームワークを使う場合、
依存ライブラリの数が多くなりがちですが、Maven を使えば
それらを pom.xml という1つのファイルで一元管理できます。
これにより、jar ファイルを個別にダウンロード・配置する手間が省け、
開発・運用の効率が大幅に向上します。

・1.pom.xml(Maven 用ビルド定義ファイル)

◇役割:
pom.xml は Maven プロジェクトの中心的な設定ファイルで、使用するライブラリやプラグイン、依存関係のバージョンを記述します。
例:Struts2 の依存関係の記述
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts2-core</artifactId>
<version>2.5.26</version>
</dependency>
これにより、Maven コマンド(例:mvn install)や Eclipse の Maven 連携を通じて、必要なライブラリが自動的にダウンロードされます。

・2.classpath(Eclipse プロジェクト設定ファイル)

◇役割:
.classpath は Eclipse プロジェクトのビルドパスを管理する設定ファイルです。
Maven プロジェクトの場合、pom.xml を更新した後に mvn eclipse:eclipse コマンドや
Eclipse の「Maven → Update Project」機能を使用することで、.classpath に依存ライブラリ情報が自動的に反映されます。
例:
<classpathentry kind="lib" path="M2_REPO/org/apache/struts/struts2-core/2.5.26/struts2-core-2.5.26.jar" />
※直接 .classpath を編集することは原則ありません。Maven との連携で管理します。

・実際の運用の流れ

・pom.xml に必要なライブラリを忘れると、実行時に ClassNotFoundException などのエラーが発生する可能性があります。

以上

8. ビルド＆実行方法（詳細設定編）

8-1.git clone 詳細

※かならずクローン用のフォルダを事前にご用意ください。
※以下の説明はWindowsにてデスクトップに「GitClones」フォルダが作成済みであることを前提としています。
※Git bushかコマンドプロンプトにて、コマンドを実効します。

①クローン用フォルダへ移動する
$ cd Desktop/GitClones/

②クローン用フォルダへ移動できたことを確認する
$ pwd
/c/users/ユーザー名/Desktop/GitClones

③クローンコマンドを実効する
$ git clone https://github.com/juehara-crypto/Portfolio_Bulletinboard.git

以下、実行ログ
Cloning into 'Portfolio_Bulletinboard'...
remote: Enumerating objects: 349, done.
remote: Counting objects: 100% (349/349), done.
remote: Compressing objects: 100% (259/259), done.
remote: Total 349 (delta 63), reused 349 (delta 63), pack-reused 0 (from 0)
Receiving objects: 100% (349/349), 26.19 MiB | 6.74 MiB/s, done.
Resolving deltas: 100% (63/63), done.
Updating files: 100% (387/387), done.

④クローン結果を確認する
$ ls -l
total 4 drwxr-xr-x 1 ユーザーフォルダ名 197611 0 May 10 14:27 Portfolio_Bulletinboard/
※上記の通り、「GitClones/」フォルダ内でクローン実行後、「Portfolio_Bulletinboard/」がダウンロードされている

以上

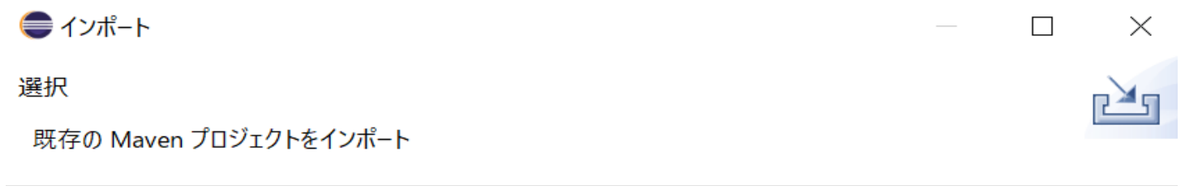
8. ビルド＆実行方法（詳細設定編）
8-2.Eclipseインポート詳細

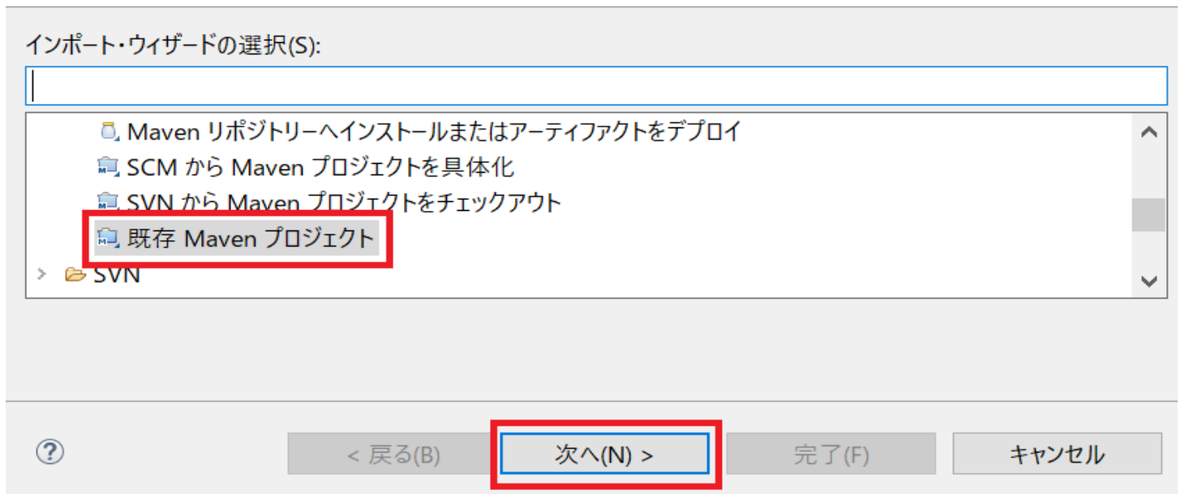
①クローンされたプロジェクトの移動
リポジトリがクローンされた「GitClones」内の「GitClones\Portfolio_Bulletinboard\03.プログラム\Bulletinboard」のフォルダを、Eclipseの「workspace」へ移動。

②Eclipseのプロジェクトのインポートを実効

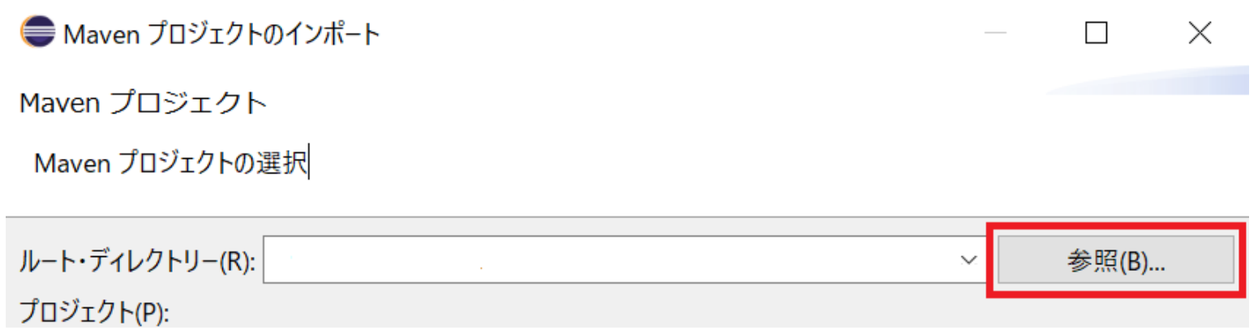
②-1.Eclipseのファイルから「インポート」をクリック

②-2.「インポート」の画面の一覧から「Maven」階層の「既存Mavenプロジェクト」を選択して「次へ」をクリックする

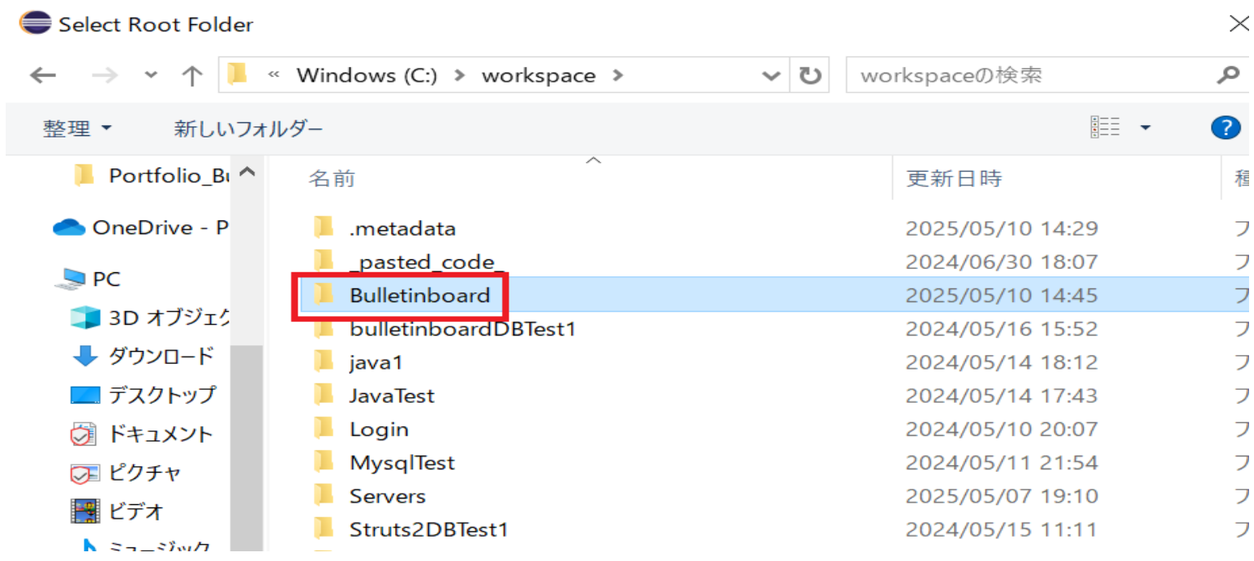


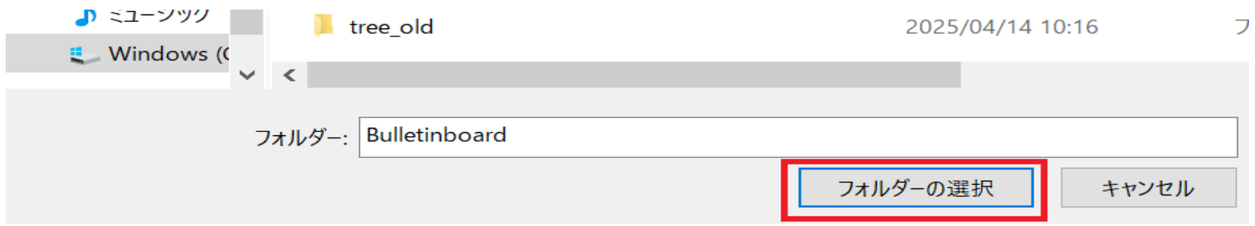


②-3.「Mavenプロジェクトフォルダ」の「ルート・ディレクトリ(R)」の「参照」をクリックする

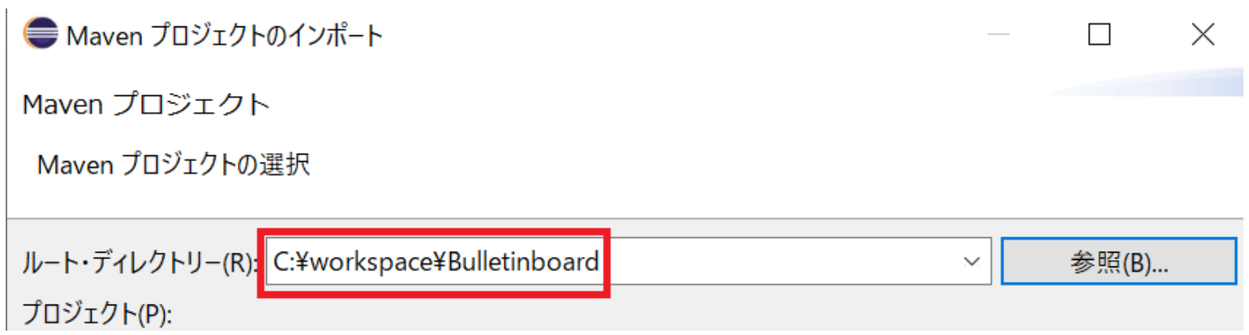


2-④. 先ほどワークスペースに移動した「Bulletinboard」を指定して「フォルダの選択」をクリックする

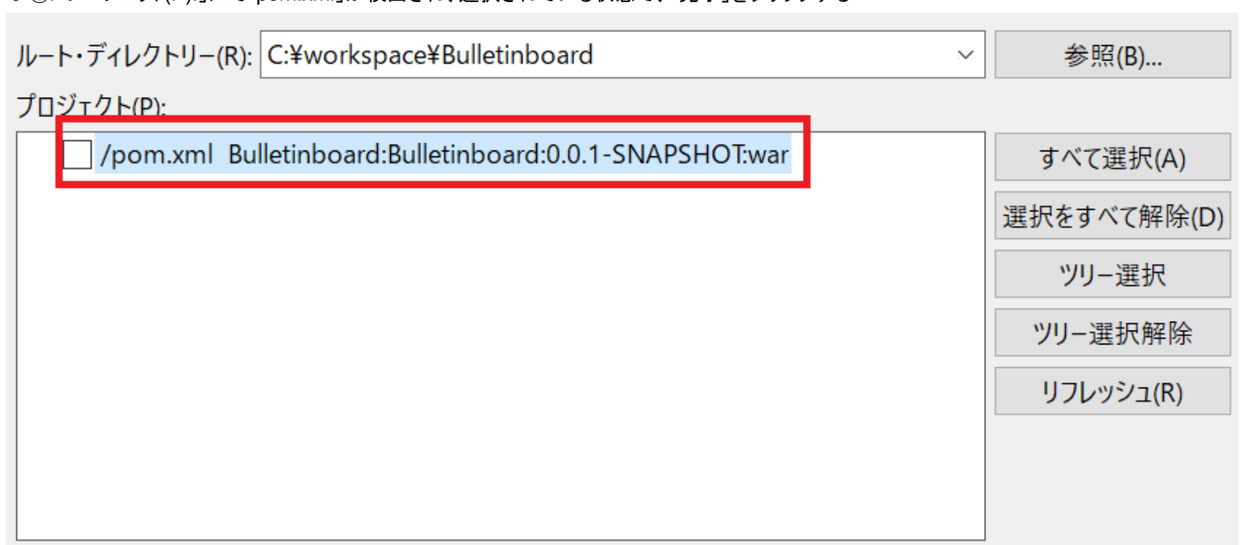




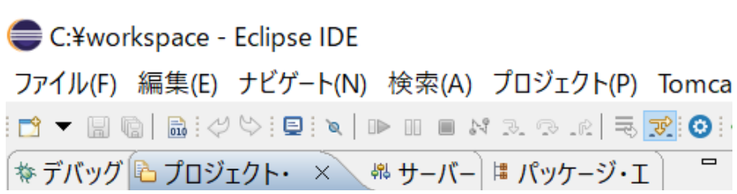
2-⑤.「ルートディレクトリ(R):」が指定したパスになります。

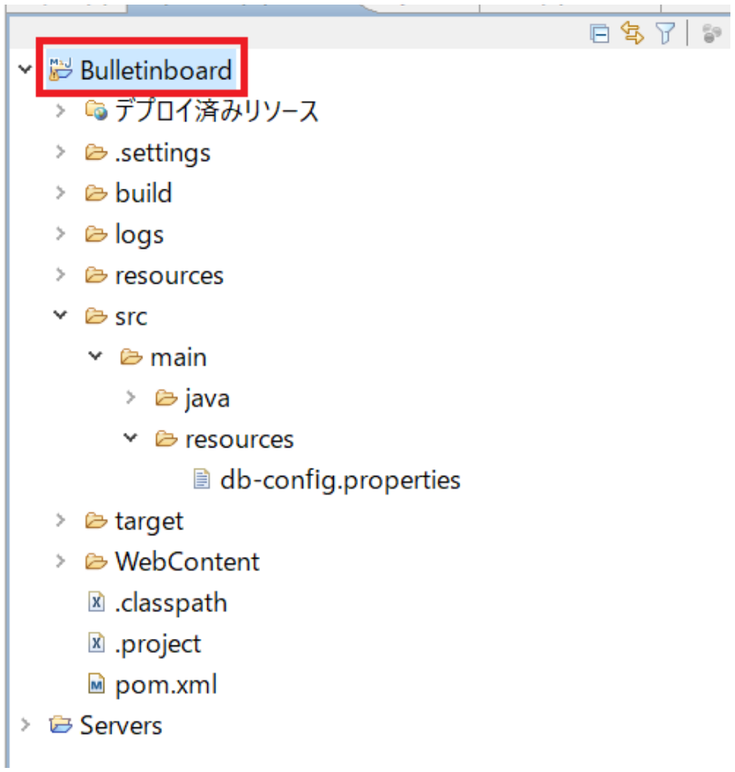


3-⑥.「プロジェクト(P):」にて「pom.xml」が検出され、選択されている状態で、「完了」をクリックする

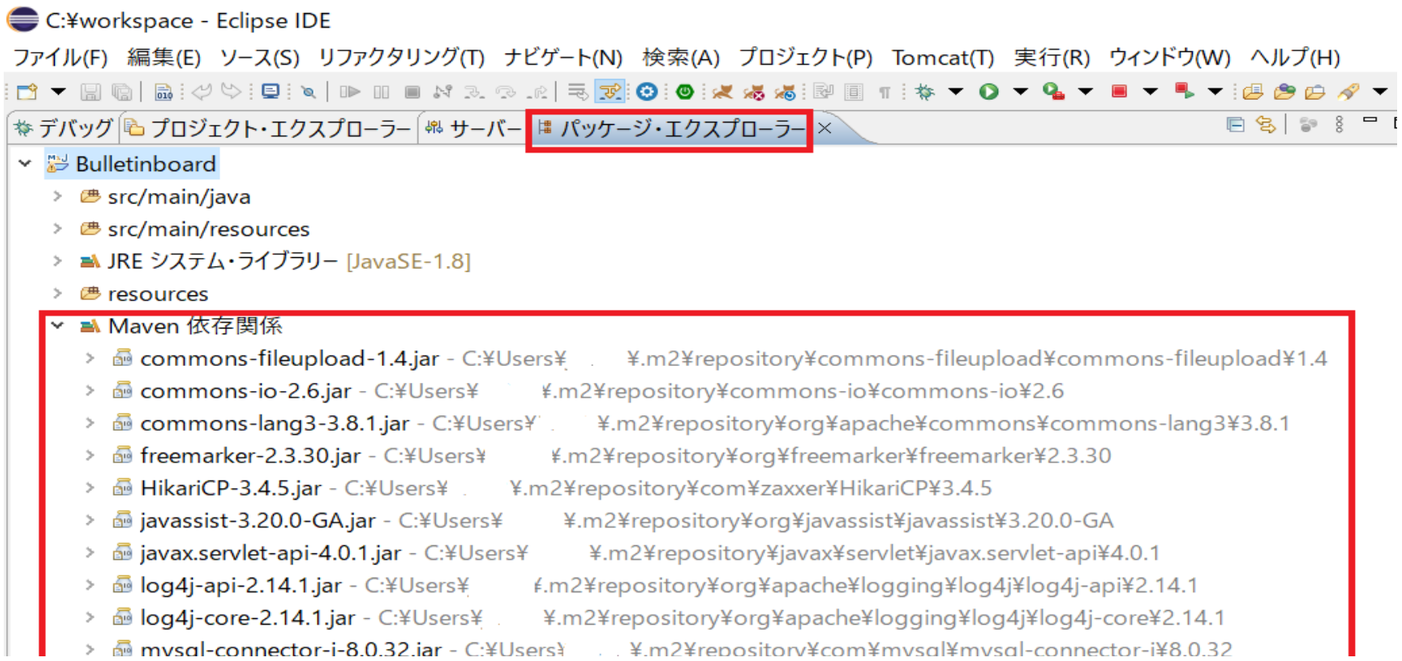


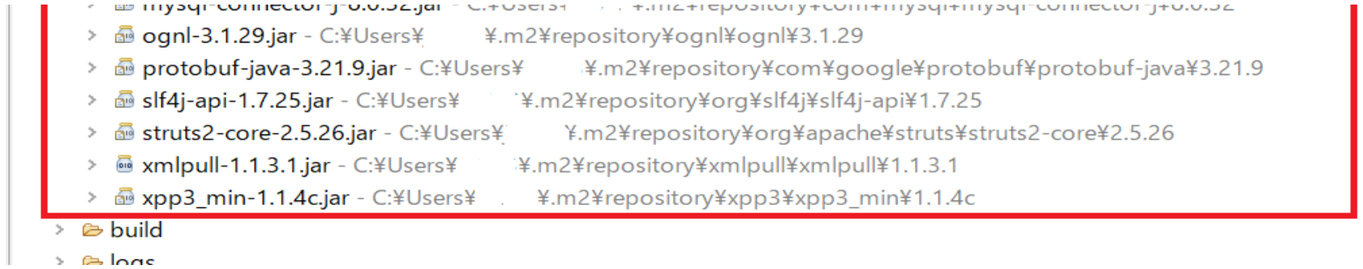
2-⑦.プロジェクトエクスプローラーに「Bulletinboard」が表示される





3.Maven依存関係の確認
パッケージエクスプローラーから「Bulletinboar」→「Maven依存関係」を開き以下のjarファイルが自動で入っていることを確認。





4.db-config.propertiesの変更
Bulletinboard/src/main/resources/db-config.propertiesの以下の内容を自分の環境の値に変更してください。
(例)
db.url=jdbc:mysql://localhost:3306/bulletinboard_db?useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&logger=com.mysql.cj.log.StandardLogger&profileSQL=true
db.username=root
db.password=pass

上記設定の後、以下を実効して下さい。
「プロジェクトのクリーン」を実効
.「Maven cleanを実効」
「Maven installを実効」

以上

8. ビルド＆実行方法（詳細設定編）

8-3-1.データベース・テーブル追加 SQL

◇背景と目的

本アプリケーションは、ユーザーが参加できる掲示板システムを提供します。掲示板は次の3階層の親子関係で構成されています：

・掲示板（Bulletin Board）

　　・掲示板ごとに複数のスレッド（Thread）を持ちます。

・スレッド（Thread）

　　・スレッドごとに複数の投稿（Post）を持ちます。

・投稿（Post）

この階層構造に対応するため、データベースおよび各テーブルを以下の通り作成します：

・データベース：bulletinboard_db

・テーブル：

　　・users：ユーザー情報を管理

　　・bulletinboard：掲示板情報を管理

　　・threads：スレッド情報を管理

　　・posts：投稿情報を管理

各テーブル間にはリレーション（外部キー制約）を設け、データの整合性を保つ構成としています。

◇設計補足

・usersテーブル：

掲示板システムでは「管理者」と「一般ユーザー」の区別が必要なためauth_typeカラムを設けています。

これにより、システム内でユーザー権限に応じた画面遷移・操作制御を実現しています。

・delete_flagおよびdelete_day：

各テーブルには共通してdelete_flag（削除フラグ）とdelete_day（削除日付）が存在します。

これは論理削除を実装するためのものです。

データベース上で物理削除を行わず、削除済みデータも保持することで、データ復旧や監査対応を容易にします。

■データベース追加用SQL文

1-1.「bulletinboard_db」追加用SQL文

```
CREATE DATABASE bulletinboard_db
DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_bin;
```

1-2.「bulletinboard_db」の構造（作成後の確認結果）

```
mysql> SELECT schema_name, default_character_set_name, default_collation_name
    -> FROM information_schema.schemata
    -> WHERE schema_name = 'bulletinboard_db';
+------------------+----------------------------+------------------------+
| SCHEMA_NAME      | DEFAULT_CHARACTER_SET_NAME | DEFAULT_COLLATION_NAME |
+------------------+----------------------------+------------------------+
| bulletinboard_db | utf8mb4                    | utf8mb4_bin            |
+------------------+----------------------------+------------------------+
```

■テーブル追加用SQL文

・1-1.「bulletinboard」テーブル用 CREATE TABLE文

```
CREATE TABLE bulletinboard (
bulletinboard_id INT AUTO_INCREMENT PRIMARY KEY,
bulletinboard_title VARCHAR(191),
bulletinboard_content TEXT,
user_id INT,
bulletinboard_delete_flag INT,
bulletinboard_delete_day DATETIME(6),
FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

・1-2.「bulletinboard」のテーブル構造（作成後の確認結果）

```
mysql> show columns from bulletinboard;
```

```
+--------------------------+--------------+------+-----+---------+----------------+
| Field                    | Type         | Null | Key | Default | Extra          |
+--------------------------+--------------+------+-----+---------+----------------+
| bulletinboard_id         | int          | NO   | PRI | NULL    | auto_increment |
| bulletinboard_title      | varchar(191) | YES  |     | NULL    |                |
| bulletinboard_content    | text         | YES  |     | NULL    |                |
| user_id                  | int          | YES  | MUL | NULL    |                |
| bulletinboard_delete_flag | int          | YES  |     | NULL    |                |
| bulletinboard_delete_day | datetime(6)  | YES  |     | NULL    |                |
+--------------------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

・2-1.「posts」テーブル用 CREATE TABLE文

```
CREATE TABLE posts (
post_id INT AUTO_INCREMENT PRIMARY KEY,
post_content TEXT,
thread_id INT,
user_id INT,
post_delete_flag INT,
post_delete_day DATETIME(6),
post_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (thread_id) REFERENCES threads(thread_id),
FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

・2-2.「posts」の構造（作成後の確認結果）

```
mysql> show columns from posts;
+------------------+-------------+------+-----+-------------------+-------------------+
| Field            | Type        | Null | Key | Default           | Extra             |
+------------------+-------------+------+-----+-------------------+-------------------+
| post_id          | int         | NO   | PRI | NULL              | auto_increment    |
| post_content     | text        | YES  |     | NULL              |                   |
| thread_id        | int         | YES  | MUL | NULL              |                   |
| user_id          | int         | YES  | MUL | NULL              |                   |
| post_delete_flag | int         | YES  |     | NULL              |                   |
| post_delete_day  | datetime(6) | YES  |     | NULL              |                   |
| post_timestamp   | datetime    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------------+-------------+------+-----+-------------------+-------------------+
7 rows in set (0.00 sec)
```

・3-1.「threads」テーブル用 CREATE TABLE文

```
CREATE TABLE threads (
thread_id INT AUTO_INCREMENT PRIMARY KEY,
thread_title VARCHAR(191),
bulletinboard_id INT,
user_id INT,
thread_delete_flag INT,
thread_delete_day DATETIME(6),
FOREIGN KEY (bulletinboard_id) REFERENCES bulletinboard(bulletinboard_id),
FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

・3-2.「threads」の構造（作成後の確認結果）

```
mysql> show columns from threads;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| thread_id          | int          | NO   | PRI | NULL    | auto_increment |
| thread_title       | varchar(191) | YES  |     | NULL    |                |
| bulletinboard_id   | int          | YES  | MUL | NULL    |                |
| user_id            | int          | YES  | MUL | NULL    |                |
| thread_delete_flag | int          | YES  |     | NULL    |                |
| thread_delete_day  | datetime(6)  | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

・4-1.「users」テーブル用 CREATE TABLE文

```
CREATE TABLE users (
user_id INT AUTO_INCREMENT PRIMARY KEY,
user_name VARCHAR(20),
password VARCHAR(60),
auth_type INT,
delete_flag INT,
delete_day DATETIME(6)
);
```

・4-2.「users」の構造（作成後の確認結果）

```
mysql> show columns from users;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| user_id     | int         | NO   | PRI | NULL    | auto_increment |
| user_name   | varchar(20) | YES  |     | NULL    |                |
| password    | varchar(60) | YES  |     | NULL    |                |
| auth_type   | int         | YES  |     | NULL    |                |
| delete_flag | int         | YES  |     | NULL    |                |
| delete_day  | datetime(6) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

以上

8. ビルド＆実行方法（詳細設定編）

8-3-2.利用者ポータル用　初期データ投入 SQL＆解説

◇背景と目的

本アプリケーションは：

●掲示板（Bulletin Board） → スレッド（Thread） → 投稿（Post）

という親子関係のデータ構造です。動作確認やテスト時に画面遷移＆データ表示が正しく行えるよう、初期データの整備が必要です。

◇初期データ投入のポイント

・ログイン後すぐ確認できる動作保証のため、親データから子データまで一式揃える。

・ユーザー個別のスレッド／投稿一覧も表示確認したいため、ユーザーごとのデータも投入。

※ID（Auto Increment）について

通常は ID を指定せず、DB に自動で採番させるべきです。

このSQL例では説明を分かりやすくするためにID を明示していますが

・テスト投入時：このまま実行可能（初期環境ならOK）

・本番環境：ID の指定は避け、LAST_INSERT_ID()などで取得＆次のINSERTに利用

※Auto Increment 環境なら：

・user_idなどの列をINSERT文から外して実行

・その後の親ID・外部キーに LAST_INSERT_ID()や確認したIDを入れる

◇データ作成の流れ

1.管理者権限ユーザー作成

2. 一般権限ユーザー作成

3.掲示板作成（管理者ユーザーに紐付け）

4.スレッド作成（掲示板ID＋一般ユーザーIDで親子関係確立）

5.投稿作成（スレッドID＋一般ユーザーIDで親子関係確立）

1.管理者権限ユーザー作成

```
INSERT INTO users (
user_id,
user_name,
password,
auth_type,
delete_flag,
delete_day
) VALUES (
1,
AdminUser1',
hashed_pw_1',                    -- パスワードはハッシュ済みにすること
1,                               -- 1 = 管理者権限
0,
2999-12-31 00:00:00'
);
```

各カラムの説明

項目	カラム名	説明
1列目	user_id	ユーザーID（整数、ユニーク）
2列目	user_name	ユーザー名（※特殊文字禁止）
3列目	password	パスワード（英字・数字・記号含む）
4列目	auth_type	権限種別（管理者=1、一般ユーザー=0など）
5列目	delete_flag	削除フラグ（0=有効、1=削除済み）

6列目	delete_day	削除日（yyyy-MM-dd HH:mm:ss形式）

制約事項

user_name:	以下の特殊文字は使わないでください： @, #, $, %, &, *, !, ?, /, \, =, +, ^, |, <, >, {, }, [,], ~
password:	英字・数字・記号を含めてください
delete_day:	yyyy-MM-dd HH:mm:ss の形式で指定。

2. 一般権限ユーザー作成

```
INSERT INTO users (
user_id,
user_name,
password,
auth_type,
delete_flag,
delete_day
) VALUES (
2,
GeneralUser1',
hashed_pw_2',                    -- パスワードはハッシュ済みにすること
0,                               -- 0 = 一般権限ユーザー
0,
2999-12-31 00:00:00'
);
```

各カラムの説明

項目	カラム名	説明
1列目	user_id	ユーザーID（整数、ユニーク）
2列目	user_name	ユーザー名（※特殊文字禁止）
3列目	password	パスワード（英字・数字・記号含む）
4列目	auth_type	権限種別（管理者=1、一般ユーザー=0など）
5列目	delete_flag	削除フラグ（0=有効、1=削除済み）
6列目	delete_day	削除日（yyyy-MM-dd HH:mm:ss形式）

制約事項

user_name:	以下の特殊文字は使わないでください： @, #, $, %, &, *, !, ?, /, \, =, +, ^, |, <, >, {, }, [,], ~
password:	英字・数字・記号を含めてください
delete_day:	yyyy-MM-dd HH:mm:ss の形式で指定。

3.掲示板作成（管理者ユーザーに紐付け）

```
INSERT INTO bulletinboard (
bulletinboard_id,
bulletinboard_title,
bulletinboard_content,
bulletinboard_delete_flag,
bulletinboard_delete_day,
user_id
) VALUES (
1,
テスト掲示板1',
これはテスト掲示板の説明です。',
0,
NULL,
1                                -- ①の操作で作成した管理者権限ユーザーのID
);
```

各カラムの説明

項目	カラム名	説明
1列目	bulletinboard_id	掲示板ID（整数、ユニーク）
2列目	bulletinboard_title	掲示板タイトル（255文字以内）
3列目	bulletinboard_content	掲示板の説明内容
4列目	bulletinboard_delete_flag	削除フラグ（0=有効、1=削除済み）
5列目	bulletinboard_delete_day	(yyyy-MM-dd HH:mm:ss形式／NULL可）
6列目	user_id	掲示板作成者のユーザーID（外部キー）

制約事項

]bulletinboard_title:	bulletinboard_title: 必須、長さは255文字以内
bulletinboard_delete_day:	yyyy-MM-dd HH:mm:ss 形式または NULL

4.スレッド作成（掲示板ID＋一般権限ユーザーIDで親子関係確立）

```
INSERT INTO threads (
thread_id,
thread_title,
bulletinboard_id,
user_id,
thread_delete_flag,
thread_delete_day
) VALUES (
1,
テストスレッド1',
1,                                  -- ③の操作で作成した掲示板のID
2,                                  -- ②の操作で作成した一般権限ユーザーのID
0,
NULL
);
```

各カラムの説明

項目	カラム名	説明
1列目	thread_id	スレッドID（整数、ユニーク）
2列目	thread_title	スレッドタイトル（255文字以内）
3列目	bulletinboard_id	掲示板ID（親掲示板のID、外部キー）
4列目	user_id	作成者ユーザーID（外部キー）
5列目	thread_delete_flag	削除フラグ（0=有効、1=削除済み）
6列目	thread_delete_day	削除日（yyyy-MM-dd HH:mm:ss形式／NULL可）

制約事項

thread_title: 必須、長さは255文字以内	bulletinboard_title: 必須、長さは255文字以内
bulletinboard_id: 存在する掲示板のID	yyyy-MM-dd HH:mm:ss 形式または NULL
thread_delete_day: yyyy-MM-dd HH:mm:ss	形式または NULL

5.投稿作成（スレッドID＋一般権限ユーザーIDで親子関係確立）

```
INSERT INTO posts (
post_id,
post_content,
thread_id,
user_id,
post_delete_flag,
post_delete_day,
post_timestamp
) VALUES (
```

```
1,
これはテスト投稿の内容です。',
1,                                  -- ④の操作で作成したスレッドのID
2,                                  -- ②の操作で作成した一般権限ユーザーのID
0,
NULL,
NOW()
);
```

各カラムの説明

項目	カラム名	説明
1列目	post_id	投稿ID（整数、ユニーク）
2列目	post_content	投稿内容
3列目	thread_id	スレッドID（親スレッドのID、外部キー）
4列目	user_id	投稿者ユーザーID（外部キー）
5列目	post_delete_flag	削除フラグ（0=有効、1=削除済み）
6列目	post_delete_day	削除日（yyyy-MM-dd HH:mm:ss形式／NULL可）
7列目	post_timestamp	投稿日時（タイムスタンプ）

制約事項

post_content:	必須
thread_id:	存在するスレッドのID
post_delete_day:	yyyy-MM-dd HH:mm:ss 形式または NULL
post_timestamp:	NOW() などの現在時刻を推奨

◇動作確認のイメージ

操作	ログインユーザー	確認できる画面
掲示板管理	管理者	掲示板管理画面
スレッド一覧	一般ユーザー	利用者ポータル（掲示板内スレッド）
投稿一覧	一般ユーザー	スレッド詳細画面内

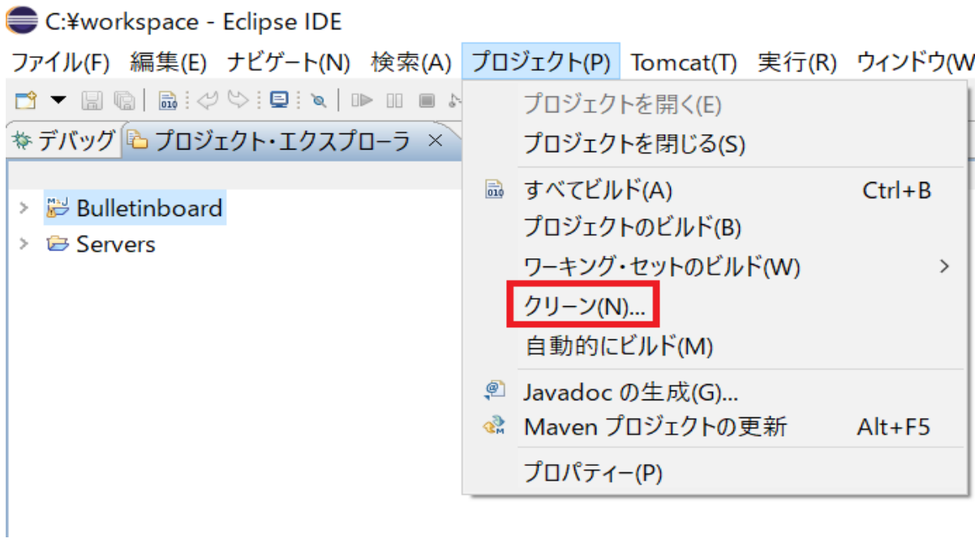
以上

9. アプリケーション起動

9-1.Tomcat起動

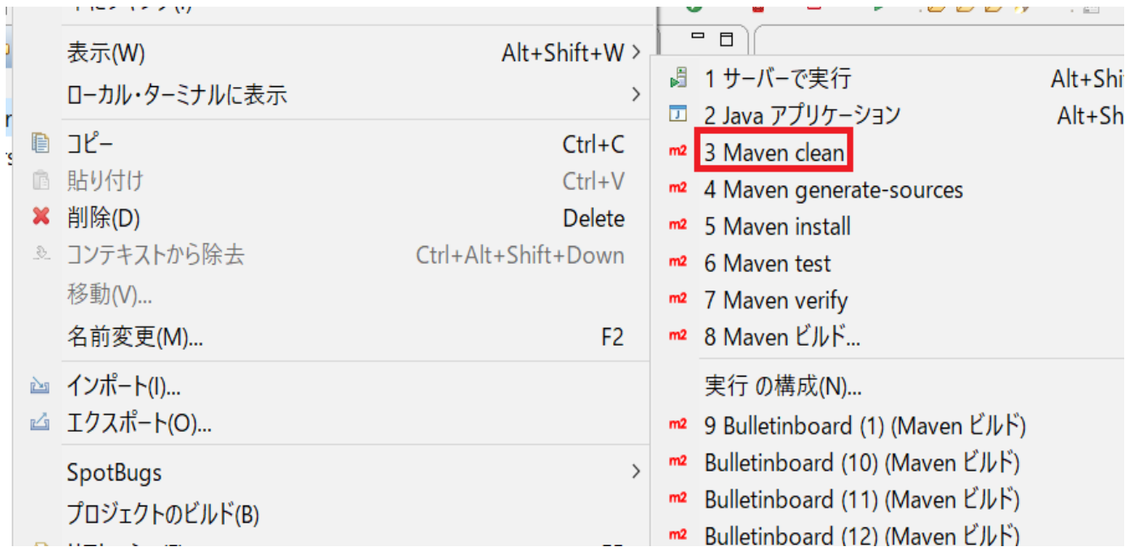
※起動の前に以下を実効する

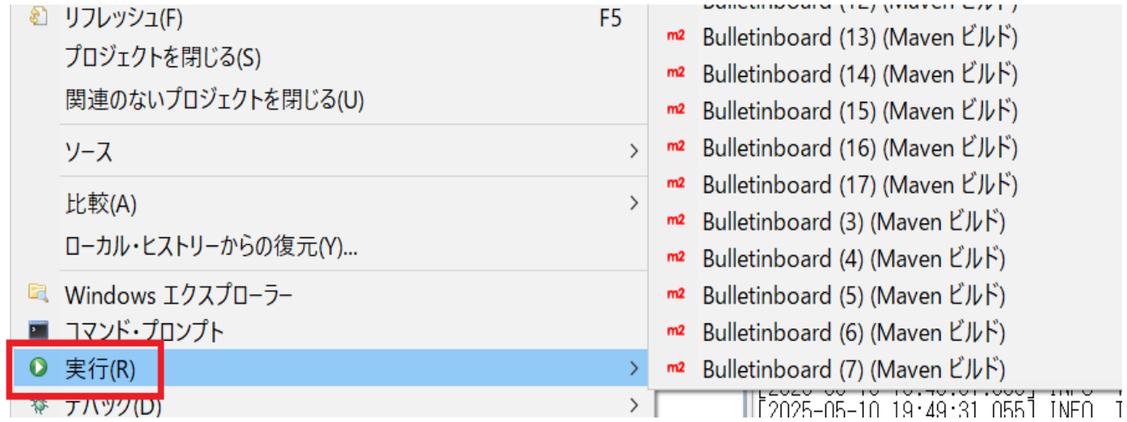
1-1.プロジェクトのクリーンを実効



1-2.Maven cleanを実効

プロジェクトを右クリックして「実効」を選択し、「Maven clean」をクリックする



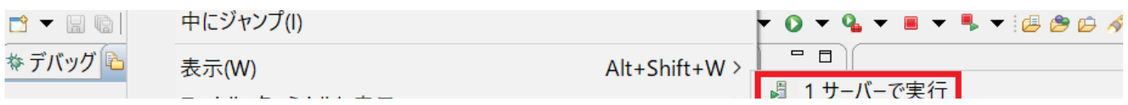


1-3.Maven installを実効

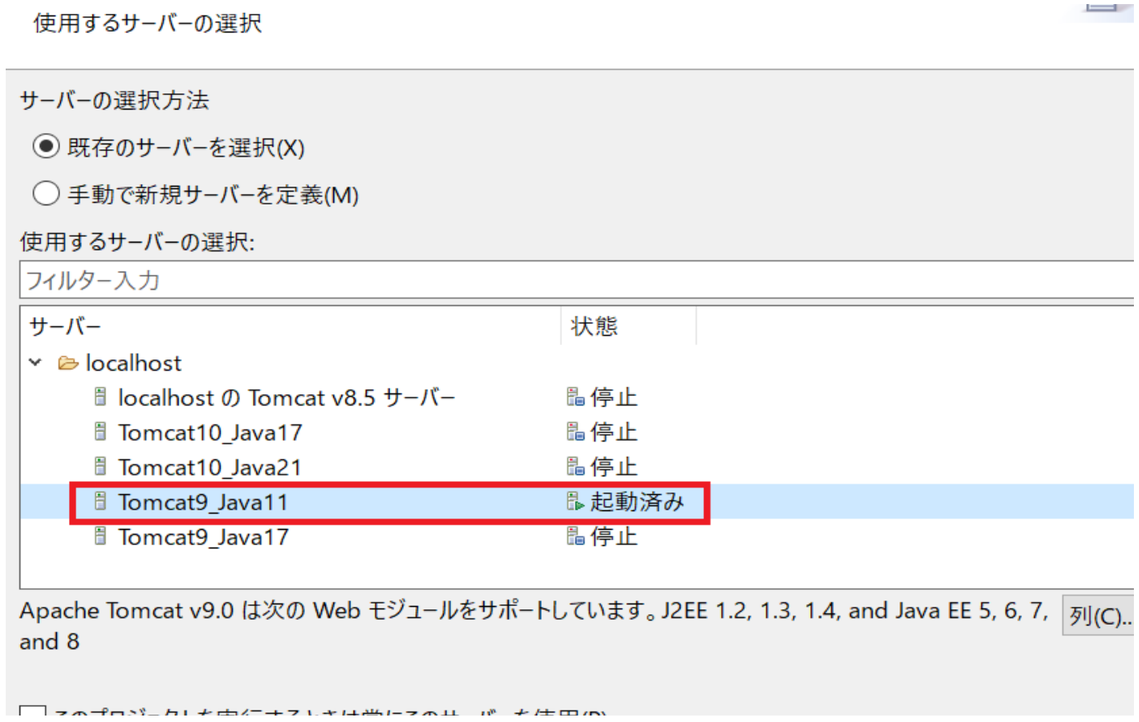
プロジェクトを右クリックして「実効」を選択し、「Maven install」をクリックする

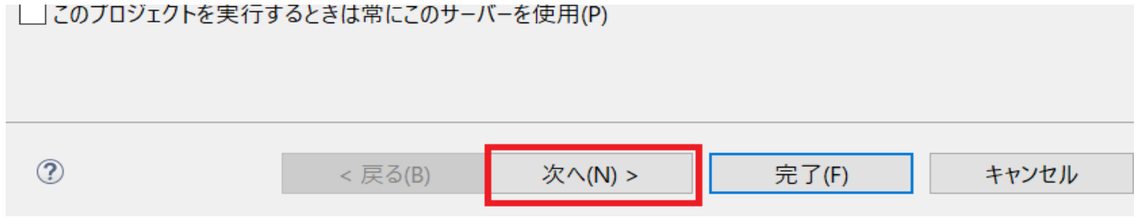
2.Tomcatを実効

2-1.プロジェクトを右クリックして「実効」を選択し、「 サーバーで実行」をクリックする

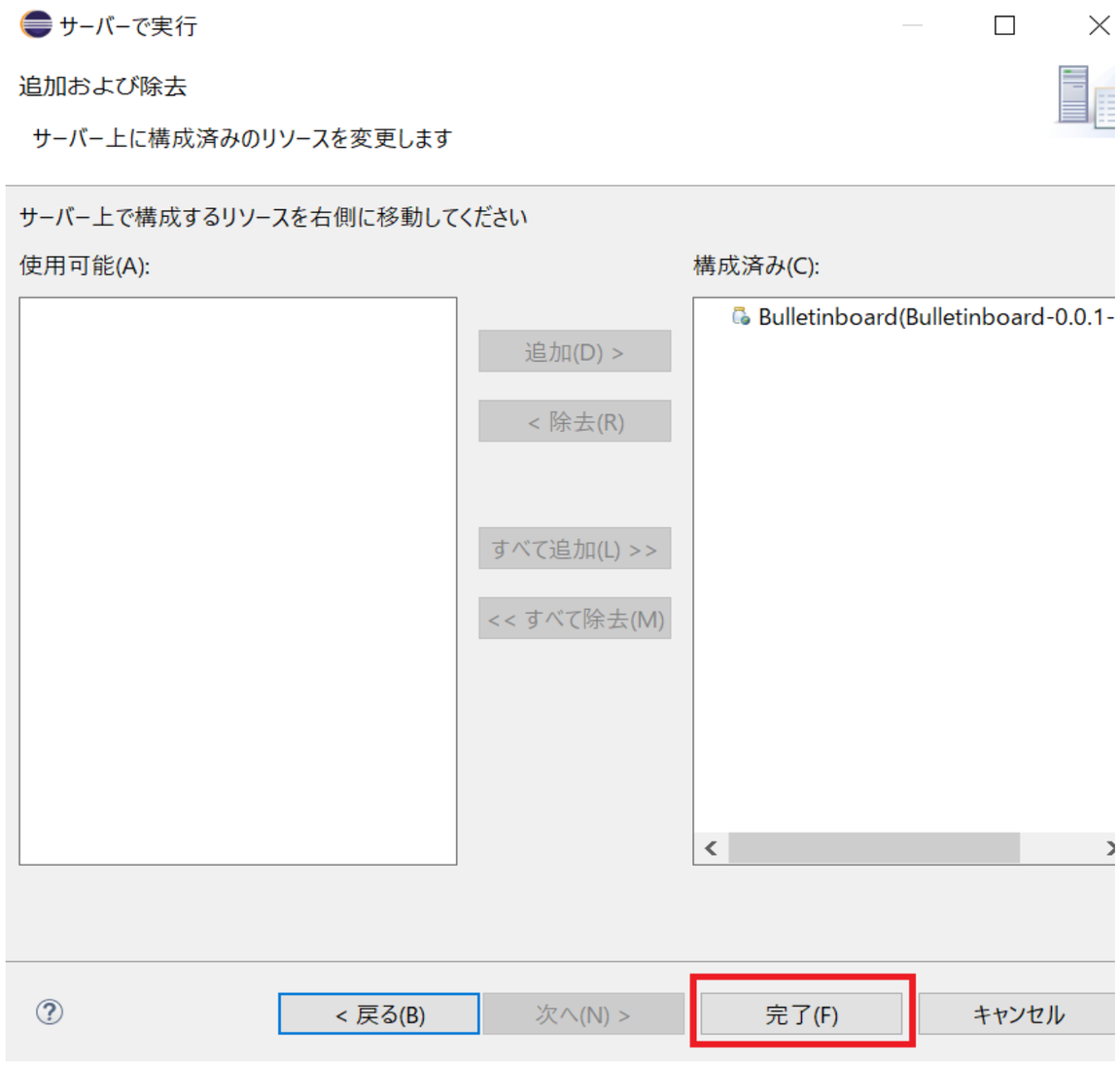


2-2.「仕様するサーバの選択」画面にてTomcatを選択し、「次へ」をクリックする

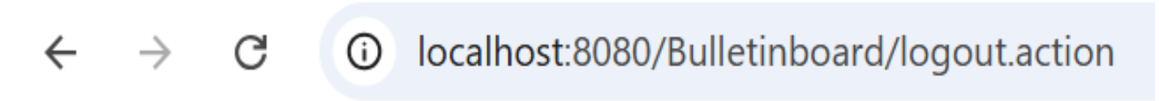




2-3.次の画面で「完了」ボタンをクリックする。



2-4.ログイン画面が表示されます



ログイン画面

ユーザー名:

パスワード:

ログイン

以上

10. アプリケーション設定

補足：HikariCP設定・モニタリング

DatabaseConnectionManager 設定概要

・HikariCP を使用し、JDBC 接続・接続プール管理を行う。

・接続情報はdb-config.properties から読み込む。

- db.url
- db.username
- db.password

・プール設定はDatabaseConnectionManager 内で直接設定している。

- 最大接続数：10 (`setMaximumPoolSize`)
- アイドルタイムアウト：30秒 (`setIdleTimeout`)
- 最大寿命：30分 (`setMaxLifetime`)

・接続取得/接続クローズ：

- 接続取得：Connection conn = dataSource.getConnection();
- 接続クローズ：conn.close(); // プールに返却

DatabaseConnectionManager の設定抜粋：

```
※以下、DatabaseConnectionManager の設定抜粋
HikariConfig config = new HikariConfig();
config.setJdbcUrl(url);
config.setUsername(username);
config.setPassword(password);
config.setDriverClassName("com.mysql.cj.jdbc.Driver");
config.setMaximumPoolSize(10);       // プール内の最大接続数
config.setIdleTimeout(30000);        // アイドル接続のタイムアウト（ms）
config.setMaxLifetime(1800000);      // 接続の最大寿命（ms）
```

・モニタリング処理：

DatabaseConnectionManager 内の startMonitoring() メソッドを使用し、
5秒ごとに接続プール状態をログ出力。

・補足：:

上記の接続設定は開発者のローカル環境（localhost、データベース名 bulletinboard_db）を前提としています。
他の環境（例：本番サーバ、他の開発者環境）では以下の値を環境に応じて修正してください。
localhost → 使用するDBサーバのホスト名またはIPアドレス
bulletinboard_db → データベース名
db.username、db.password → 使用するDBユーザー名・パスワード

以上

11. トラブルシューティング　Q&A

Q1. Maven依存関係は更新しましたが、 db-config.properties が target\classes にコピーされません。

A.
Eclipse環境では、src/main/resources が自動的にソースフォルダとして認識されないことがあります。
その結果、db-config.properties などの設定ファイルがビルド時にtarget\classes 配下にコピーされず、
アプリケーションが正しく動作しなくなる場合があります。

対処手順:
1.Eclipseのプロジェクトエクスプローラーでプロジェクトを右クリックし、
「プロパティ」→「Javaのビルド・パス」を開く。
2.「ソース」タブを選択。
3.src/main/resources をソースフォルダとして追加する。
これで、db-config.properties はMavenのビルド時に target\classes に正しく反映されるようになります。

Q2. MySQLドライバーを手動で「 Javaのビルド・パス」に追加したところ、実行時に　HikariCPなどが正常に動作しなくなりました。

A.
この現象は、**Maven依存関係と手動で追加したライブラリが競合** して発生します。特に、mysql-connector-j などのライブラリはMavenの pom.xml でバージョン管理されているため、
手動で「Javaのビルド・パス」に追加するのは NGです
●絶対にやってはいけない例：
・mysql-connector-j をビルド・パスに手動追加
・そのまま Maven の「プロジェクトの更新」を実行
これをやると、ライブラリが二重登録されてクラスパスが衝突しHikariCPなどの接続プールが正常動作しなくなります。
正しい手順:
・必ず pom.xml だけで依存関係を管理する。
・Eclipse利用時は、「プロジェクトの更新（Update Project）」を実行して、Maven依存性を自動解決させる。

前提（非エンジニアの方向け補足）:
このアプリケーションは、Eclipse上でMavenという仕組みを使って、必要なライブラリ（＝プログラムを動かすための追加パーツ）を自動でインストール・管理する前提になっています。
つまり、
・Maven = 自動で必要なものを揃えてくれる便利なツール
・pom.xml = どのライブラリを使うかをまとめた「リスト」
です。
本来はこの「リスト（pom.xml）」だけをメンテナンスすればOKなのですが、手動でライブラリを追加してしまうと、Mavenの仕組みと衝突して、動かなくなることがあるので注意が必要です。

Q3. EclipseのMavenプロジェクトをクローンしただけでは動かないのですが？

A.
GitHubなどからプロジェクトをクローンした後は、以下の操作を忘れずに実施してください。
1.Eclipseでプロジェクトをインポート。
2.右クリック → Maven → プロジェクトの更新（Update Project） を実行。
これで pom.xml に従った依存関係がすべて解決されます。