SpringBootアプリでLiquibaseを使ってみる。

1. Spring Initializr で [Liquibase Migration] を選択す
 赤字の依存関係が追加される

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.5.3</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.example</groupId>
	<artifactId>demo</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>demo</name>
	<description>Demo project for Spring Boot</description>
	<properties>
		<java.version>11</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter</artifactId>
		</dependency>
		<dependency>
			<groupId>org.liquibase</groupId>
			<artifactId>liquibase-core</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
</project>
```

2. Maven プラグインを追加
 -> liquibaseをmavenプラグインメニューから使えるようになる

```
			<plugin>
				<groupId>org.liquibase</groupId>
				<artifactId>liquibase-maven-plugin</artifactId>
				<configuration>

<propertyFile>src/main/resources/liquibase.properties</propertyFile>
				</configuration>
			</plugin>
```

3. プラグインからコマンド実行するための準備
src/main/resources/liquibase.properties

```
# updateやgenerateChangeLogの対象となるデータベース
url=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=before_cmsboot
driver=org.postgresql.Driver
username=admin
password=admin

# diffコマンド(差分抽出)を利用時に基準となるデータベース
referenceUrl=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=new_cmsboot
referenceDriver=org.postgresql.Driver
referenceUsername=admin
referencePassword=admin

# liquibaseがテーブル作成に利用する変更ログ(変更しないこと)
changeLogFile=src/main/resources/db/changelog/db.changelog-master.yaml

# generateChangeLogコマンドの出力先
# outputChangeLogFile=src/main/resources/db/changelog/db.changelog-generated.yaml
outputChangeLogFile=src/main/resources/db/changelog/db.changelog-generated.postgresql.sql

# dffコマンドの出力先
diffChangeLogFile=src/main/resources/db/changelog/db.changelog-diff-changeLog.yaml
# diffChangeLogFile=src/main/resources/db/changelog/db.changelog-diff-changeLog.postgresql.sql
```

src/main/resources/db/changelog/db.changelog-master.yaml

```
databaseChangeLog:
  - include:
      file: src/main/resources/db/changelog/db.changelog-0001.postgresql.sql
```

2. 公式サイトの説明はこちら

https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.data-initialization.migration-tool.liquibase

- POMにliquibase-coreを追加すると自動的にliquibaseが動く。
- 以下、デフォルトの動作
- メイン処理とテストの前にテーブルマイグレーションが実行される。(spring.liquibase.enabledで設定可能)
- liquibaseを使うと他のテーブル初期化機能との併用はできない。???
- DBの変更情報はdb/changelog/db.changelog-master.yamlから読み取られる(spring.liquibase.change-logで設定可能)
- YAMLに加えて、LiquibaseはJSON、XML、およびSQLの変更ログ形式もサポート
- DataSourceで設定されたデータベースに対して変更を加える。(複数の場合は@Primaryが設定されるもの)
- @LiquibaseDataSourceアノテーションで明示的に変更を加えるデータソースを指定できる。
- 外部プロパティ(pring.liquibase.[driver-class-name,url,user,password])を使って独自のデータベース接続を設定できる
- 全てのプロパティはこちら https://github.com/spring-projects/spring-boot/blob/v2.5.3/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseProperties.java

3. liquibaseを無効化する

```
spring.liquibase.enabled=false
```

4. 現在のデータベースからチェンジログを作成する

```
./mvnw liquibase:genarateChangeLog
```

5. 2つのデータベースの差分をチャンジログに出力する

```
./mvnw liquibase:diff
```

6. Use the Liquibase Hibernate Plugin
 -> 未調査
 -> Hibernateはモデル定義から自動的にテーブルを作成してくれるし、spring.jpa.hibernate.ddl-auto=update
 の設定で、自動的に変更も行ってくれるが、削除が反映しないなど、本運用での利用は想定されていない。
 このプラグインは、モデル定義に基づきテーブルを変更するのではなく、チェンジログを作成してくれる。

 https://github.com/liquibase/liquibase-hibernate

JPAを使用している開発環境でLiquibaseを利用するシナリオ

◯前提条件
 ・チェンジログのフォーマットはSQLを使う
 ※本当はyamlの方が良い(ロールバックさせるための手間が少ない)

◯基本的な方針
 ・開発中はJPAの機能で自動的にテーブルを更新する。
 ・開発が終わったら、Liquibaseで開発前と開発後の変更差分を取得し、チェンジログを作成する。
 ・JPAの自動更新を無効にし、テーブルを開発前の状態に戻して、Liquibaseで変更を適用する。
 ・テストする。

◯具体的な手順
※利用スキーマを「cmsboot」のケースで記載

1. 開発中のモードに切り替える(Liquibaseを無効にし、JPAのテーブル更新を有効にする)

```
# Hibernate
spring.jpa.hibernate.ddl-auto=update

# liquibaseの状態
# spring.liquibase.enabled=false
```

2. 開発前の状態を別スキーマに準備する。

(1)空のスキーマを準備する(DBに直接操作)

```
drop schema before_cmsboot cascade; -- 必要な場合
create schema before_cmsboot;
```

(2)Liquibaseの接続先を空のスキーマに変更する。

liquibase.properties

```
url=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=before_cmsboot
```

(3)テーブルを作成する

```
.mvnw liquibase:update
```

3. 開発終了後に開発前との変更差分を取得する。

(1)Diffを取得できる様にliquibaseの設定を変更する
liquibase.properties

```
# updateやgenerateChangeLogの対象となるデータベース、diffを使う場合は変更前の状態のデータベース
```

```
url=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=before_cmsboot

# diffコマンド(差分抽出)実行時、変更後の状態のデータベース
referenceUrl=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=cmsboot

# dffコマンドの出力先
diffChangeLogFile=src/main/resources/db/changelog/db.changelog-diff-changeLog.postgresql.sql
```

(2)Diffコマンドを実行

```
.mvnw liquibase:diff
```

-> db.changelog-diff-changeLog.postgresql.sql に変更差分が出力されていることを確認

4. 現在のスキーマを開発前の状態に戻す(空にする)

(1)現在のスキーマを空にする。※データが消えるのでバックアップも取得しておくこと

```
drop schema cmsboot cascade;
create schema cmsboot;
```

(2)Liquibaseの接続先を現在のスキーマに設定して、テーブルを作成
liquibase.properties

```
url=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=cmsboot
```

```
.mvnw liquibase:update
```

※バックアップしていたデータを復元、シーケンスの更新を忘れないように。（この作業が一番めんどくさい？）

5. 抽出した変更差分をLiquibaseのチェンジログに追加する。

db.changelog-0002.postgresql.sql を新規作成

```
※db.changelog-diff-changeLog.postgresql.sqlの中身をコピーする
★★ logicalFilePath: <ファイル名> を追加する(changeset単位)★★ ※1
★★ ロールバックアクションを追加する(未検証) ★★
```

※1 これをやらないとSpringBootとCLI(Maven)の両方でテーブル作成した場合に不一致と判断されてしまう。

```
-- changeset taku:1628246859811-1 logicalFilePath:db.changelog-0001.postgresql.sql
CREATE TABLE document_index (id BIGINT NOT NULL, no INTEGER NOT NULL, announce_date ...
```

db.changelog-master.yaml ･･･青字が追加

```
databaseChangeLog:
  - include:
      file: ./db.changelog-0001.postgresql.sql
      relativeToChangelogFile: true
  - include:
      file: ./db.changelog-0002.postgresql.sql
      relativeToChangelogFile: true
```

6. 開発モードを元に戻す

```
# Hibernate
# spring.jpa.hibernate.ddl-auto=update

# liquibaseの状態
spring.liquibase.enabled=false
```

7. SpringBoot起動時にテーブルが更新されることを確認する。

チェンジログにYaml形式を使いたい場合

拡張子を変更するだけ

```
# updateやgenerateChangeLogの対象となるデータベース、diffを使う場合は変更前の状態のデータベース
url=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=cmsboot
driver=org.postgresql.Driver
username=admin
password=admin

# diffコマンド(差分抽出)実行時、変更後の状態のデータベース
referenceUrl=jdbc:postgresql://localhost:5432/cmsboot?currentSchema=cmsboot
referenceDriver=org.postgresql.Driver
referenceUsername=admin
referencePassword=admin

# liquibaseがテーブル作成に利用する変更ログ(変更しないこと)
changeLogFile=src/main/resources/db/changelog/db.changelog-master.yaml

# generateChangeLogコマンドの出力先
outputChangeLogFile=src/main/resources/db/changelog/db.changelog-generated.yaml
# SQL形式の場合
# outputChangeLogFile=src/main/resources/db/changelog/db.changelog-generated.postgresql.sql

# dffコマンドの出力先
diffChangeLogFile=src/main/resources/db/changelog/db.changelog-diff-changeLog.yaml
# SQL形式の場合
# diffChangeLogFile=src/main/resources/db/changelog/db.changelog-diff-changeLog.postgresql.sql
```

```
databaseChangeLog:
  - include:
      file: ./db.changelog-0001.yaml
      relativeToChangelogFile: true
  - include:
      file: ./db.changelog-0002.yaml
      relativeToChangelogFile: true
  - include:
      file: ./db.changelog-0003.yaml
      relativeToChangelogFile: true
```

```
databaseChangeLog:
-  logicalFilePath: db.changelog-0001.yaml ※1
- changeSet:
    id: 1628246974739-1
    author: taku (generated)
    changes:
    - createTable:
        columns:
        - column:
            constraints:
              nullable: false
              primaryKey: true
              primaryKeyName: document_index_pkey
            name: id
            type: BIGINT
        - column:
```

```
            constraints:
```

※1 これを追加しないとSpringBootとCLI(Mavenコマンド)の両方でテーブル作成した場合に不一致と判断されてしまう。

(参考)diffコマンドを詳細な説明
https://docs.liquibase.com/tools-integrations/maven/commands/maven-diff.html

---

◯liquibase Mavenコマンド一覧

```
liquibase:changelogSync
  Marks all unapplied changes to the database as applied in the change log.

liquibase:changelogSyncSQL
  Generates SQL that marks all unapplied changes as applied.

liquibase:clearCheckSums
  Clears all checksums in the current changelog, so they will be recalculated
  next update.

liquibase:dbDoc
  Generates dbDocs against the database.

liquibase:diff
  Generates a diff between the specified database and the reference database.
  The output is either a report or a changelog depending on the value of the
  diffChangeLogFile parameter.

liquibase:dropAll
  Drops all database objects in the configured schema(s). Note that functions,
  procedures and packages are not dropped.

liquibase:futureRollbackSQL
  Generates the SQL that is required to rollback the database to current state
  after the next update.

liquibase:generateChangeLog
  Generates a changelog based on the current database schema. Typically used
  when beginning to use Liquibase on an existing project and database schema.

liquibase:help
  Display help information on liquibase-maven-plugin.
  Call mvn liquibase:help -Ddetail=true -Dgoal=<goal-name> to display parameter
  details.

liquibase:history
  Outputs history of deployments against the configured database.

liquibase:listLocks
  Lists all Liquibase updater locks on the current database.

liquibase:migrate
  Deprecated. Use the LiquibaseUpdate class or Maven goal "update" instead.

liquibase:migrateSQL
```

```
  Deprecated. Use {@link LiquibaseUpdateSQL} or Maven goal "updateSQL" instead.

liquibase:releaseLocks
  Removes any Liquibase updater locks from the current database.

liquibase:rollback
  Invokes Liquibase to rollback the database (and mark changesets as unapplied).
  The change sets to be rolled back are specified using attributes
  'rollbackCount', 'rollbackTag' and/or 'rollbackDate'

liquibase:rollbackOneChangeSet
  Reverts (rolls back) one non-sequential changeSet made during a previous
  change to your database. It is only available for Liquibase Pro users.

liquibase:rollbackOneChangeSetSQL
  Displays the SQL which will be executed when the corresponding
  rollbackOneChangeSet command is executed. This command does not perform the
  actual rollback. A Liquibase Pro license key is required.

liquibase:rollbackOneUpdate
  Rolls back all changesets from any specific update, if all changesets can be
  rolled back. By default, the last update is rolled back, but an optional
  deployentId parameter can target any update. (Liquibase Pro only).

liquibase:rollbackOneUpdateSQL
  Displays the SQL which will be executed when the corresponding
  rollbackOneUpdate command is executed. This command does not perform the
  actual rollback. A Liquibase Pro license key is required.

liquibase:rollbackSQL
  Generates the SQL that is required to rollback the database using one or more
  of the specified attributes 'rollbackCount', 'rollbackTag' and/or
  'rollbackDate'

liquibase:status
  Prints which changesets need to be applied to the database.

liquibase:tag
  Writes a Liquibase tag to the database.

liquibase:update
  Applies the DatabaseChangeLogs to the database. Useful as part of the build
  process.

liquibase:updateSQL
  Generates the SQL that is required to update the database to the current
  version as specified in the DatabaseChangeLogs.

liquibase:updateTestingRollback
  Applies the DatabaseChangeLogs to the database, testing rollback. This is done
  by updating the database, rolling it back then updating it again.
```

```
liquibase:changelogSync
  データベースに適用されていないすべての変更を変更ログに適用されたものとしてマークします。

liquibase:changelogSyncSQL
  すべての適用されていない変更を適用されたものとしてマークする SQL を生成します。
```

liquibase:clearCheckSums
  現在の変更ログのすべてのチェックサムをクリアします。
  次回の更新時に再計算されるようにします。

liquibase:dbDoc
  データベースに対して dbDocs を生成します。

liquibase:diff
  指定されたデータベースと参照データベースの間の diff を生成します。
  出力は、diffChangeLogFile パラメータの値に応じて、レポートまたは変更ログになります。
  diffChangeLogFile パラメータの値によって、レポートまたは変更ログが出力されます。

liquibase:dropAll
  設定されたスキーマのすべてのデータベースオブジェクトを削除します。関数、プロシージャ、パッケージは削除されないことに注意してください。
  関数、プロシージャ、パッケージは削除されません。

liquibase:futureRollbackSQL
  次の更新後にデータベースを現在の状態にロールバックするのに必要な SQL を生成します。
  生成します。

liquibase:generateChangeLog
  現在のデータベーススキーマに基づいて、変更ログを生成します。一般的には
  既存のプロジェクトとデータベーススキーマで Liquibase を使い始めるときに使用されます。

liquibase:ヘルプ
  liquibase-maven-plugin のヘルプ情報を表示します。
  パラメータの詳細を表示するには、mvn liquibase:help -Ddetail=true -Dgoal=<goal-name> を呼び出します。
  の詳細を表示します。

liquibase:history
  設定されたデータベースに対するデプロイメントの履歴を出力します。

liquibase:リストロック
  現在のデータベース上のすべての Liquibase updater のロックをリストアップします。

liquibase:マイグレート
  非推奨。代わりに LiquibaseUpdate クラスか Maven goal "update" を利用してください。

liquibase:migrateSQL
  非推奨。代わりに {@link LiquibaseUpdateSQL} か Maven goal "updateSQL" を使ってください。

liquibase:releaseLocks
  現在のデータベースから、すべての Liquibase 更新プログラムのロックを解除します。

liquibase:ロールバック
  Liquibase を起動し、データベースをロールバックします(そして、変更セットを未適用とします)。
  ロールバックされるチェンジセットは、属性
  rollbackCount', 'rollbackTag' および/または 'rollbackDate' 属性で指定されます。

liquibase:ロールバックワンチェンジセット
  以前のデータベースへの変更時に行われた非連続の変更セットを1つ戻します。
  戻します。これは、Liquibase Pro ユーザーのみが利用できます。

liquibase:ロールバック一つの変更セットSQL
  対応するロールバックワンチェンジセットコマンドが実行されたときに、実行される SQL を表示します。
  rollbackOneChangeSet コマンドが実行されたときに実行される SQL を表示します。このコマンドは、実際のロールバックを行いません。
  実際のロールバックは行いません。Liquibase Pro のライセンスキーが必要です。

Liquibase:ロールバックワンアップデート
  すべてのチェンジセットをロールバックできる場合、特定の更新からすべてのチェンジセットをロールバックします。
  ロールバックされます。デフォルトでは、最後の更新をロールバックしますが、オプションの
  deploymentId パラメーターは、任意の更新を対象とすることができます。(Liquibase Pro のみ)。

```
Liquibase:ロールバックワンアップデートSQL
  対応する rollbackOneUpdate コマンドが実行されたときに、実行される SQL を表示します。
  rollbackOneUpdate コマンドが実行されたときに実行される SQL を表示します。このコマンドは、実際のロールバックを実行しません。
  実際のロールバックは行いません。Liquibase Pro のライセンスキーが必要です。

Liquibase:ロールバックSQL
  データベースをロールバックするのに必要な SQL を生成します。
  指定された属性 'rollbackCount', 'rollbackTag' および/または
  'rollbackDate'

液化ベース:ステータス
  どのチェンジセットがデータベースに適用される必要があるかを出力します。

Liquibase:タグ
  データベースに Liquibase タグを書き込みます。

liquibase:update
  DatabaseChangeLog をデータベースに適用します。構築プロセスの一部として有用です。
  プロセスの一部として有用です。

LIQIBASE:UPDATESQL
  データベースを DatabaseChangeLogs で指定された現在のバージョンに更新するのに必要な SQL を生成します。
  データベースを DatabaseChangeLogs で指定された現在のバージョンに更新するために必要な SQL を生成します。

liquibase:updateTestingRollback (テストロールバック)
  DatabaseChangeLogs をデータベースに適用し、ロールバックをテストします。これは
  これは、データベースを更新し、それをロールバックし、再び更新することで行われます。


www.DeepL.com/Translator(無料版)で翻訳しました。
```

参考文献

Liquibase | Open Source Version Control for Your Database

Spring Bootでliquibaseを使う方法（MySQL）

Liquibase使い方（基本）メモ

自動的にデータベースを移行！？「LiquiBase」| infoScoop開発者ブログ

DBAを救え! DBリファクタリングツール「LiquiBase」を使ってみよう (1)

“How-to” Guides

Using Liquibase with Spring Boot

Using Liquibase with Spring Boot and Maven Tutorial

Use Liquibase to Safely Evolve Your Database Schema

Using liquibase in idea spring boot maven

https://medium.com/echohub/spring-boot-usage-of-liquibase-c9c5794828b7