Spring Boot Flyway Example of Database Migration

Last modified @ 17 February 2020

Spring Boot # JPA and Hibernate

Flyway is a database migration and version control tool

- It has Java API, command-line client, a plugin for Maven and Gradle
- Supports most of the relational databases such as MySQL, PostgreSQL, SQL Server, and Oracle
- Migration scripts can be written in either SQL or Java

Spring Boot can autorun database migration at the application startup with a variety of mechanisms such as javax.sql.DataSource, JPA and Hibernate, Flyway and Liquibase

This tutorial will give you some highlights and an implementation example of using Flyway in Spring Boot

Add Flyway into your project

You can add Flyway into your project as a dependency on `pom.xml` or `build.gradle` file. The library versions can be found on the Maven Central Repository

If you use Flyway in a Spring Boot project, the dependency version can be omitted as Spring Boot can help you to resolve

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

You may also need the Spring Boot Web and Actuator to query the Flyway migration status and history via a web interface

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Learn more about Spring Boot Actuator

Recommend using only a single migration mechanism

Using multiple migration mechanisms can cause confuse and hard to manage, so it's a good practice to disable others when you are using Flyway

Spring Boot, via JPA and Hibernate, can auto-export schema DDL to a database via your definition on @Entity classes. You can turn it off by setting `validate` or `none` (`none` is the default value for non-embedded databases) to the `spring.jpa.hibernate.ddl-auto` property

Spring Boot can autorun `classpath:schema.sql` and `classpath:data.sql` script files for your DataSource. This feature can be controlled via `spring.datasource.initialization-mode` property. Its value is `embedded` (only apply for embedded databases) by default

Flyway migration scripts

Unlike JPA and Hibernate, database migration in Flyway is not based on the definition of @Entity classes, you have to manually write the migration scripts in either SQL or Java, SQL is the most commonly used

Typically, SQL scripts are in the form `V<VERSION>__<DESCRIPTION>.sql`

- `<VERSION>` is a dot or underscore separated version, such as '1.0' or '1_1'). `<VERSION>` must be unique
- `<DESCRIPTION>` should be informative for you able to remember what each migration does

The following gives you some example SQL scripts

V1.0__create_book.sql

```
CREATE TABLE `book` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `description` varchar(255) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

V1.1__insert_book.sql

```
INSERT INTO `book`(`title`, `description`) VALUES('Hello Koding', 'Coding tutorials series');
```

By default, Spring Boot looks for them in `classpath:db/migration` folder, you can modify that location by setting `spring.flyway.locations`

```
│       └── resources
│           ├── db
│           │   └── migration
│           │       ├── V1.0__create_book.sql
│           │       ├── V1.1__insert_book.sql
│           │       ├── V1.2__insert_book.sql
│           │       └── V1.3__delete_book.sql
│           └── application.properties
└── pom.xml
```

How Flyway works

Flyway applies migration scripts to the underlying database in the order based on the version number specified in the script file naming

At each execution, only pending migrations are applied. Flyway manages this via creating (if not exists) and updating a metadata table. You can find more details about this table in the latter part of this tutorial

The migration scripts can not be changed after applied. Flyway compares the checksum of each script in every execution and throws an exception if there's a mismatch

Config Flyway DataSource

Spring Boot uses either annotations or external properties to connect Flyway to the underlying data source

- @Primary DataSource or @FlywayDataSource annotation
- `spring.datasource.[url, username, password]` ,or `spring.flyway.[url, user, password]` properties

Run Flyway with Spring Boot

Spring Boot auto enable and trigger Flyway at the application startup when you include the Flyway core library into the project. In case you'd like to turn it off, update this setting `spring.flyway.enabled` to `false` (`true` is the default value)

The application startup may be failed if there's an exception (such as the checksum mismatch error of migration scripts mentioned in the previous section) thrown by Flyway during migration

Each migration script is run within a single transaction. You can configure to run all pending migrations in a single transaction with `spring.flyway.group=true` (the default value is `false`)

Query migration status and history

You can query migration status and history in web interface with Spring Boot Actuator by enabling it in this property `management.endpoints.web.exposure.include=info,health,flyway` and access to `{endpoints}/actuator/flyway`

localhost:8080/actuator/flyway


```
- contexts: {
    - application: {
        - flywayBeans: {
            - flyway: {
                - migrations: [
                    - {
                        type: "SQL",
                        checksum: -1590906343,
                        version: "1.0",
                        description: "create book",
                        script: "V1.0__create_book.sql",
                        state: "SUCCESS",
                        installedBy: "root",
                        installedOn: "2019-11-09T07:41:41Z",
                        installedRank: 1,
                        executionTime: 11
                      },
                    - {
                        type: "SQL",
                        checksum: -811051907,
                        version: "1.1",
                        description: "insert book",
                        script: "V1.1__insert_book.sql",
                        state: "SUCCESS",
                        installedBy: "root",
                        installedOn: "2019-11-09T07:41:41Z",
                        installedRank: 2,
                        executionTime: 4
                      },
```


Apart from that, you can also query the table `flyway_schema_history` in your database. It is created by Flyway to manage migration status and history. The table name can be changed via setting `spring.flyway.table` (`flyway_schema_history` is the default name)

```
mysql> describe flyway_schema_history;
+----------------+---------------+------+-----+-------------------+-------+
| Field          | Type          | Null | Key | Default           | Extra |
+----------------+---------------+------+-----+-------------------+-------+
| installed_rank | int(11)       | NO   | PRI | NULL              |       |
| version        | varchar(50)   | YES  |     | NULL              |       |
| description    | varchar(200)  | NO   |     | NULL              |       |
| type           | varchar(20)   | NO   |     | NULL              |       |
| script         | varchar(1000) | NO   |     | NULL              |       |
| checksum       | int(11)       | YES  |     | NULL              |       |
| installed_by   | varchar(100)  | NO   |     | NULL              |       |
| installed_on   | timestamp     | NO   |     | CURRENT_TIMESTAMP |       |
| execution_time | int(11)       | NO   |     | NULL              |       |
| success        | tinyint(1)    | NO   | MUL | NULL              |       |
+----------------+---------------+------+-----+-------------------+-------+
10 rows in set (0.04 sec)
```

```
mysql> select version, description, script, installed_on, execution_time, success from flyway_schema_history;
+---------+-------------+----------------------+---------------------+----------------+---------+
| version | description | script               | installed_on        | execution_time | success |
+---------+-------------+----------------------+---------------------+----------------+---------+
| 1.0     | create book | V1.0__create_book.sql | 2019-11-09 07:41:41 |             11 |       1 |
| 1.1     | insert book | V1.1__insert_book.sql | 2019-11-09 07:41:41 |              4 |       1 |
| 1.2     | insert book | V1.2__insert_book.sql | 2019-11-09 07:41:41 |              6 |       1 |
| 1.3     | delete book | V1.3__delete_book.sql | 2019-11-09 07:41:41 |              4 |       1 |
+---------+-------------+----------------------+---------------------+----------------+---------+
4 rows in set (0.00 sec)
```

Integrate Flyway into an existing database on the production

Several steps have to be executed when you integrate Flyway into a project with an existing database on the production environment. You can learn more about it at here

Conclusion

In this tutorial, we learned using Flyway in Spring Boot to auto migrate database at the application startup. You can find the implementation example on GitHub

Spring Boot # JPA and Hibernate

Share to social

Twitter Facebook

Giau Ngo

Giau Ngo is a software engineer, creator of Hello Koding. Get in touch with him on Twitter, GitHub and LinkedIn

Comments

Hoang Nam • 10 months ago

Hello,
Could you let me know how to generate sql script ? I often let Hibernate generate table schemes for me by changing entities (domain), so how do I generate sql change using Flyway? Thank you!

HelloKoding - Practical Coding Courses, Tutorials and Examples Series

Courses

Java

Spring Boot

Security with Spring

JPA and Hibernate

REST with Spring

Golang

Data Structure

Algorithm

GitHub

Facebook

Twitter

LinkedIn