参考

https://blog.waterstrong.me/flyway-in-practice/

一、基础

1 概要

基本命令

migrate

> - 作用：把数据库schema迁移到最新版本
> - 知识点：
>   - 执行migrate命令时，如果metadata表不存在，会先创建metadata表
> - 实现流程
>   - 检查验证metadata表，表不存在则创建metadata表，验证失败则中断操作

    - 扫描locations指定目录下的migration，将未应用的migration文件 按顺序 应用到数据库
    - migration执行脚本顺序：**Versioned migrations**>**Repeatable migrations**

clean

- 作用：清除掉对应数据库Schema中的所有对象，包括表结构，视图，存储过程，函数以及所有的数据等都会被清除。

info

- 作用：用于打印所有Migrations的详细和状态信息，

validate

- 作用：Validate是指验证已经Apply的Migrations是否有变更，Flyway是默认是开启验证的。
- 工作原理：
    - 对比Metadata表与本地Migrations的Checksum值，如果值相同则验证通过，否则验证失败，从而可以防止对已经Apply到数据库的本地Migrations的无意修改。

baseline

- 作用：
    - 对于已经非空数据库：执行baseline命令，会初始化metadata表，不会应用migration文件，需要再执行migrate命令
    - 对于空数据库：也可以执行baseline进行初始化；
- 工作原理
    - 对于非空的数据库，第一次应用flyway进行数据库管理时，执行baseline可以将已经存在的数据库相关对象应用到flyway中进行管理，并且会初始化metadata表；
    - **非空数据库，首次应用flyway，需要先执行baseline命令，再执行migrate；**
    - 如果在非空的数据库中，直接执行migrate会报异常，如：

      ```
      Found non-empty schema(s) `flyway_practice` but no schema history table. Use baseline() or set baselineOnMigrate to true to initialize the schema history table.
      ```

repair

- 作用：Repair操作能够修复Metadata表，该操作在Metadata表出现错误时是非常有用的。
- 工作原理
    - 移除执行失败的migration记录
    - 重修调整checksum的值

undo

- 声明：**flyway社区版不支持此命令**
- 作用：撤消最近应用的版本迁移。

其他补充

> metadata表：数据库中的`flyway_schema_history`表，用来记录数据库表的变更
>
> Migrations：要执行的sql脚本

二、实验

migration脚本命名规范

- prefix ：脚本前缀包括（U 、V、R）
- version ：U、V 的版本号，R无需定义版本号；由一个或多个数字组成，数字之间可以采用点或下划线，下划线会被替代成点（1.3_1=>1.3.1）
- separator ：分隔符，默认双下划线(__)
- description ：自定义对文件进行描述，描述内容允许包含下划线或空格
- suffix ：默认`.sql`

脚本分类：

V脚本：版本化迁移

prefix	version	separator	description	suffix
V	1.1	__	init-user	.sql

- 作用：用于版本升级，
- 要求：
  - 每个版本只能被应用一次
  - 并且不能修改已经被加载过的migration，（不能修改文件名和文件内容）
  - 版本号不能重复
  - 版本号递增，不能插入（V2migrate失败：V1-> V3->V2）（成功：V1->V2->V3）
- 例子：`V20200507_103000__init-user.sql`、`V1.1__init_user.sql`

R脚本：可重复迁移

- 脚本命名规范

prefix	separator	description	suffix
R	__	user	.sql

- 作用：管理不稳定的数据库对象的更新
- 要求：
  - 其每一次的更新会影响Checksum值，然后都会被重新加载，并不用于版本升级。
  - 执行顺序：Versioned -> Repeatable的Migrations

- 例子：

U脚本：撤销迁移

prefix	version	separator	description	suffix
U	1.1	__	init-user	.sql

- 社区版不支持撤销：

```
Flyway Pro Edition or Flyway Enterprise Edition upgrade required: undo is
not supported by Flyway Community Edition.
```

migration脚本存放位置

> 当项目运行的时候，会去指定位置的目录以及其子目录下扫描migration文件

locations的默认值：

- `classpath:db/migration/` ：src/main/resources/db/migration
- `filesystem` ：my-project/my-other-folder

locations自定义值：

`locations属性值` ：通过`locations属性`指定migration文件的位置

migration的SQL脚本 语法

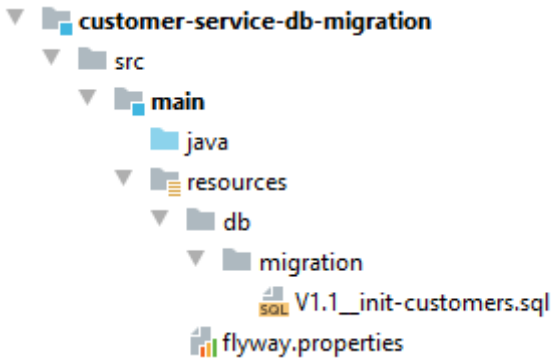
> 官网：https://flywaydb.org/documentation/database/mysql

- 占位符：`${myplaceholder}`
- 多行注释：`/* .... */`
-

三、应用

在gradle项目中应用flyway

子项目结构：



步骤

1、在Gradle中引入Flyway插件

> db-migration.gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'org.flywaydb:flyway-gradle-plugin:6.3.1'
    }
}

// use gradlew -Penv=${env} to pass
def env = hasProperty('env') ? env : null

apply plugin: org.flywaydb.gradle.FlywayPlugin

tasks.withType(org.flywaydb.gradle.task.AbstractFlywayTask).configureEach {
    doFirst {
        flyway {
            configurations = ['runtimeClasspath'] // use runtimeOnly scope in
actual db-migration project
            placeholderReplacement = false
            assert project.file('src/main/resources/db/migration').exists()

            def flywayProperties = env == null ?
'src/main/resources/flyway.properties' :
"conf/${env}/resources/flyway.properties"
            assert project.file(flywayProperties).exists()

            def properties = new Properties()
            project.file(flywayProperties).withInputStream {
                properties.load(it)
            }
            properties.each { key, value ->
                project.ext[key as String] = value
            }
        }
    }
}
```

2、在Gradle中配置Flyway Properties

> flyway/properties

```
flyway.password=root
flyway.url=jdbc:mysql://127.0.0.1:3306/core_ng_practice
flyway.user=root
```

3、在build.gradle给子项目新增MySQL驱动依赖

```
configure(subprojects.findAll {it.name.endsWith('-db-migration') }) {
    apply from: file("${rootDir}/gradle/db-migration.gradle")

    dependencies {
        runtimeOnly "mysql:mysql-connector-java:${mysqlVersion}"
    }
}
```

四、问题

1、手动修改数据库表结构，再次执行migrate或者validate，可以执行成功，并没有提示需要修改checksum的值

2、对于非空数据库，执行baseline，执行baseline之前的数据库对象做变更没有受flyway管理吗？？？

3、checksum的值时怎么计算的？