Apache Flink 漫谈系列 - 搭建Flink 1.11 版本 Table API/SQL开发环境(需求驱动)

一句话需求

整篇贯穿【利用Flink将数据从Kafka迁移到MySQL】进行展开介绍 :-)

基础环境需求

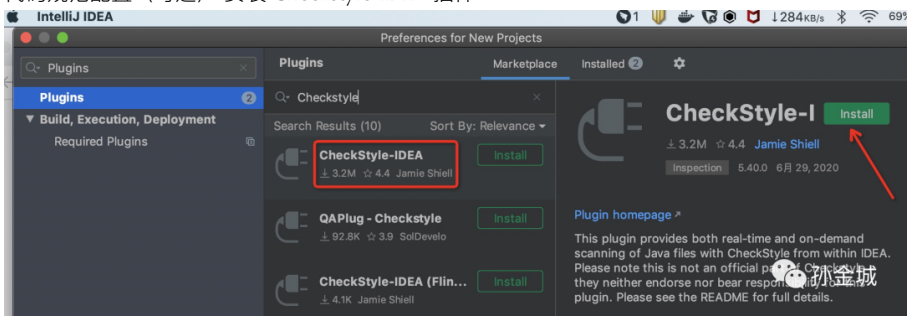
Flink的开发环境需要安装Java和Maven，Java 8 或者 Java 11，Maven3.2+, 我的本地安装是Java 8 和 Maven 3.2.5， 如下：

```
mvn -version
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-15T01:29:23+08:00)
Maven home: /Users/jincheng.sunjc/tools/maven
Java version: 1.8.0_211, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home/jre
Default locale: zh_CN, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.6", arch: "x86_64", family: "mac"
```

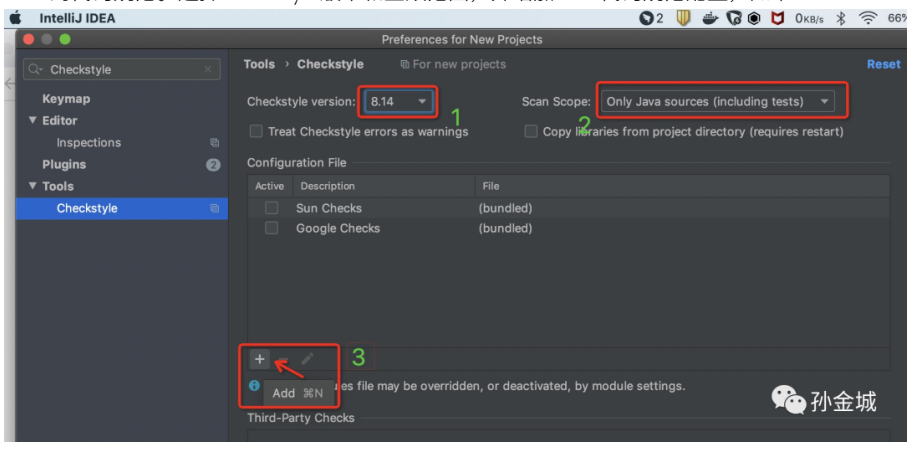
IDE&初始工程

工欲善其事，必先利其器, 我们开发还是选择一个IDE，以IDEA为例。

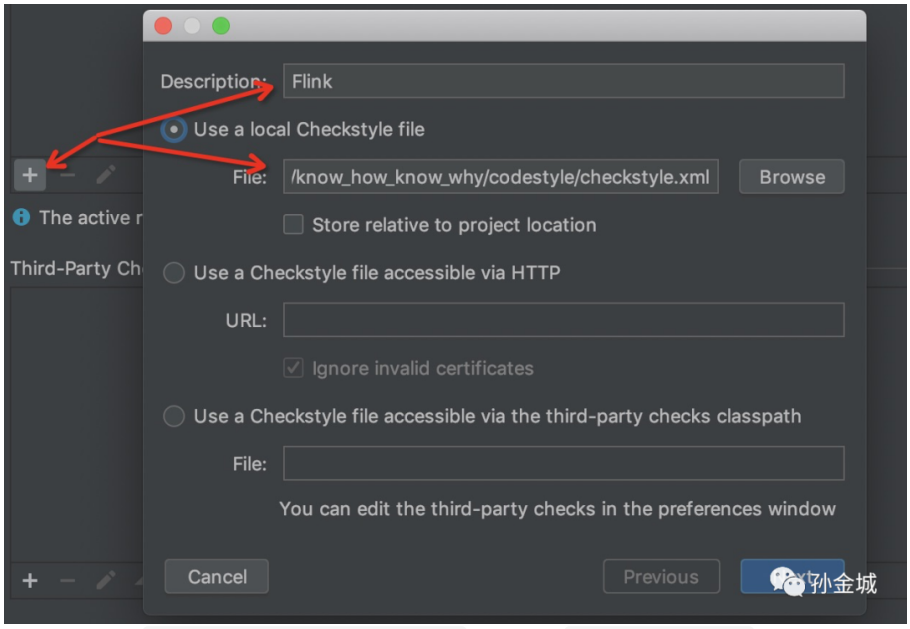
- IDE的下载和安装 这个几乎看这篇文章的没有不会的，贴个下载link都感觉多余：https://www.jetbrains.com/idea/download
- 代码规范配置（可选） 安装 Checkstyle-IDEA 插件



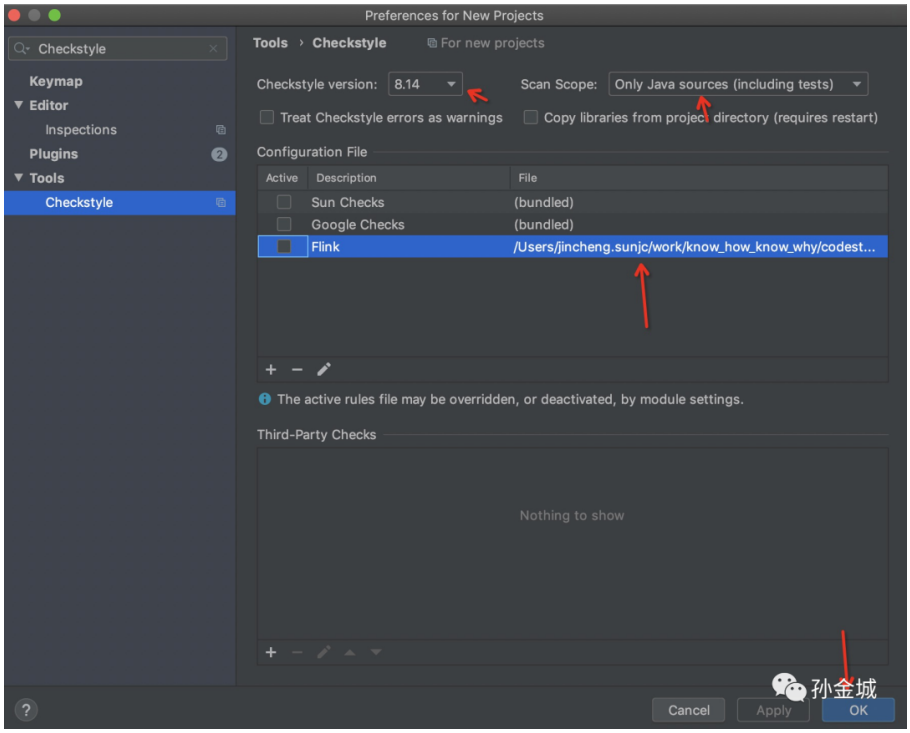
安装完成之后，大家可以配置自己需要的代码规范。如果将来也想参与Flink的贡献，可以配置Flink的代码规范。选择Checkstyle版本和生效范围，并增加Flink代码规范配置，如下：



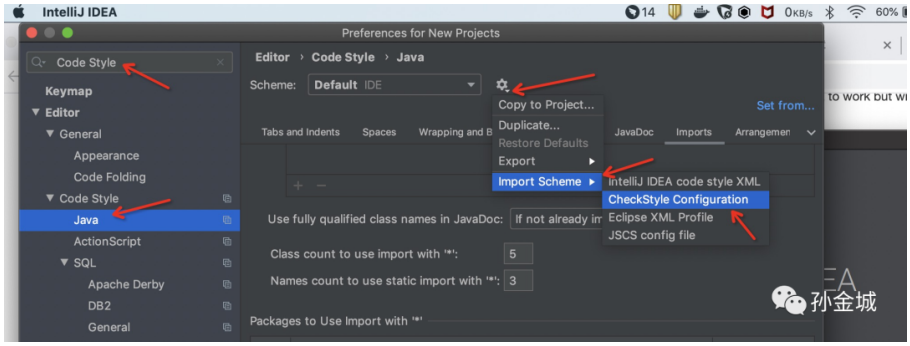
添加的配置文件描述“Flink”，文件从这里获取：
https://github.com/sunjincheng121/know_how_know_why/blob/master/codestyle/checkstyle.xml



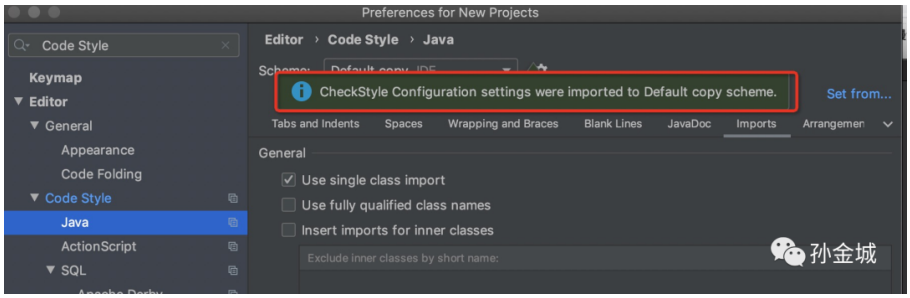
点击 Next，将 `checkstyle.suppressions.file` 值配置为 `suppressions.xml` 点击 Next,点击 Finish，最后点击 “Apply"。



- 配置项目的CodeStyle 我可以为具体某个项目单独配置，也可以配置全局一致的CodeStyle。如下：



选择我们刚才你下载的
https://github.com/sunjincheng121/know_how_know_why/blob/master/codestyle/checkstyle.xml
之后，显示如下：



点击 “Apply”。完成配置。

- 初始工程 我们以Java开发为例，创建一个Java的基础工程，如下：

```
mvn archetype:generate \
-DinteractiveMode=false \
-DgroupId=org.khkw \
-DartifactId=No37-flink-env \
-Dpackage=sql \
-Dversion=0.0.1
```

创建完成之后，我们导入IDEA，并运行自动生成的`App.java`。如果一切顺利，我们就进入和FlinkSQL开发相关的部分。

Flink Hello World

pom依赖

所谓HelloWorld，就是不关心业务逻辑，重在开发环境的调试。首先我们要在项目里面添加Flink SQL开发需要的依赖，作为第一次学习，不用关心每个细节，当作黑盒子。`pom.xml`文件核心依赖（部分）如下详细参阅 https://github.com/sunjincheng121/know_how_know_why/blob/master/khkw/No37-flink-env/pom.xml：

```
<dependencies>
     <!-- 利用Java开发 -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-api-java-bridge_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- 使用Blink Planner -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-planner-blink_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- 支持一些自定义的消息格式，比如kafka里面消息格式是json的，或者需要自定义函数支持 -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-common</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- JDBC Connector的支持，本案例会是使用MySQL -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-jdbc_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- Kafka Connector的支持 -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-sql-connector-kafka-0.11_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- Kafka里面的消息采用Json格式 -->
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-json</artifactId>
    <version>${flink.version}</version>
  </dependency>

  <!-- MySQL的驱动 -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
  </dependency>

  <!--提交作业所必须的依赖，比如：LocalExecutorFactory -->
```

```
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-clients_${scala.binary.version}</artifactId>
      <version>${flink.version}</version>
    </dependency>
  </dependencies>
```

示例代码

Flink为了方便用户测试，特意开发了内置的随机数据源(datagen source)和控制台打印数据汇(print sink)，我们Helloworld示例就是声明数据源和数据输出DDL，然后直接将表结构形同的数据源（source）表写入到数据汇（sink）表。代码如下：

- 数据源（source table）定义 为了方便测试，flink提供了自动生成数据的随机Source, DDL声明如下：

```
String sourceDDL = "CREATE TABLE random_source (\n" +
                " f_sequence INT,\n" +
                " f_random INT,\n" +
                " f_random_str STRING\n" +
                ") WITH (\n" +
                " 'connector' = 'datagen',\n" +
                " 'rows-per-second'='5',\n" +
                " 'fields.f_sequence.kind'='sequence',\n" +
                " 'fields.f_sequence.start'='1',\n" +
                " 'fields.f_sequence.end'='1000',\n" +
                " 'fields.f_random.min'='1',\n" +
                " 'fields.f_random.max'='1000',\n" +
                " 'fields.f_random_str.length'='10'\n" +
                ")";
```

双面代码就是声明了一个有三个字段，并且制定了一定数据生成规则的随机数据源表。非常方便吧？：）

- 数据汇（sink table）定义 为了方便测试，flink提供了控制台打印的sink, DDL声明如下：

```
 String sinkDDL = "CREATE TABLE print_sink (\n" +
                " f_sequence INT,\n" +
                " f_random INT,\n" +
                " f_random_str STRING \n" +
                ") WITH (\n" +
                " 'connector' = 'print'\n" +
                ")";
```

上面就声明了和上面随机数据源一样的表schema的输出表。

- 数据同步逻辑 我们的数据同步逻辑非常简单，就是无任何数据转化的同步数据，如下：

```
//数据提取
Table sourceTab = tEnv.from("random_source"); // 读
sourceTab.insertInto("print_sink");//写
```

- 数据表注册 当然仅仅是DDL声明还不能让表生效，我们还需要将表进行注册，如下：

```
 //注册source和sink
 tEnv.executeSql(sourceDDL); //其中 tEnv需要提前进行创建，见下面初始化代码部分
 tEnv.executeSql(sinkDDL);
```

- 环境初始化和提交作业 一个完整的Flink Table API/SQL作业，需要指定批流运行模式，需要指定使用flink planner或者blink planner，所以会需要一些初始化工作。当然写完作业之后还需要作业的部署提交。最简单的逻辑如下：

```
// 创建执行环境
EnvironmentSettings settings = EnvironmentSettings .newInstance()
                .useBlinkPlanner() //  使用blink planner
                .inStreamingMode() // 流模式运行
                .build();
TableEnvironment tEnv = TableEnvironment.create(settings);

...
业务逻辑
...

//提交作业
tEnv.execute("Flink Hello World"); //这里我们暂时先使用 标注了 deprecated 的API，因为新的异步提交测试有待改进...
```

完整详细代码参阅
https://github.com/sunjincheng121/know_how_know_why/blob/master/khkw/No37-flink-env/src/main/java/sql/App.java。

当然大家也可下载完整的工程代码直接进行运行观察效果
https://github.com/sunjincheng121/know_how_know_why/tree/master/khkw/No37-flink-env

到这里简单的HelloWorld就跑通了，也就是Flink环境问题搞定了，但是我们还没有考虑业务需求整

体链路的环境问题，也就是需求是【利用Flink将数据从Kafka迁移到MySQL】，接下来我们考虑Kafka和Mysql的环境怎么搞定。

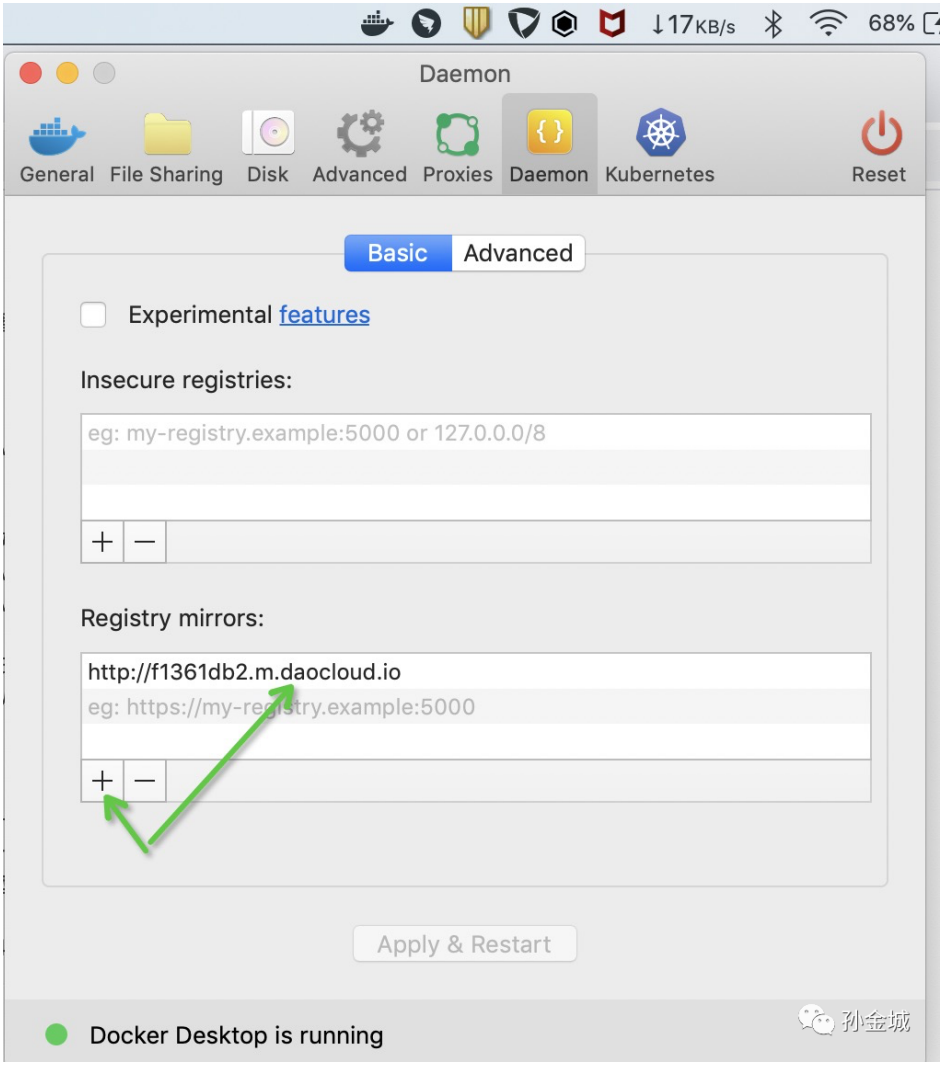
外部数据源环境

目前看来利用Docker来搭建开发环境是最方便的方式，下载和安装参见https://www.docker.com/get-started 。安装完之后，我们可以根据自己机器配置对Docke的CPU，内存进行简单的配置，我的环境配置如下：

然后，我们可以在命令行查看一下Docker的版本，如下：

```
$ docker --version
Docker version 19.03.4, build 9013bf5
```

同时，国内有些时候拉起镜像会很慢，那么大家也可以选择配置加速器来解决，比如：`http://f1361db2.m.daocloud.io`（不确定是否是最好的选择），如下：



设置之后，点击 Apply & Restart 按钮使设置生效。查看信息如下：

```
$ docker info
```

安装Kafka

目前有很多Kafka的镜像可以供我们学习是使用，对于学习Flink而言，大多数都满足我们的需求，所以你可以任意选择，我们可以现在搜索一些可用的镜像，如下：

```
$ docker search kafka
NAME                              DESCRIPTION                                     STARS               OFFICIAL
wurstmeister/kafka                Multi-Broker Apache Kafka Image                 1200
spotify/kafka                     A simple docker image with both Kafka and Zo…   398
...
...
```

我们可以选择第一个`wurstmeister/kafka`就可以，进行安装如下：

```
jincheng:~ jincheng.sunjc$ docker pull wurstmeister/kafka:2.12-2.5.0
2.12-2.5.0: Pulling from wurstmeister/kafka
Image docker.io/wurstmeister/kafka:2.12-2.5.0 uses outdated schema1 manifest format. Please upgrade to a schema2 image for
e7c96db7181b: Retrying in 3 seconds
f910a506b6cb: Retrying in 3 seconds
...
...
18616ed64100: Pull complete
Digest: sha256:a9980b591efe62a68de0acf5f5ce2f6fa7112ab07ec1099c976cdadc740c7ea4
Status: Downloaded newer image for wurstmeister/kafka:2.12-2.5.0
docker.io/wurstmeister/kafka:2.12-2.5.0
jincheng:~ jincheng.sunjc$
```

我们还需要安装Zookeeper，一样，我们可以先搜索在选择安装，如下：

```
$ docker search zookeeper
NAME                              DESCRIPTION                                     STARS               OFFICIAL
```

```
zookeeper                         Apache ZooKeeper is an open-source server wh…   900                 [OK]
jplock/zookeeper                  Builds a docker image for Zookeeper version …   164
wurstmeister/zookeeper                                                            126
mesoscloud/zookeeper              ZooKeeper                                       73
```

我们选择 `zookeeper` ,进行安装，如下：

```
jincheng:~ jincheng.sunjc$ docker pull zookeeper:3.6.1
3.6.1: Pulling from library/zookeeper
Image docker.io/library/zookeeper:3.6.1 uses outdated schema1 manifest format. Please upgrade to a schema2 image for better
6ec8c9369e08: Retrying in 13 seconds
3aa4e9b77806: Retrying in 13 seconds
6d8b5d3bc409: Retrying in 13 seconds
...
...
Digest: sha256:987edbc16352baf0e68a383906d2fc8cd9af89f4593c6a37abb73df0dd35f413
Status: Downloaded newer image for zookeeper:3.6.1
docker.io/library/zookeeper:3.6.1
jincheng:~ jincheng.sunjc$
```

安装完成之后，我们可以查看一下已经安装的镜像，如下命令：

```
jincheng:~ jincheng.sunjc$ docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
zookeeper            3.6.1               7341c5373a13        3 days ago          252MB
wurstmeister/kafka   2.12-2.5.0          f1905dce9659        2 months ago        431MB
jincheng:~ jincheng.sunjc$
```

如上简单命令我们就在Docker里面安装了Kafka环境，接下来我们进行容器的启动,先启动zookeeper,并用 `docker ps` 查看启动状态，如下：

```
jincheng:~ jincheng.sunjc$ docker run -d --name zookeeper  -p 2181:2181 -t zookeeper:3.6.1
db408eff568a8c519c6f6c6683ddc41e8370ea66872f71916da139f80f8c0145
jincheng:~ jincheng.sunjc$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
db408eff568a        zookeeper:3.6.1     "/docker-entrypoint.…"   11 seconds ago      Up 9 seconds        2888/tcp, 3888/tcp
jincheng:~ jincheng.sunjc$
```

如上信息说明我们已经启动了zookeeper，并且映射了2181端口号。接下来，我们启动Kafka，如下：

```
jincheng:~ jincheng.sunjc$ docker run -d --name kafka --publish 9092:9092 --link zookeeper --env KAFKA_ZOOKEEPER_CONNECT=zo
125900ffcd28c7390282fa2bf09f5287be78de1fd3f4aac30d3e0bffdfd80ae7
```

查看Docker启动的所有容器如下；

```
jincheng:~ jincheng.sunjc$ docker ps
CONTAINER ID        IMAGE                             COMMAND                  CREATED             STATUS              PORTS
a6f9518b7c4a        wurstmeister/kafka:2.12-2.5.0     "start-kafka.sh"         14 seconds ago      Up 13 seconds       0.0.0.
e4b476f3165e        zookeeper:3.6.1                   "/docker-entrypoint.…"   49 seconds ago      Up 48 seconds       2888/t
jincheng:~ jincheng.sunjc$
```

如上信息证明，Kafka已经成功启动，接下来我们简单测试一下,我们启动Kafka所在容器的shell，并且收发消息：

- 进入Kafka命令行

```
jincheng:~ jincheng.sunjc$ docker exec -it kafka /bin/bash
bash-4.3#
```

- 尝试创建一个Topic

```
bash-4.4# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic test
Created topic test.
```

- 向 `test` Topic 发数据：

```
bash-4.4# /opt/kafka/bin/kafka-console-producer.sh --topic=test --broker-list localhost:9092
>hello
>welcome you
```

这时候, 我们需要启动另一个Kafka的Shell来消费消息，如下：

```
jincheng:~ jincheng.sunjc$ docker exec -it kafka /bin/bash
bash-4.4# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 -from-beginning --topic test
hello
welcome you
```

好的，至此，我们Kafka的环境部署测试完成，下面进行MySQL部分。

安装MySQL

安装MySQL的过程和Kafka类似，我们可以先搜索，再安装，如下：

```
jincheng:~ jincheng.sunjc$ docker search mysql
NAME                              DESCRIPTION                                     STARS               OFFICIAL            A
mysql                             MySQL is a widely used, open-source relation…   9790                [OK]
mariadb                           MariaDB is a community-developed fork of MyS…   3572                [OK]
mysql/mysql-server                Optimized MySQL Server Docker images. Create…   717                                     [
...
...
```

我们选择第一个，进行安装MySQL5.7，如下：

```
jincheng:~ jincheng.sunjc$ docker pull mysql:5.7
5.7: Pulling from library/mysql
Image docker.io/library/mysql:5.7 uses outdated schema1 manifest format. Please upgrade to a schema2 image for better futur
6ec8c9369e08: Already exists
177e5de89054: Retrying in 10 seconds
ab6ccb86eb40: Retrying in 10 seconds
e1ee78841235: Retrying in 10 se
...
...
5f13eadfe747: Pull complete
Digest: sha256:97869b42772dac5b767f4e4692434fbd5e6b86bcb8695d4feafb52b59fe9ae24
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

再次查看一下我们安装的所有镜像信息，如下：

```
jincheng:~ jincheng.sunjc$ docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
zookeeper            3.6.1               7341c5373a13        3 days ago          252MB
mysql                5.7                 8679ced16d20        10 days ago         448MB
wurstmeister/kafka   2.12-2.5.0          f1905dce9659        2 months ago        431MB
```

Mysql成功安装我们进行启动，如下：

```
jincheng:~ jincheng.sunjc$ mkdir -p $PWD/conf $PWD/logs $PWD/data
jincheng:~ jincheng.sunjc$ docker run -p 3306:3306 --name flink_mysql -v $PWD/conf:/etc/mysql/conf.d -v $PWD/logs:/logs -v
cbef5d43fe22b2e8138756619cfda358a2a5de4590cbfc797d2eb153ca301966
```

查看启动情况，如下：

```
jincheng:~ jincheng.sunjc$ docker ps -a |grep flink_mysql
cbef5d43fe22        mysql:5.7                      "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0
```

简单的解释一下命令参数含义如下：

- `-p 3306:3306`：将容器的 3306 端口映射到主机的 3306 端口。
- `-v $PWD/conf:/etc/mysql/conf.d -v $PWD/logs:/logs -v $PWD/data:/var/lib/mysql`：本机目录和容器目录进行挂载，比如：PWD/logs:/logs：将主机当前目录下的 logs 目录挂载到容器的 /logs。
- `-e MYSQL_ROOT_PASSWORD=123456`：初始化 root 用户的密码。
- `-d`：后台运行容器，并返回容器ID

Mysql已经启动，并映射了3306端口号，我们尝试连接一下：

```
jincheng:~ jincheng.sunjc$ docker exec -it flink_mysql bash
root@cbef5d43fe22:/# mysql -h localhost -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.31 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

进入了Mysql的Shell命令行，我们可以尝试创建数据库和表，如下：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.03 sec)

mysql> create database flinkdb;
Query OK, 1 row affected (0.00 sec)
mysql> use flinkdb;
```

```
Database changed
mysql> create table welcome ( user_id VARCHAR(20), age INT(11), name VARCHAR(20))
    -> ;
Query OK, 0 rows affected (0.03 sec)

mysql> insert into welcome values('No1', 22, 'Eden');
Query OK, 1 row affected (0.01 sec)

mysql> select * from welcome;
+---------+------+------+
| user_id | age  | name |
+---------+------+------+
| No1     |   22 | Eden |
+---------+------+------+
1 row in set (0.00 sec)

mysql>
```

好了，如果上面这轮操作都成功了，那么证明MySQL环境已经可用了，接下来我尝试利用Flink进行操作。

Kafka迁移到MySQL的完整示例

Kafka的Topic和MySQL结果表

- 创建一个Kafka的Topic，比如叫：`cdn-log`：

```
bash-4.4# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic cdn-log
Created topic cdn-log.
```

- 创建一个Mysql的Table，比如也叫：`cdn_log`：

```
mysql> create table cdn_log (msg VARCHAR(300));
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+-------------------+
| Tables_in_flinkdb |
+-------------------+
| cdn_log           |
| welcome           |
+-------------------+
2 rows in set (0.01 sec)
```

DDL声明

- 配置Kafka的DDL连接属性

```
CREATE TABLE kafka_source (
 msg STRING
) WITH (
 'connector' = 'kafka-0.11',
 'topic' = 'cdn-log',
 'properties.bootstrap.servers' = 'localhost:9092',
 'format' = 'json',
 'scan.startup.mode' = 'latest-offset'
)
```

其中，

(1) `format`：可以有多种选择，如：`JSON`，`CSV`，`AVRO`，`Canal CDC`，`Debezium CDC`等，详细见：https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/connectors/formats/

(2) `can.startup.mode`：也可以有多种选择，如：`earliest-offset`，`latest-offset`，`group-offsets`，`timestamp` and `specific-offsets`详见：https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/connectors/kafka.html#start-reading-position

- 配置MySQL的DDL连接属性

```
CREATE TABLE mysql_sink (
 msg STRING
) WITH (
  'connector' = 'jdbc',
   'url' = 'jdbc:mysql://localhost:3306/flinkdb?characterEncoding=utf-8&useSSL=false',
   'table-name' = 'cdn_log',
   'username' = 'root',
   'password' = '123456',
   'sink.buffer-flush.max-rows' = '1'
)
```

我们配置Mysql的连接属性比较检查，详细的其他参数参见https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/connectors/jdbc.html

完整的代码

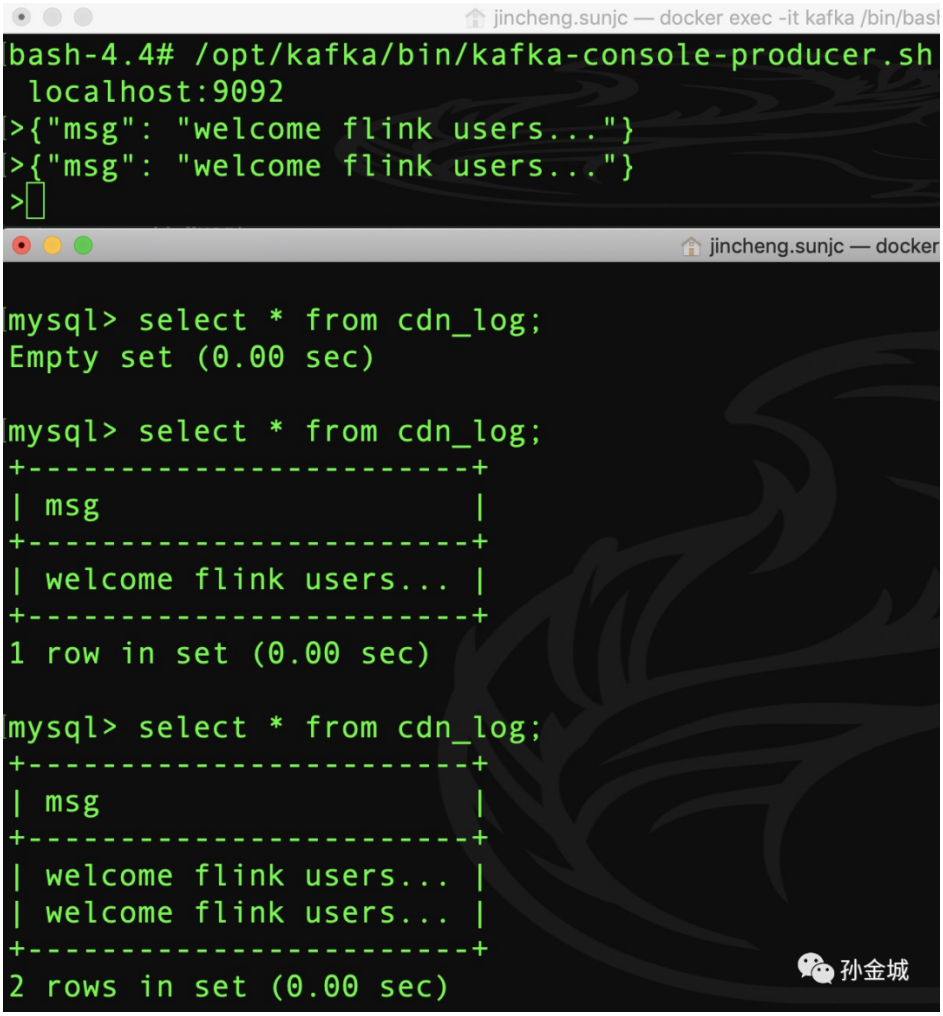
本示例的完整SQL代码，如下：

```
// Kafka {"msg": "welcome flink users..."}
 String sourceDDL = "CREATE TABLE kafka_source (\n" +
         " msg STRING\n" +
         ") WITH (\n" +
         " 'connector' = 'kafka-0.11',\n" +
         " 'topic' = 'cdn-log',\n" +
         " 'properties.bootstrap.servers' = 'localhost:9092',\n" +
         " 'format' = 'json',\n" +
         " 'scan.startup.mode' = 'latest-offset'\n" +
         ")";

 // Mysql
 String sinkDDL = "CREATE TABLE mysql_sink (\n" +
         " msg STRING \n" +
         ") WITH (\n" +
         "  'connector' = 'jdbc',\n" +
         "   'url' = 'jdbc:mysql://localhost:3306/flinkdb?characterEncoding=utf-8&useSSL=false',\n" +
         "   'table-name' = 'cdn_log',\n" +
         "   'username' = 'root',\n" +
         "   'password' = '123456',\n" +
         "   'sink.buffer-flush.max-rows' = '1'\n" +
         ")";

 // 创建执行环境
 EnvironmentSettings settings = EnvironmentSettings
         .newInstance()
         .useBlinkPlanner()
         .inStreamingMode()
         .build();
 TableEnvironment tEnv = TableEnvironment.create(settings);

 //注册source和sink
 tEnv.executeSql(sourceDDL);
 tEnv.executeSql(sinkDDL);

 //数据提取
 Table sourceTab = tEnv.from("kafka_source");
 //这里我们暂时先使用 标注了 deprecated 的API，因为新的异步提交测试有待改进...
 sourceTab.insertInto("mysql_sink");
 //执行作业
 tEnv.execute("Flink Hello World");
```

也可以查阅 https://github.com/sunjincheng121/know_how_know_why/blob/master/khkw/No37-flink-env/src/main/java/sql/Kafka2Mysql.java

完整测试

我们在IDE启动SQL作业，然后向Kafka的 `cdn-log` 主题发送消息 `{"msg": "welcome flink users ..."}` ,然后再Mysql的shell环境对表 `cdn_log` 表进行查询，具体操作过程如图：



小结

本篇对Flink 1.11版本SQL学习的开发环境进行step by step的介绍，我们以一句话需求 利用Flink将数据从Kafka迁移到MySQL 驱动整篇内容的介绍，并且为大家提供了可以进行独立测试的示例代码:https://github.com/sunjincheng121/know_how_know_why/blob/master/khkw/No37-flink-env,期望本篇对大家走进Flink SQL 有所帮助！最后，感谢大家花时间查阅～

与本篇对应的视频演示

本篇是为《Apache Flink 知其然，知其所以然》视频课程第37讲的文字辅助内容，大家也可以关注我订阅号，收看No.37 Flink SQL 开发环境视频课程。