

# FlinkSQL Client的概念

- FlinkSQL就是使用ansi-SQL标准实现流处理应用
- 不需要写一行java代码，纯SQL实现
- 在FlinkSQL Client客户端中实现SQL-client.sh

## FlinkClient工具的使用

- FlinkClient就是不需要嵌入Java和scala、Python代码，直接通过SQL展现结果工具，不需要每次提交作业都要打包并提交集群，这或多或少限制了Flink的使用
- 目标 - 易写、调试、提交表程序到Flink集群，易交互
- 开启SQL-client.sh

```
twalthr@TMMACBOOK ~/flink/flink/build-target FLINK-9181 ./bin/sql-client.sh embedded --library ./my-sql-libraries
```

## • 使用FLinkClient步骤

### ○ 1.配置FlinkClient及catalog

- catalog默认内存，基于session会话；
- 基于jdbc metastore例如postgresql默认；
- hivecatalog 使用hive metastore

### ○ 2.开启Flink集群 standalone模式

```
1 # 开启hadoop 集群，checkpoint 存储目录在 hdfs
2 start-all.sh
3 # 开启zookeeper集群，HA高可用，选举
4 ./bin/zkServer.sh -server node1:2181
5 # 开启Flink cluster集群，standalone-HA高可用模式
```

```
6 ./bin/start-cluster.sh
```

- **3./export/server/flink/bin/sql-client.sh embedded 进入 Flink Client**
  - HELP 命令用于查看脚本的指令

```
1 SELECT 'HELLO WORLD';
```

## • 4.设置参数

```
1 # 设置显示的表格样式
2 SET sql-client.execution.result-mode=table;
3 ## DML 插入、更新、删除
4 SET sql-client.execution.result-mode=changelog;
5 ## 设置显示样式是二维表
6 SET sql-client.execution.result-mode=tableau;
7 # CTRL + C 结束当前的会话
8 ## 设置显示的记录条数的大小,默认是 1000 行
9 SET sql-client.execution.max-table-result.rows=1000
```

## • 5.需要外部数据源而不是手动输入

- 常见外部数据源
  - kafka
  - hive
  - MySQL
  - FileSystem
- 如何加载这些数据源
  - 1.catalog元数据
    - catalog就是元数据、表名、字段、分区、类型、数据库等，元数据是临时的，每次启动session都需要使用use catalog
    - catalog类型
      - GenerichMemoryCatalog 自带的catalog
      - JdbcCatalog 连接外部MySQL Oracle数据库的catalog
      - HiveCatalog连接外部hive数据仓库的catalog
      - 用户自定义的catalog
  - 2.连接catalog，初始化外部链接元数据

- **6.创建catalog，数据库和表，数据来自于FileSystem】 kafka、elasticsearch**

```
1 CREATE DATABASE IF NOT EXISTS flink;
```

```
2 use flink;
3
4 CREATE TABLE IF NOT EXISTS t_students(
5     sno STRING,
6     sname STRING,
7     gender STRING,
8     age INT,
9     course STRING
10 ) WITH (
11     'connector' = 'filesystem',
12     'path' = 'hdfs://node1:8020/user/hive_data/students.txt',
13     'format' = 'csv',
14     'csv.field-delimiter' = ','
15 );
```

- 7.编写SQL执行结果

- FlinkClient SQL-client.sh 客户端配置

[illegible]

`-l,--library <JAR directory>`

`-pyarch,--pyArchives <arg>`

`-pyexec,--pyExecutable <arg>`

functions, `table` sources, `or` sinks.  
Can be used multiple times.  
A JAR file directory with which every new `session` is initialized. The files might contain `user`-defined classes needed `for` the execution of statements such `as` functions, `table` sources, `or` sinks. Can be used multiple times.  
Add python archive files for job. The archive files will be extracted `to` the working directory `of` python UDF worker. Currently only zip-format `is` supported. `For` each archive `file`, a target directory be specified. `If` the target directory name `is` specified, the archive `file` will be extracted `to` a directory `with` the specified name. Otherwise, the archive `file` will be extracted `to` a directory `with` the same name `of` the archive `file`. The files uploaded via this `option` are accessible via relative path. `'#'` could be used `as` the separator `of` the archive `file` path `and` the target directory name. Comma `(',')` could be used `as` the separator `to` specify multiple archive files. This `option` can be used `to` upload the virtual environment, the `data` files used `in` Python UDF (e.g.: `--pyArchives file:///tmp/py37.zip,file:///tmp/data.zip#data --pyExecutable py37.zip/py37/bin/python`). The `data` files could be accessed `in` Python UDF, e.g.: `f = open('data/data.txt', 'r')`.  
Specify the path of the python interpreter used `to execute` the

63		python UDF worker (e.g.:
64		--pyExecutable
65		/usr/local/bin/python3). The python
66		UDF worker depends on Python 3.6+,
67		Apache Beam (version == 2.27.0), Pip
68		(version >= 7.1.0) and SetupTools
69		(version >= 37.0.0). Please ensure
70		that the specified environment meets
71		the above requirements.
72	-pyfs,--pyFiles <pythonFiles>	Attach custom files for job.
73		The standard resource file suffixes
74		such as .py/.egg/.zip/.whl or
75		directory are all supported. These
76		files will be added to the PYTHONPATH
77		of both the local client and the
78		remote python UDF worker. Files
79		suffixed with .zip will be extracted
80		and added to PYTHONPATH. Comma (',')
81		could be used as the separator to
82		specify multiple files (e.g.:
83		--pyFiles
84		file:///tmp/myresource.zip,hdfs:/// \$n
85		amenode_address/myresource2.zip).
86	-pyreq,--pyRequirements <arg>	Specify a requirements.txt file which
87		defines the third-party dependencies.
88		These dependencies will be installed
89		and added to the PYTHONPATH of the
90		python UDF worker. A directory which
91		contains the installation packages of
92		these dependencies could be specified
93		optionally. Use '#' as the separator
94		if the optional parameter exists
95		(e.g.: --pyRequirements
96		file:///tmp/requirements.txt#file:///
97		tmp/cached_dir).
98	-s,--session <session identifier>	The identifier for a session.
99		'default' is the default identifier.

- Flink ON Hive创建Hive的catalog，连接hive表去实现

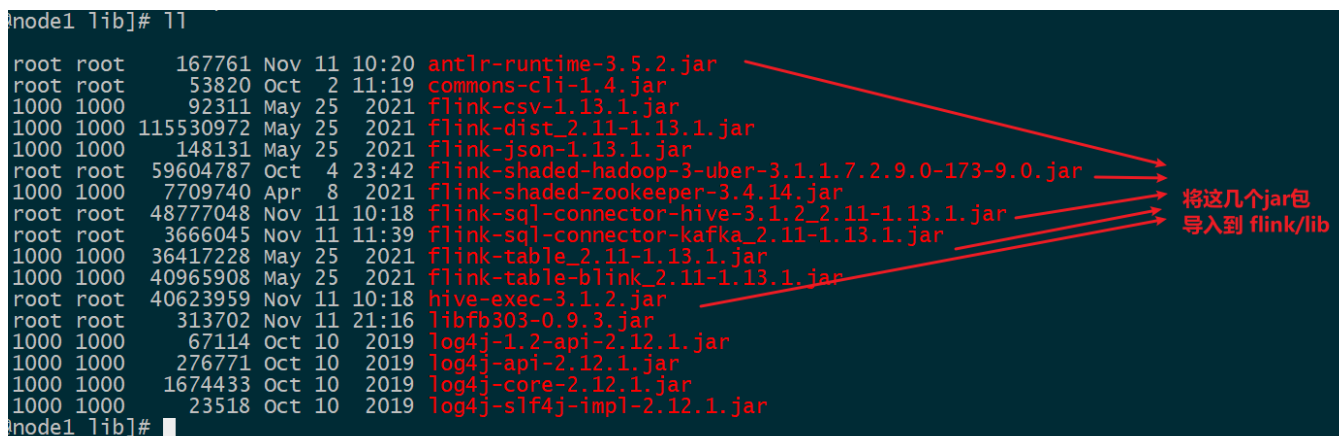
# sql

- Flink连接Hive的配置

```
1 # 环境变量中设置 Hadoop lib 配置依赖,三台
2 # vim /etc/profile
3 export HADOOP_CLASSPATH=`hadoop classpath`
4 # 重置环境变量 source /etc/profile
```

- Flink连接hive的lib包导入, 导入三台节点

```
1 # 将 lib 目录中 上传到 node1 /export/server/flink/lib
2 # 在 今天的 资料包中有
3 # 将 flink-sql-connector-hive-3.1.2_2.11-1.13.1.jar 分发到另外两台节点中
4 # 在lib目录下 flink-sql-connector-hive-3.1.2_2.11-1.13.1.jar
5
6 # 在 /export/server/hive/lib 包中有
7 # 在hive lib目录中 hive-exec-3.1.2.jar
8 # 在hive lib目录中 libfb303-0.9.3.jar
9 # 在hive lib目录中 antlr-runtime-3.5.2.jar
10 # 将以上五个包拷贝到 Flink lib 目录下并分发到各个节点
```



```
node1 lib]# ll
root root      167761 Nov 11 10:20 antlr-runtime-3.5.2.jar
root root       53820 Oct  2 11:19 commons-cli-1.4.jar
1000 1000      92311 May 25 2021 flink-csv-1.13.1.jar
1000 1000 115530972 May 25 2021 flink-dist_2.11-1.13.1.jar
1000 1000      148131 May 25 2021 flink-json-1.13.1.jar
root root     59604787 Oct  4 23:42 flink-shaded-hadoop-3-uber-3.1.1.7.2.9.0-173-9.0.jar
1000 1000      7709740 Apr  8 2021 flink-shaded-zookeeper-3.4.14.jar
root root     48777048 Nov 11 10:18 flink-sql-connector-hive-3.1.2_2.11-1.13.1.jar
root root     3666045 Nov 11 11:39 flink-sql-connector-kafka_2.11-1.13.1.jar
1000 1000     36417228 May 25 2021 flink-table_2.11-1.13.1.jar
1000 1000     40965908 May 25 2021 flink-table-blink_2.11-1.13.1.jar
root root     40623959 Nov 11 10:18 hive-exec-3.1.2.jar
root root      313702 Nov 11 21:16 libfb303-0.9.3.jar
1000 1000      67114 Oct 10 2019 log4j-1.2-api-2.12.1.jar
1000 1000      276771 Oct 10 2019 log4j-api-2.12.1.jar
1000 1000     1674433 Oct 10 2019 log4j-core-2.12.1.jar
1000 1000      23518 Oct 10 2019 log4j-slf4j-impl-2.12.1.jar
node1 lib]#
```

将这几个jar包  
导入到 flink/lib

- 重启Flink Standalone

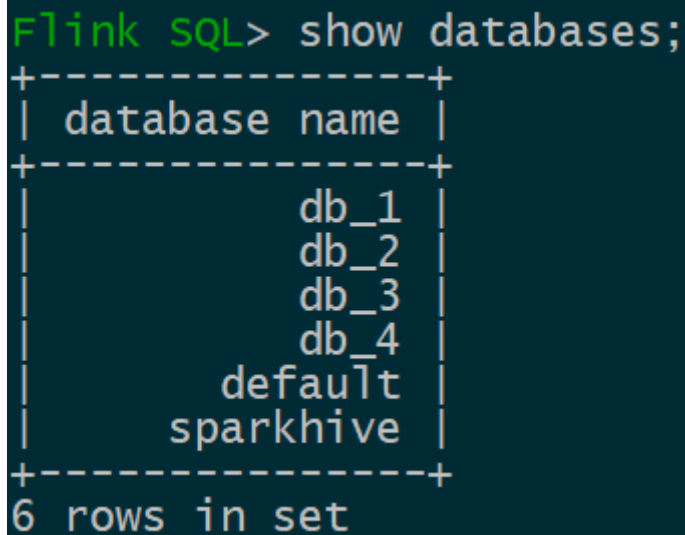
```
1 [root@node1 flink]# bin/stop-cluster.sh
2 [root@node1 flink]# bin/start-cluster.sh
```

- 开启Hive服务

```
1 [root@node1 hive]# nohup bin/hive --service metastore &
2 [root@node1 hive]# nohup bin/hive --service hiveserver2 &
```

- 初始化操作

```
1 # 创建 catalog 元数据, 告诉 Flink 要操作的外部系统的信息
2 CREATE CATALOG myhive WITH (
3     'type' = 'hive',
4     'hive-conf-dir' = '/export/server/hive/conf/'
5 );
6 -- set the HiveCatalog as the current catalog of the session
7 USE CATALOG myhive;
```



The screenshot shows a terminal window with the prompt 'Flink SQL>'. The command 'show databases;' has been executed, resulting in a table listing the available databases. The table has one column, 'database name', and six rows of data: 'db\_1', 'db\_2', 'db\_3', 'db\_4', 'default', and 'sparkhive'. Below the table, it says '6 rows in set'.

database name
db_1
db_2
db_3
db_4
default
sparkhive

- 使用初始化init.sh 初始化元数据操作

```
1 # 设置
2 SET sql-client.verbose = true;
3 CREATE CATALOG myhive WITH (
4     'type' = 'hive',
5     'hive-conf-dir' = '/export/server/hive/conf/'
6 );
7
8 USE CATALOG myhive;
```

```
1 # 执行命令
2 (base) [root@node1 flink]# bin/sql-client.sh embedded -i bin/init.txt -f /root/cd.sql
```

## • Flink操作Hive

- 创建数据表的格式

```
1 CREATE TABLE MyUserTable (
2     column_name1 INT,
```

```

3  column_name2 STRING,
4  ...
5  part_name1 INT,
6  part_name2 STRING
7  ) PARTITIONED BY (part_name1, part_name2) WITH (
8  'connector' = 'filesystem',          -- required: specify the connector
9  'path' = 'file:///path/to/whatever', -- required: path to a directory
10 'format' = '...',                   -- required: file system connector requires to
    specify a format,
11                                     -- Please refer to Table Formats
12                                     -- section for more details
13 'partition.default-name' = '...',    -- optional: default partition name in case the
    dynamic partition
14                                     -- column value is null/empty string
15
16 -- optional: the option to enable shuffle data by dynamic partition fields in sink
    phase, this can greatly
17 -- reduce the number of file for filesystem sink but may lead data skew, the default
    value is false.
18 'sink.shuffle-by-partition.enable' = '...',
19 ...xxxxxxxxx CREATE TABLE MyTable( MyField1 INT, MyField2 STRING) WITH (
    'connector' = 'filesystem', 'path' = '/path/to/something', 'format' = 'csCREATE
    TABLE MyUserTable ( column_name1 INT, column_name2 STRING, ... part_name1 INT,
    part_name2 STRING) PARTITIONED BY (part_name1, part_name2) WITH ( 'connector' =
    'filesystem',          -- required: specify the connector 'path' =
    'file:///path/to/whatever', -- required: path to a directory 'format' = '...',
        -- required: file system connector requires to specify a format,
        -- Please refer to Table Formats
        -- section for more details 'partition.default-name' = '...',    --
    optional: default partition name in case the dynamic partition
        -- column value is null/empty string -- optional: the option to enable
    shuffle data by dynamic partition fields in sink phase, this can greatly -- reduce the
    number of file for filesystem sink but may lead data skew, the default value is false.
    'sink.shuffle-by-partition.enable' = '...', ...v');

```

- 创建数据库并创建Hive表

```

1  # 创建数据库
2  CREATE DATABASE IF NOT EXISTS flink;
3  # 使用数据库
4  USE flink;
5  # 创建数据表
6  CREATE TABLE IF NOT EXISTS t_students(
7  sno STRING,
8  sname STRING,
9  gender STRING,

```



```
10  age INT,
11  course STRING
12 ) WITH (
13  'connector' = 'filesystem',
14  'path' = 'hdfs://node1:8020/hive_data/04_students.txt',
15  'format' = 'csv',
16  'csv.field-delimiter' = ',',
17 );
```

- 其他参数设置
- <https://nightlies.apache.org/flink/flink-docs-release-1.13/docs/connectors/table/formats/csv/#csv-allow-comments>

## Format Options

Option	Required	Default	Type	Description
format	required	(none)	String	Specify what format to use, here should be 'csv'.
csv.field-delimiter	optional	,	String	Field delimiter character ('' by default), must be single character. You can use backslash to specify special characters, e.g. '\t' represents the tab character. You can also use unicode to specify them in plain SQL, e.g. 'csv.field-delimiter' = U&'\0001' represents the 0x01 character.
csv.disable-quote-character	optional	false	Boolean	Disabled quote character for enclosing field values (false by default). If true, option 'csv.quote-character' can not be set.
csv.quote-character	optional	"	String	Quote character for enclosing field values ("" by default).
csv.allow-comments	optional	false	Boolean	Ignore comment lines that start with '#' (disabled by default). If enabled, make sure to also ignore parse errors to allow empty rows.
csv.ignore-parse-errors	optional	false	Boolean	Skip fields and rows with parse errors instead of failing. Fields are set to null

- FlinkClient常见配置

1 -- 更改表程序基本执行行为的属性。

```

2
3 SET table.planner = blink; -- 可选: 'blink' (默认) 或 'old'
4 SET execution.runtime-mode = streaming; -- 必选: 执行模式为 'batch' 或
  'streaming'
5 SET sql-client.execution.result-mode = table; -- 必选: 'table', 'changelog'
  或 'tableau'
6 SET sql-client.execution.max-table-result.rows = 10000; -- 可选: 'table' 模式下可维护的最
  大行数 (默认为 1000000, 小于 1 则表示无限制)
7 SET parallelism.default = 1; -- 可选: Flink的并行度 (默认为1)
8 SET pipeline.auto-watermark-interval = 200; -- 可选: 周期水印的间隔
9 SET pipeline.max-parallelism = 10; -- 可选: Flink的最大并行度
10 SET table.exec.state.ttl=1000; -- 可选: 表程序的空闲状态时间
11 SET restart-strategy = fixed-delay;
12
13 -- 用于调整和表程序的配置选项。
14 SET table.optimizer.join-reorder-enabled = true;
15 SET table.exec.spill-compression.enabled = true;
16 SET table.exec.spill-compression.block-size = 128kb;

```

- 使用FLink Client提交作业

```

1 sql-clients.sh embedded -f 脚本文件.sql
2 # 监控每个作业错误的详细信息
3 SET sql-client.verbose = true;

```

## catalogs

- catalogs类似于Hive metastore, 元数据存储, 哪些库、表、字段、类型、数据来自location、创建/修改时间、分隔符、用于权限、列表、路径等
- catalog分类
  - GenericInMemoryCatalog 默认的catalog, 存储到内存中的元数据
  - jdbcatalog 将元数据保存到 postgresql 或 mysql、derby
  - hiveCatalog, 存储Flink的元数据; 直接读取Hive元数据接口
  - 自定义 catalog
- 创建Flink表注册到Catalog
- 案例-使用Java实现注册catalog

```

1 //todo 设置读取hdfs的用户
2 //todo 创建流执行环境
3 //todo 实例化环境设置
4 //todo 创建流表环境

```

```
5 //todo 创建变量catalogName, 数据库名称, 表名
6 //todo 实例化 HiveCatalog
7 //todo 通过tEnv注册catalog
8 //todo 通过tEnv使用catalog
9 //todo 使用catalog创建数据库
10 //todo 创建TableSchema
11
```

```
1 public class CatalogSQLApi {
2     public static void main(String[] args) {
3         //开发步骤
4         //todo 设置读取hdfs的用户, 读写Hadoop集群, 需要具备HDFS的权限
5         System.setProperty("HADOOP_USER_NAME", "root");
6         //todo 创建流执行环境
7         StreamExecutionEnvironment env =
8 StreamExecutionEnvironment.getExecutionEnvironment();
9         env.setParallelism(1);
10        //设置chk和重启策略
11        env.enableCheckpointing(5000);
12        env.getCheckpointConfig().setCheckpointStorage("file:///d:/chk-5");
13        //设置重启策略
14        env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3, 5000));
15        //todo 实例化环境设置
16        EnvironmentSettings settings = EnvironmentSettings.newInstance()
17            .useBlinkPlanner()
18            .inStreamingMode()
19            .build();
20        //todo 创建流表环境
21        StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
22        //todo 创建catalogName, 数据库名称, 表名
23        String catalogName = "myHive1";
24        String dbName = "flink02";
25        String tableName = "t_product";
26        //todo 实例化 HiveCatalog
27        String path = new Path(CatalogSQLApi.class.getClassLoader().getResource("hive-
28 site.xml").getPath()).getParent().getPath();
29
30        HiveCatalog catalog = new HiveCatalog(
31            catalogName,
32            "default",
33
```

```
31         path,
32         "3.1.2"
33     );
34
35     //todo 通过tEnv注册catalog
36     tEnv.registerCatalog(catalogName, catalog);
37     //todo 通过tEnv使用catalog
38     tEnv.useCatalog(catalogName);
39     //todo 使用catalog创建数据库
40     HashMap<String, String> config = new HashMap<>();
41     try {
42         catalog.createDatabase(
43             dbName,
44             new CatalogDatabaseImpl(config, null),
45             true
46         );
47
48         //删除数据库
49         catalog.dropDatabase(dbName, true, true);
50         //获取数据库
51         catalog.getDatabase(dbName);
52         //验证数据库是否存在 true:exists false: non-exists
53         boolean isExists = catalog.databaseExists(dbName);
54         //列出所有数据库
55         List<String> dbList = catalog.listDatabases();
56
57     } catch (DatabaseAlreadyExistException e) {
58         e.printStackTrace();
59     } catch (DatabaseNotEmptyException e) {
60         e.printStackTrace();
61     } catch (DatabaseNotExistException e) {
62         e.printStackTrace();
63     }
64     tEnv.useDatabase(dbName);
65     //todo 创建TableSchema
66     try {
67         catalog.createTable(
68             new ObjectPath(dbName, "/hive_data/flink02"),
69             CatalogTable.of(
70                 Schema.newBuilder()
```

```

71         .column("uid", DataTypes.STRING())
72         .column("uname", DataTypes.STRING())
73         .build(),
74         null,
75         null,
76         null
77     ),
78     true
79 );
80 } catch (TableAlreadyExistsException e) {
81     e.printStackTrace();
82 } catch (DatabaseNotExistException e) {
83     e.printStackTrace();
84 }
85 //todo 通过catalog创建表 ObjectPath catalogTableImpl
86 }
87 }

```

## • FlinkSQL client读取kafka的数据并使用hivecatalog元数据管理

- 案例-读取kafka中的数据并将其写入到Hive中
- 开发步骤
  - 1.配置FLink支持读取kafka的数据，导入jar包
    - 将 flink-sql-connector-kafka\_2.11-1.13.1.tar 拷贝到 /export/server/flink/lib 目录下
    - scp 到三台节点
  - 2.重启Flink集群环境
    - stop-cluster.sh
    - start-cluster.sh
  - 3.开启服务
    - hadoop 集群
    - zookeeper 集群
    - kafka 集群
    - Hive 元数据和hiveserver2 服务
    - Flink 集群
  - 4.登入FlinkSQL client

```
1 ./bin/sql-client.sh embedded -i init.sh
```

- 5.操作kafka中的数据到hive
  - 配置支持 FlinkSQL client --> kafka 环境

```
1 /flink-1.13.0
2 /lib
3     // Flink's Hive connector
4     flink-sql-connector-hive-3.1.2_2.11-1.13.1.jar
5     flink-sql-connector-kafka_2.11-1.13.1.jar
6     flink-shaded-hadoop-3-uber-3.3.0-10.0.jar
7
8     // Hive dependencies
9     hive-exec-3.1.2.jar
10    // add antlr-runtime if you need to use hive dialect
11    antlr-runtime-3.5.2.jar
```

- 重启Flink集群 stop-cluster.sh start-cluster.sh 重启集群
- 读取kafka中的数据

```
1 # 创建数据库
2 CREATE DATABASE IF NOT EXISTS day08;
3 # 使用数据库
4 USE day08;
5 # 创建数据表
6 CREATE TABLE IF NOT EXISTS `t_order` (
7     id INT,
8     category STRING,
9     areaName STRING,
10    money INT,
11    `timestamp` BIGINT,
12    eventTime AS TO_TIMESTAMP(FROM_UNIXTIME(`timestamp` / 1000, 'yyyy-MM-dd HH:mm:ss')), -
13    - 事件时间
14    WATERMARK FOR eventTime AS eventTime - INTERVAL '10' SECOND -- 水印
15 ) WITH (
16     'connector' = 'kafka',
17     'topic' = 'output', -- 指定消费的topic
18     'scan.startup.mode' = 'latest-offset', -- 指定起始offset位置
19     'properties.zookeeper.connect' = 'node1:2181,node2:2181,node3:2181',
20     'properties.bootstrap.servers' = 'node1:9092,node2:9092,node3:9092',
21     'properties.group.id' = '_consumer1_order_info_',
22     'format' = 'json',
23     'json.ignore-parse-errors' = 'true'
```

```

23 );
24
25 # 模拟从 kafka 中消费存储到 Hive 中
26 {"id":1,"timestamp":1588870980000,"category":"电脑","areaName":"石家庄","money":"1450"}
27 {"id":2,"timestamp":1588870860000,"category":"手机","areaName":"北京","money":"1450"}
28 {"id":3,"timestamp":1588870980000,"category":"手机","areaName":"北京","money":"8412"}
29 {"id":4,"timestamp":158885260000,"category":"电脑","areaName":"上海","money":"1513"}
30 {"id":5,"timestamp":1588870980000,"category":"家电","areaName":"北京","money":"1550"}
31 {"id":6,"timestamp":1588870860000,"category":"电脑","areaName":"深圳","money":"1550"}
32 # 生产数据到 kafka 命令
33 bin/kafka-console-producer.sh --broker-list node1:9092 --topic orderinfo
34
35 # FlinkSQL Client 读取消费到的数据
36 SELECT * FROM `t_order`;

```

## 流处理的概念

### • FlinkSQL的流模式，explain的执行计划

- FlinkTable & SQL实现流批一体处理
  - 自动根据数据源source发现数据是有界还是无界，有界就用批处理，否则流处理
  - 显示指定

```

1 EnvironmentSettings.newInstance()
2 .useBlinkPlanner()
3 .inBatchMode() # 批处理 ， 默认就是使用流模式
4 .build()
5

```

#### • 常见概念

- 1.动态表：数据是无界的，动态
  - 动态表就是事实发生变化，在原有表数据的基础上
  - 连续查询
- 2.时间属性

```

1 字段.rowtime() # 指定当前这个字段的时间是 event time 事件时间
2 字段.processTime() # 处理时间

```

```

StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
//4. 注册表 创建临时视图并分配 rowtime
// 列模式 Alt + 鼠标左键 , 全选 ctrl + shift + -> , 选中 ctrl + ->
tEnv.createTemporaryView(
    path: "t_order",
    watermarkStream,
    $( name: "oid"),
    $( name: "uid"),
    $( name: "money"),
    $( name: "createTime", rowtime()) //当前字段当成 event time
);

//5. 编写FlinkSQL, 根据 userId 和 createTime 滚动分组统计 userId、订单总笔数、最大、最小金额
/*Table result = tEnv.sqlQuery(
    "select uid,count(oid) as cnt,max(money) as maxMoney,min(money) as minMoney " +
    "from t_order " +
    "group by uid, tumble(createTime, interval '5' second)"
);*/
//编写 FlinkTable from -> table
Table order = tEnv.from( path: "t_order");
// window 开窗 -> Tumble -> Lit 推断出来类型
Table resultTable = order.window(Tumble.over(Lit(v: 5).second()).on($( name: "createTime")).as(a

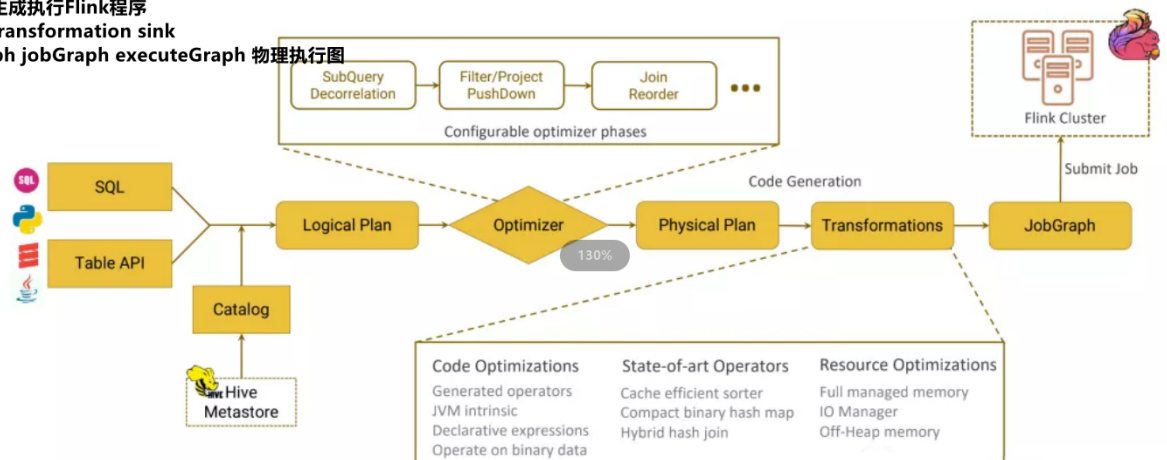
```

- 3.流上的join操作
  - window join窗口内 join 关联
  - interval join 间隔join 上界和下界
- 4.时态表
  - 每个时刻的动态的表

## FlinkSQL的执行原理

### • FlinkSQL工作执行原理

- 1.FlinkTable&SQL 会使用 Hive Metastore 构建catalog
2. SQL转换成抽象语法树 AST
3. 编译成逻辑执行计划
4. 对逻辑执行计划进行优化, 生成了优化的逻辑执行计划
5. 将优化的逻辑执行计划转换成物理执行计划, 根据Flink模板生成执行Flink程序
6. 提交Flink执行 Source transformation sink
7. 生成四个图 streamGraph jobGraph executeGraph 物理执行图



### • Hive的join的底层原理

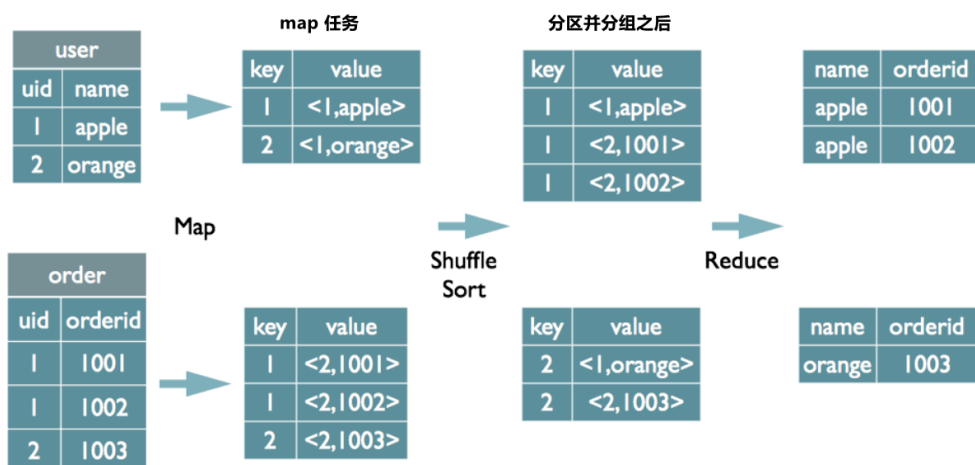


## Hive MR的Join底层原理

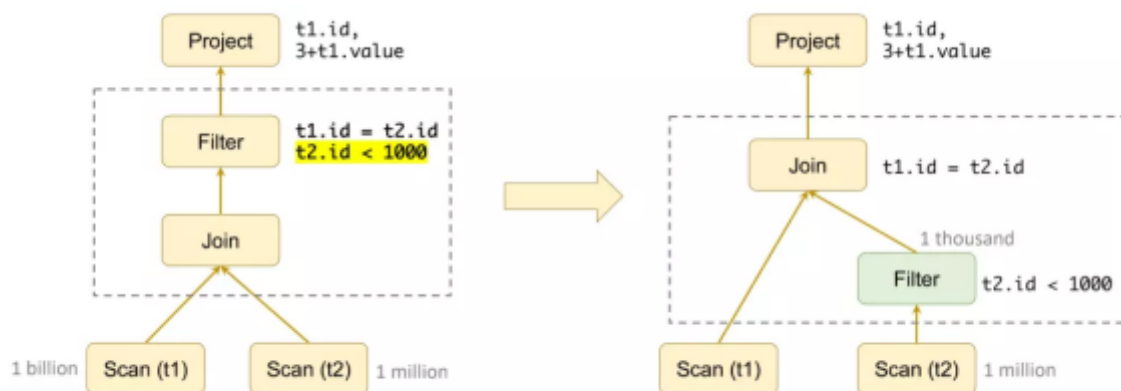
```
select
from user u
join order o on u.uid=o.uid
```

MR任务分为三大阶段八大步骤:

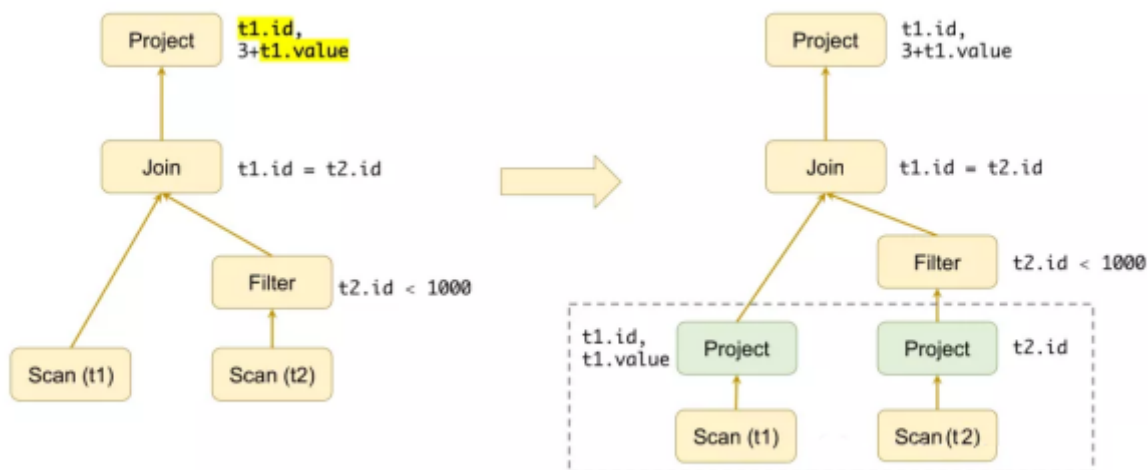
- 1.每个表都需要读取是对应的 HDFS的路径的文件数据
- 2.将每行的数据转换成 k2,v2 ,k2是关联的用户字段;
- 3.根据k2进行分区、排序、分组将相同k2的值放到一起
- 4.根据reduce任务,将相同k2的值按照 project投影字段编排到一张表中



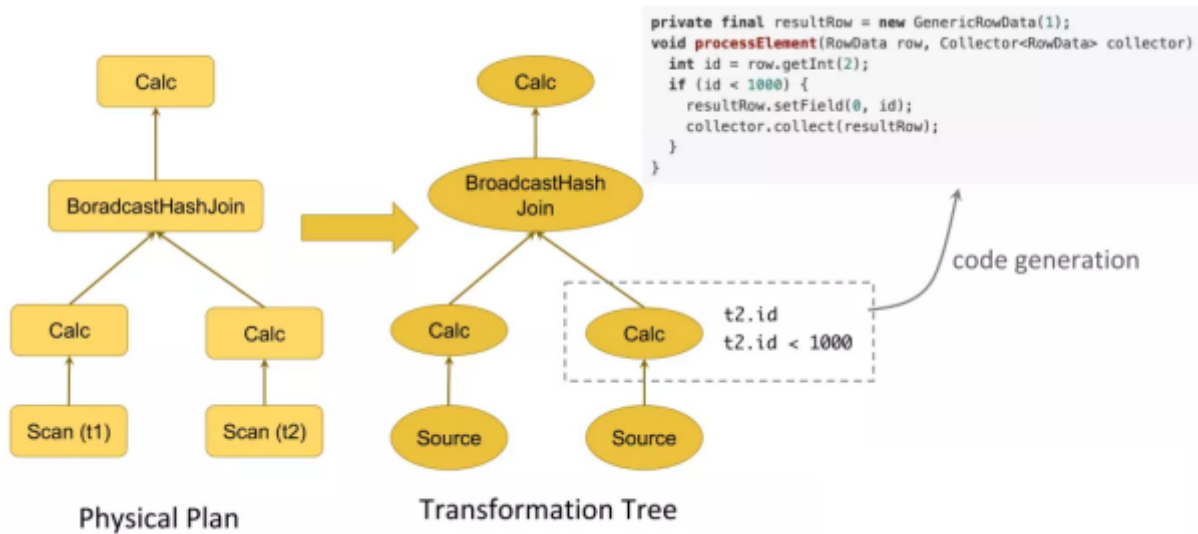
- 逻辑执行计划进行优化
- 1.常量折叠
- 2.谓词下推 filter where 字段, 贴近 scan, 再 join



- 3.投影project下推



- 优化的逻辑执行计划转换成物理执行计划



## ● 遇到的问题

- 创建Flink Table 不能在默认的catalog和database找到 table?
  - 原因：建表失败
    - i. SQL写的有问题
      - 字段和字段之间的空格
      - 表已经存在了，字段不一致，没有重新创建表
      - 数据类型和输入的数据不一致
    - ii. create table if not exists t\_order 在数据库中没有这张表
      - 创建的表的是一个虚拟表，映射FileSystem， kafka， mysql， 对应的路径、topic、数据表本身要存在，Flink内存中创建了一个映射，Hive（CREATE EXTERNAL TABLE if not exists t\_order(表名 表类型 ...) row format delimiter field terminated by ',' stored as TextFile LOCATION '/user/hive/warehouse/ods.db/t\_order/'）
- Flink 打包运行流程
  - a. mvn package 打包，会在 targets 目录下 jar 包
  - b. 将 jar 包上传到 服务器
  - c. Flink run -c cn.itcast.flink.Wordcount /root/job/jar包名称.jar -参数列表 值列表
    - standalone 先开启 start-cluster 本地 standalone 模式
    - yarn
      - yarn-session 再 flink run 执行
      - flink run -m yarn-cluster -c ... /jar
- 输出数据到外部系统，需要指定字段和字段类型

```

);*/
//编写 FlinkTable from -> table
Table order = tEnv.from(path: "t_order");
// window 开窗 -> Tumble -> Lit 推断出来类型
Table resultTable = order.window(Tumble.over(Lit(v: 5).second()).on($ ( name: "createTime")).as( alias: "tumbleWindow"
    .groupBy($ ( name: "uid"), $ ( name: "tumbleWindow"))
    .select($ ( name: "uid"), $( name: "oid").count().as( name: "cnt"),
        $( name: "money").max().as( name: "maxMoney"),
        $( name: "money").min().as( name: "minMoney")
    );

```

Schema

```

//6. 执行查询语句返回结果
//7. Sink to RetractStream -> 将计算后的新的数据在DataStream原数据的基础上更新true或是删除false
DataStream<Tuple2<Boolean, Row>> orderResult = tEnv.toRetractStream(resultTable, Row.class);
//8. 打印输出
orderResult.print();
//9. 执行流环境
env.execute();
}

```

两者之间的类型，字段都不一样  
不能用 Order 对象去接收  
结果集，如何做？  
1. 自定义结果集Result{四个属性}  
2. Row 来接收，通用类型，是可以  
不去指定数据的类型

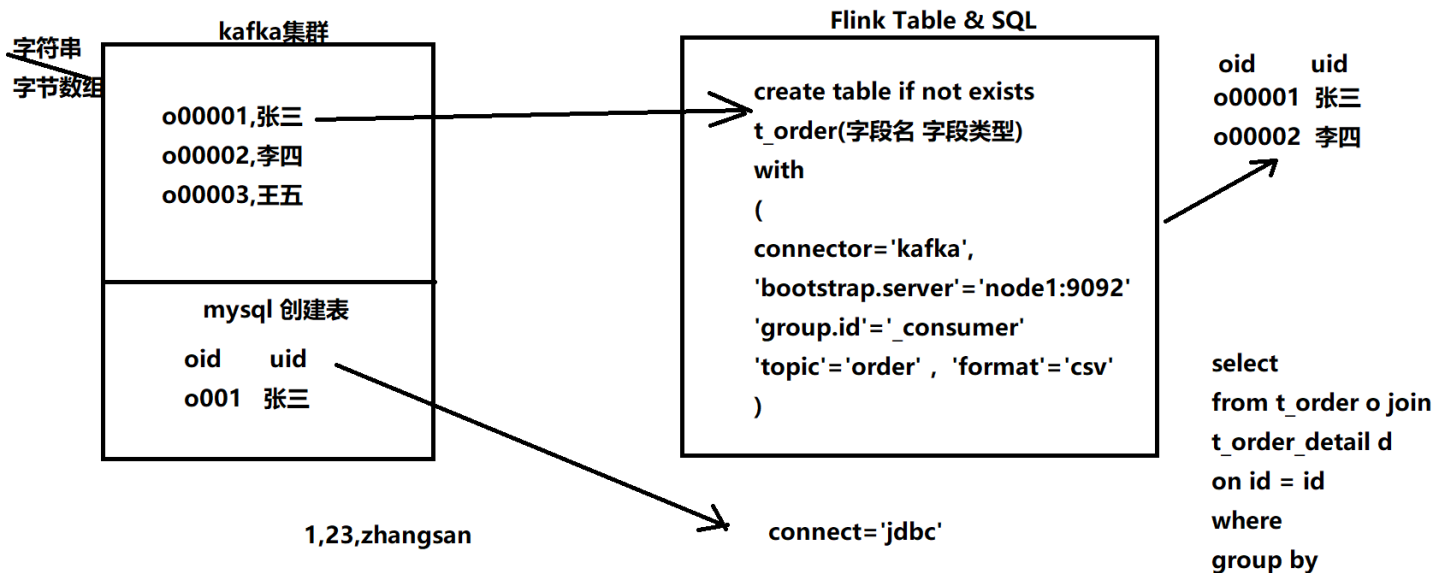
Order

oid	uid	_result_set
uid	money	cnt
money	createTime	maxMoney
createTime		minMoney

//自定义实现 Order，乱序生成订单数据

public static class CustomOrder implements SourceFunction<Order> {

- 将 kafka 或 mysql 的数据映射到 Flink 的临时表



- Flink Table 返回数据类型
  - Row 通用的类型，字段会推断
  - 自定义数据类型

84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105

```
//6. 执行查询语句返回结果
//7. Sink toRetractStream → 将计算后的新的数据在DataStream原数据的基础上更新true或是删除false
DataStream<Tuple2<Boolean, OrderInfo>> orderResult = tEnv.toRetractStream(resultTable, OrderInfo.class);
//8. 打印输出
orderResult.print();
//9. 执行流环境
env.execute();
}
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public static class OrderInfo {
    private String uid;
    private Long cnt;
    private Double maxMoney;
    private Double minMoney;
}
```

自定义数据结构  
保证 sink 输出的字段类型要和 结果集的类型要  
保持一致。

- select 'hello world'
  - 需要将 jar 包拖进去 /export/server/flink/lib
  - 重新启动集群 stop-cluster.sh start-cluster.sh 重启 flink 集群。
  - 重新进入 sql-client embedded
- init.sh 指定绝对路径 去运行

```
1 [root@node1 flink]# bin/sql-client.sh embedded -i /export/server/flink/bin/ini.sh
```