# 环境恢复

### 安装新mysql

```
#命令1
sudo apt-get update
#命令2
sudo apt-get install mysql-server

# 初始化安全配置*（可选）
sudo mysql_secure_installation

# 远程访问和权限问题*（可选）
#前情提要：事先声明一下，这样做是对安全有好处的。刚初始化好的MySQL是不能进行远程登录的。要实现登录的话，强烈建议新建一个权限低一点的用户再进行远程登录。直接使用root用户远程登录有很大的风险。分分钟数据库就有可能被黑客drop掉。
#首先，修改/etc/mysql/my.cnf文件。把bind-address = 127.0.0.1这句给注释掉。解除地址绑定（或者是绑定一个你的固定地址。但宽带上网地址都是随机分配的，固定ip不可行）。
#然后，给一个用户授权使他能够远程登录。执行下面两句即可。

grant all PRIVILEGES on *.* to user1@'%'identified by '123456' WITH GRANT
OPTION;
FLUSH  PRIVILEGES;
service mysql restart。
```

### 重新启动Hive

STILL ON `37测试机` `/home/anxin/apache-hive-3.1.2-bin`

```
./schematool -initSchema -dbType mysql
# 加载我的环境变量，应为本机还安装了ambari的hive
source /etc/profile
hive --service metastore
```

# Hive基础操作

参考：https://www.cnblogs.com/wangrd/p/6275162.html

```
--就会在HDFS的[/user/hive/warehouse/]中生成一个tabletest.db文件夹。
CREATE DATABASE tableset;

-- 切换当前数据库
USE tableset;

-- 创建表
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
```

```
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path]

CREATE TABLE t_order (
    id int,
    name string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' -- 指定字段分隔符
STORED AS TEXTFILE; -- 指定数据存储格式

-- 查看表结构
DESC t_order;

-- 导入数据
load data local inpath '/home/anxin/data/data.txt' [OVERWRITE] into table
t_order;

-- EXTERNAL表
-- 创建外部表，不会对源文件位置作任何改变
-- 删除外部表不会删除源文件
CREATE EXTERNAL TABLE ex_order (
    id int,
    name string
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/external/hive';

--分区
CREATE TABLE t_order(id int,name string) partitioned by (part_flag string)
row format delimited fields terminated by '\t';
load data local inpath '/home/hadoop/ip.txt' overwrite into table t_order
partition(part_flag='part1'); -- 数据上传到part1子目录下

-- 查看所有表
SHOW TABLES;
SHOW TABLES 'TMP';
SHOW PARTITIONS TMP_TABLE;-- 查看表有哪些分区
DESCRIBE TMP_TABLE; -- 查看表结构

-- 分桶表
create table stu_buck(Sno int,Sname string,Sex string,Sage int,Sdept string)
clustered by(Sno)
sorted by(Sno DESC)
into 4 buckets
row format delimited
fields terminated by ',';
-- 通过insert into ...select...进行数据插入
set hive.enforce.bucketing = true;
set mapreduce.job.reduces=4;
insert overwrite table stu_buck
select * from student cluster by(Sno); --等价于 distribute by(Sno) sort by(Sno
asc);

-- 删除表
DROP TABLE tablename;
```

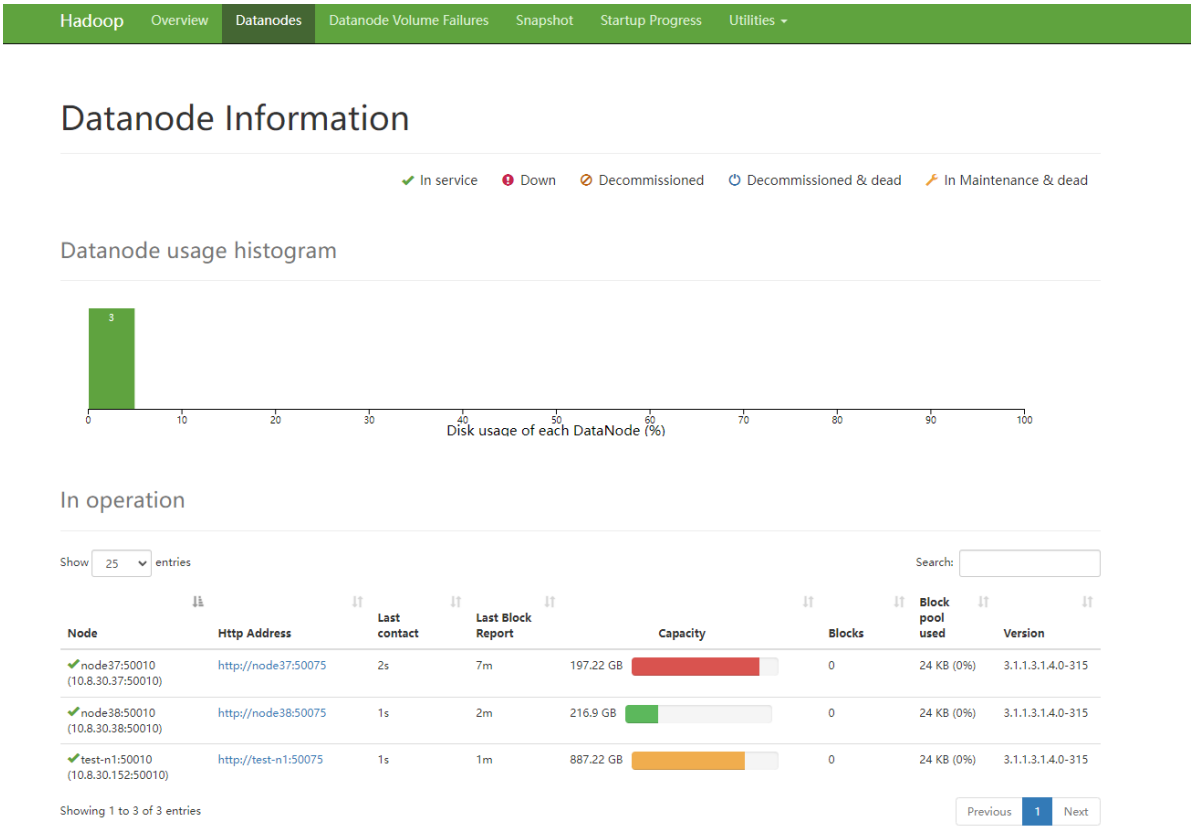
```
-- 临时表
CREATE TABLE tmp_table
AS
SELECT id,name
FROM t_order
SORT BY new_id;

-- UDF 用户定义函数
-- 基层UDF函数，打包jar到程序，注册函数
CREATE TEMPORARY function tolowercase as 'cn.demo.Namespace';

select id,tolowercase(name) from t_order;
```

## Hadoop基础操作

本例中最终选择通过Hadoop Catalog实现IceBerg数据存储：

```
# -skipTrash 直接删除不放到回收站
hdfs dfs -rm -skipTrash /path/to/file/you/want/to/remove/permanently
# 清理所有Trash中的数据
hdfs dfs -expunge

## **清理指定文件夹下的所有数据**
hdfs dfs -rm -r -skipTrash /user/hadoop/*

## hadoop 启动错误：
chown -R hdfs:hdfs /hadoop/hdfs/namenode
# DataNode启动失败：可能多次format导致。修改data-node的clusterid和namenode中的一致
/hadoop/hdfs/data/current/VERSION
/hadoop/hdfs/namenode/current/VERSION

# 查看DataNode启动日志
root@node38:/var/log/hadoop/hdfs# tail -n 1000 hadoop-hdfs-datanode-node38.log
```

查看恢复的Hadoop集群:




# Flink SQL流式从Kafka到Hive

https://www.cnblogs.com/Springmoon-venn/p/13726089.html

读取kafka的sql：

```
tableEnv.getConfig.setSqlDialect(SqlDialect.DEFAULT)

create table myhive.testhive.iotaKafkatable(
`userId` STRING,
`dimensionId` STRING,
`dimCapId` STRING,
`scheduleId` STRING,
`jobId` STRING,
`jobRepeatId` STRING,
`thingId` STRING ,
`deviceId` STRING,
`taskId` STRING,
`triggerTime` STRING,
`finishTime` STRING,
`seq` STRING,
`result` STRING,
 `data` STRING
)with
('connector' = 'kafka',
'topic'='iceberg',
'properties.bootstrap.servers' = '10.8.30.37:6667',
'properties.group.id' = 'iceberg-demo' ,
'scan.startup.mode' = 'latest-offset',
'format' = 'json',
'json.ignore-parse-errors'='true')
```

创建hive表：

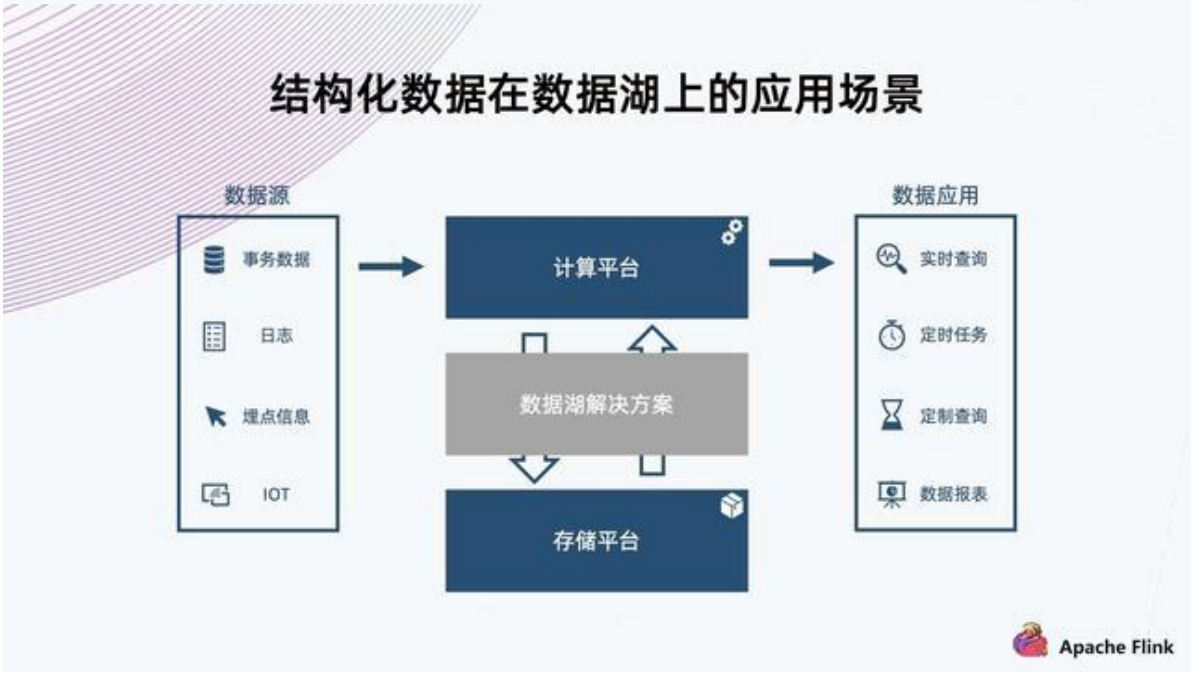
```
tableEnv.getConfig.setSqlDialect(SqlDialect.HIVE)

CREATE TABLE  myhive.testhive.iotatable2(
`userId` STRING,
`dimensionId` STRING,
`dimCapId` STRING,
`scheduleId` STRING,
`jobId` STRING,
`jobRepeatId` STRING,
`thingId` STRING ,
`deviceId` STRING,
`taskId` STRING,
`triggerTime` TIMESTAMP,
`seq` STRING,
`result` STRING,
 `data` STRING
)
PARTITIONED BY ( finishTime STRING)  -- 分区间字段，该字段不存放实际的数据内容
STORED AS PARQUET
TBLPROPERTIES (
      'sink.partition-commit.policy.kind' = 'metastore,success-file',
      'partition.time-extractor.timestamp-pattern' = '$finishTime'
    )
```

## IceBerg

概念再解析：

> 好文推荐：
>
> - 数据湖存储架构选型
> -

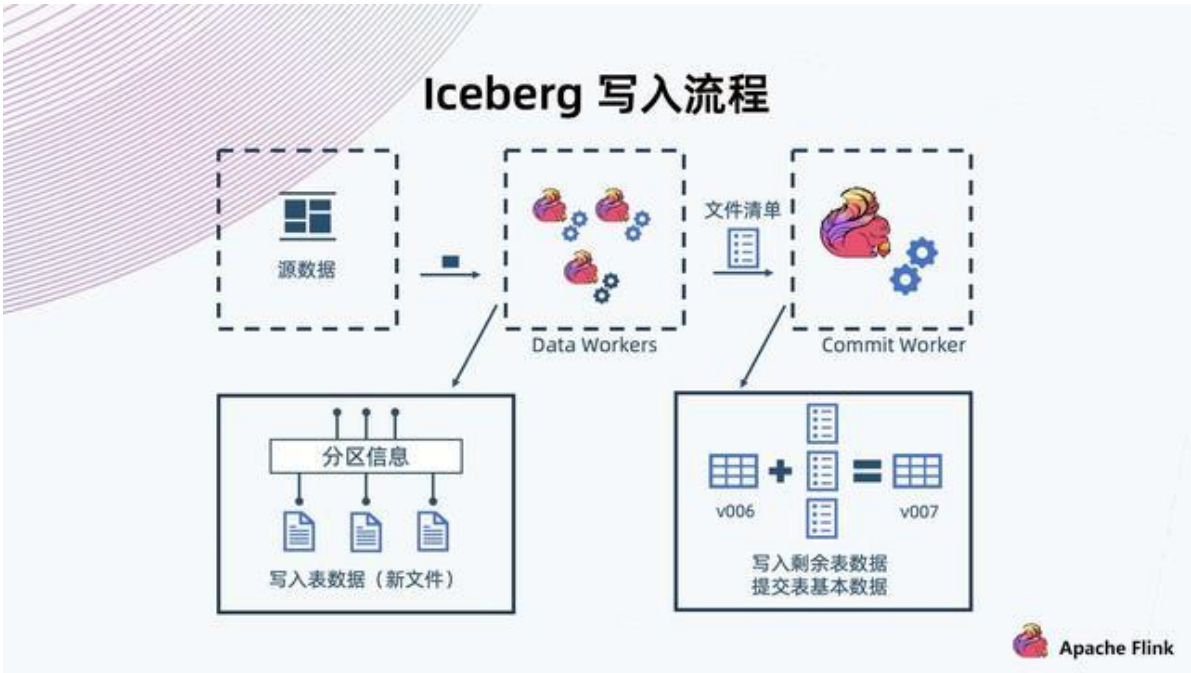
参考：Flink+IceBerg+对象存储，构建数据湖方案


IceBerg表数据组织架构：

命名空间-》表-》快照》表数据(Parquet/ORC/Avro等格式)

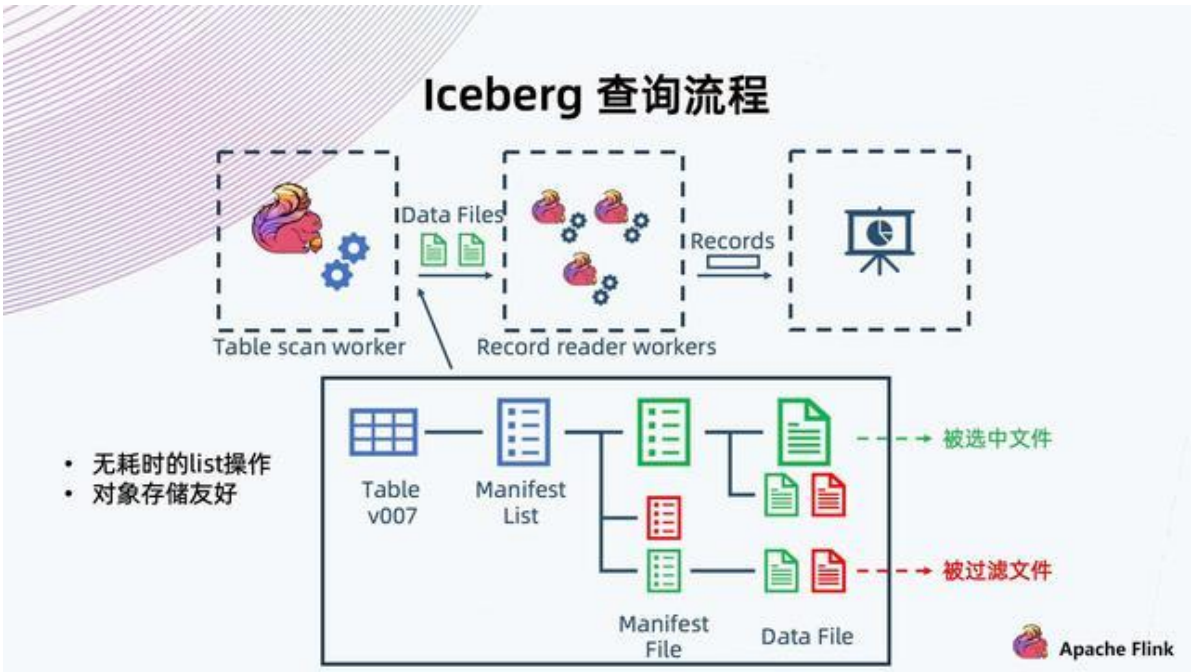
- **快照 Metadata**：表格 Schema、Partition、Partition spec、Manifest List 路径、当前快照等。
- **Manifest List**：Manifest File 路径及其 Partition，数据文件统计信息。
- **Manifest File**：Data File 路径及其每列数据上下边界。
- **Data File**：实际表内容数据，以 Parque，ORC，Avro 等格式组织。

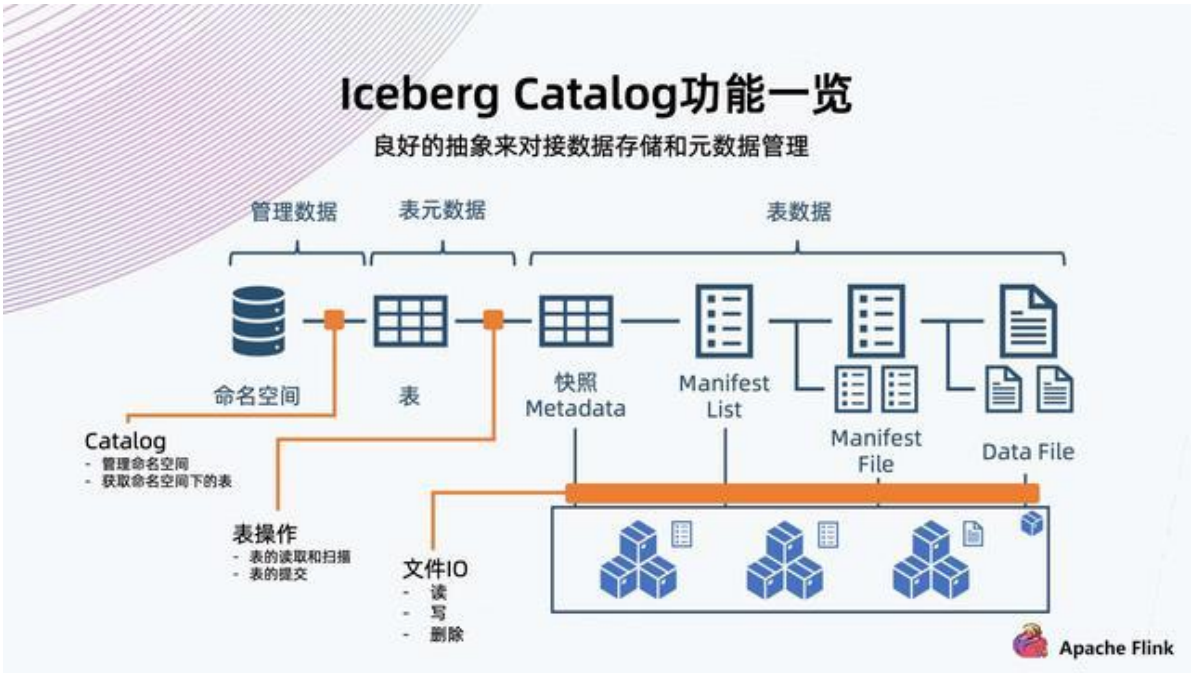



由DataWorker读取元数据进行解析，让后把一条记录提交给IceBerg存储，IceBerg将记录写入预定义的分区，形成一些新文件。

Flink在执行Checkpoint的时候完成这一批文件的写入，然后生成这批文件的清单，提交给Commit Worker.

CommitWorker读出当前快照信息，然后与本次生成的文件列表进行合并，生成新的ManifestList文件以及后续元数据的表文件的信息。之后进行提交，成功后形成新快照。






catalog是Iceberg对表进行管理（create、drop、rename等）的一个组件。目前Iceberg主要支持HiveCatalog和HadoopCatalog两种。

HiveCatalog通过metastore数据库（一般MySQL）提供ACID，HadoopCatalog基于乐观锁机制和HDFS rename的原子性保障写入提交的ACID。

Flink兼容性

Iceberg's Flink integration is compatible with Flink using the modules in the following table:

| Iceberg version | Flink 1.11.x | Flink 1.12.x | Flink 1.13.x | Flink 1.14.x |
|---|---|---|---|---|
| master branch | | iceberg-flink-runtime-1.12 | iceberg-flink-runtime-1.13 | iceberg-flink-runtime-1.14 |
| 0.12.x | | iceberg-flink-runtime | | |
| 0.11.x | iceberg-flink-runtime | | | |
| 0.10.x | iceberg-flink-runtime | | | |

# 写入IceBerg

- IceBerg官网 https://iceberg.apache.org/#flink/
- 官网翻译 https://www.cnblogs.com/swordfall/p/14548574.html
- 基于HiveCatalog的问题（未写入Hive） https://issueexplorer.com/issue/apache/iceberg/3092
- Flink + Iceberg: How to Construct a Whole-scenario Real-time Data Warehouse
- 被他玩明白了 https://miaowenting.site/2021/01/20/Apache-Iceberg/

## 1.使用HadoopCatalog

https://cloud.tencent.com/developer/article/1807008

关键代码：

svn: http://svn.anxinyun.cn/Iota/branches/fs-iot/code/flink-iceberg/flink-iceberg/src/main/scala/com/fs/IceBergDealHadoopApplication.scala

```
...
```

## 2. 使用HiveCatalog

> 进展：??? Hive中可以查询到数据。在FlinkSQL中查询不到数据

关键代码说明：

```
env.enableCheckpointing(5000)
 // 创建IceBerg Catalog和Database
val createIcebergCatalogSql =
"""CREATE CATALOG iceberg WITH(
              | 'type'='iceberg',
              | 'catalog-type'='hive',
              | 'hive-conf-dir'='E:\Iota\branches\fs-iot\code\flink-
iceberg\flink-iceberg'
              |)
            """.stripMargin

// 创建原始数据表 iota_raw
val createIotaRawSql =
            """CREATE TABLE iceberg.iceberg_dba.iota_raw (
              |`userId` STRING,
              |`dimensionId` STRING,
              |`dimCapId` STRING,
              |`scheduleId` STRING,
              |`jobId` STRING,
              |`jobRepeatId` STRING,
              |`thingId` STRING ,
              |`deviceId` STRING,
              |`taskId` STRING,
              |`triggerTime` TIMESTAMP,
              |`day` STRING,
              |`seq` STRING,
              |`result` STRING,
              | `data` STRING
              |) PARTITIONED BY (`thingId`,`day`)
              |WITH (
              | 'engine.hive.enabled' = 'true',
              | 'table.exec.sink.not-null-enforcer'='ERROR'
              |)
            """.stripMargin

 val kafka_iota_sql =
            """create table myhive.testhive.iotaKafkatable(
              |`userId` STRING,
              |`dimensionId` STRING,
```

```
              |`dimCapId` STRING,
              |`scheduleId` STRING,
              |`jobId` STRING,
              |`jobRepeatId` STRING,
              |`thingId` STRING ,
              |`deviceId` STRING,
              |`taskId` STRING,
              |`triggerTime` STRING,
              |`finishTime` STRING,
              |`seq` STRING,
              |`result` STRING,
              | `data` STRING
              |)with
              |('connector' = 'kafka',
              |'topic'='iceberg',
              |'properties.bootstrap.servers' = '10.8.30.37:6667',
              |'properties.group.id' = 'iceberg-demo' ,
              |'scan.startup.mode' = 'latest-offset',
              |'format' = 'json',
              |'json.ignore-parse-errors'='true'
              |)
            """.stripMargin

// 注册自定义函数 Transform
        tenv.createTemporarySystemFunction("dcFunction",
classOf[DateCgFunction])
        tenv.createTemporarySystemFunction("tcFunction",
classOf[TimeStampFunction])
val insertSql =
            """
              |insert into iceberg.iceberg_dba.iota_raw
              | select userId,
dimensionId,dimCapId,scheduleId,jobId,jobRepeatId,thingId,deviceId,taskId,
              |tcFunction(triggerTime),
              |DATE_FORMAT(dcFunction(triggerTime),'yyyy-MM-dd'),
              |seq,`result`,data
              |from myhive.testhive.iotakafkatable
            """.stripMargin
```

1. 使用HiveCatalog方式，必须指定 'engine.hive.enabled' = 'true'
2. 'table.exec.sink.not-null-enforcer'='ERROR' 在非空字段插入空值时的处理办法
3. 自定义函数实现

```
class TimeStampFunction extends ScalarFunction {
    def eval(@DataTypeHint(inputGroup = InputGroup.UNKNOWN) o: String):
Timestamp = {
        val v = DateParser.parse(o)
        if (v.isEmpty) {
            null
        } else {
            new Timestamp(v.get.getMillis)
        }
    }
}
```

4. PARTITIONED BY (`thingId`,`day`) 根据thingid和日期分区，文件路径如： http://10.8.30.37:50070/explorer.html#/user/hive/warehouse/iceberg_dba.db/iota_raw/data/thingId=b6cfc716-3766-4949-88bc-71cb0dbf31ee/day=2022-01-20
5. 详细代码见 http://svn.anxinyun.cn/Iota/branches/fs-iot/code/flink-iceberg/flink-iceberg/src/main/scala/com/fs/DataDealApplication.scala

查看创建表结构的语句

```
show create table iota_raw;

CREATE EXTERNAL TABLE `iota_raw`(
  `userid` string COMMENT 'from deserializer',
  `dimensionid` string COMMENT 'from deserializer',
  `dimcapid` string COMMENT 'from deserializer',
  `scheduleid` string COMMENT 'from deserializer',
  `jobid` string COMMENT 'from deserializer',
  `jobrepeatid` string COMMENT 'from deserializer',
  `thingid` string COMMENT 'from deserializer',
  `deviceid` string COMMENT 'from deserializer',
  `taskid` string COMMENT 'from deserializer',
  `triggertime` timestamp COMMENT 'from deserializer',
  `day` string COMMENT 'from deserializer',
  `seq` string COMMENT 'from deserializer',
  `result` string COMMENT 'from deserializer',
  `data` string COMMENT 'from deserializer')
ROW FORMAT SERDE
  'org.apache.iceberg.mr.hive.HiveIcebergSerDe'
STORED BY
  'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'

LOCATION
  'hdfs://node37:8020/user/hive/warehouse/iceberg_dba.db/iota_raw'
TBLPROPERTIES (
  'engine.hive.enabled'='true',
  'metadata_location'='hdfs://node37:8020/user/hive/warehouse/iceberg_dba.db/iota_raw/metadata/00010-547022ad-c615-4e2e-854e-8f85592db7b6.metadata.json',
  'previous_metadata_location'='hdfs://node37:8020/user/hive/warehouse/iceberg_dba.db/iota_raw/metadata/00009-abfb6af1-13dd-439a-88f5-9cb822d6c0e4.metadata.json',
  'table_type'='ICEBERG',
  'transient_lastDdlTime'='1642579682')
```

在Hive中查看数据

```
hive> add jar /tmp/iceberg-hive-runtime-0.12.1.jar;
hive> select * from iota_raw;
```

### 报错记录

1. HiveTableOperations$WaitingForLockException

```
-- HiveMetaStore中的HIVE_LOCKS表 将报错的表所对应的锁记录删除
select
hl_lock_ext_id,hl_table,hl_lock_state,hl_lock_type,hl_last_heartbeat,hl_bloc
kedby_ext_id from HIVE_LOCKS;

delete from HIVE_LOCKS;
```

## 查询IceBerg

### 启动Flink SQL Client

flink 配置master `localhost:8081`,配置workers `localhost`.

配置flink.conf (可选)

```
# The number of task slots that each TaskManager offers. Each slot runs one
parallel pipeline.

taskmanager.numberOfTaskSlots: 4

# The parallelism used for programs that did not specify and other parallelism.

parallelism.default: 1
```

配置sql-client-defaults.yaml (可选)

```
execution:
  # select the implementation responsible for planning table programs
  # possible values are 'blink' (used by default) or 'old'
  planner: blink
  # 'batch' or 'streaming' execution
  type: streaming
  # allow 'event-time' or only 'processing-time' in sources
  time-characteristic: event-time
  # interval in ms for emitting periodic watermarks
  periodic-watermarks-interval: 200
  # 'changelog', 'table' or 'tableau' presentation of results
  result-mode: table
  # maximum number of maintained rows in 'table' presentation of results
  max-table-result-rows: 1000000
  # parallelism of the program
  # parallelism: 1
  # maximum parallelism
  max-parallelism: 128
  # minimum idle state retention in ms
  min-idle-state-retention: 0
  # maximum idle state retention in ms
```

```
  max-idle-state-retention: 0
  # current catalog ('default_catalog' by default)
  current-catalog: default_catalog
  # current database of the current catalog (default database of the catalog by
default)
  current-database: default_database
  # controls how table programs are restarted in case of a failures
  # restart-strategy:
    # strategy type
    # possible values are "fixed-delay", "failure-rate", "none", or "fallback"
(default)
    # type: fallback
```

启动flink集群:

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

访问Flink UI http://node37:8081

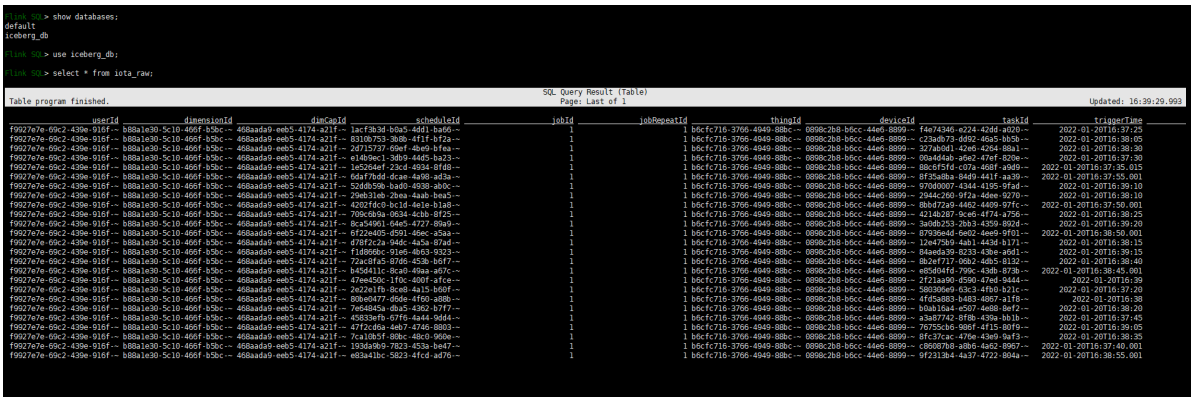
启动sql-client

```
export HADOOP_CLASSPATH=`hadoop classpath`

./bin/sql-client.sh embedded \
-j /home/anxin/iceberg/iceberg-flink-runtime-0.12.0.jar \
-j /home/anxin/iceberg/flink-sql-connector-hive-2.3.6_2.11-1.11.0.jar \
-j /home/anxin/flink-1.11.4/lib/flink-sql-connector-kafka-0.11_2.11-1.11.4.jar \
shell
```

## 查询语句基础

```
CREATE CATALOG iceberg WITH(
 'type'='iceberg',
 'catalog-type'='hadoop',
 'warehouse'='hdfs://node37:8020/user/hadoop',
 'property-version'='1'
);
use catalog iceberg;
use iceberg_db; -- 选择数据库


--可选区域
SET; -- 查看当前配置
SET sql-client.execution.result-mode = table; -- changelog/tableau
SET sql-client.verbose=true; -- 打印异常堆栈
SET sql-client.execution.max-table-result.rows=1000000; -- 在表格模式下缓存的行数
SET table.planner = blink; -- planner: either blink (default) or old
SET execution.runtime-mode = streaming; -- execution mode either batch or
streaming
SET sql-client.execution.result-mode = table; -- available values: table,
changelog and tableau
SET parallelism.default = 1; -- optional: Flinks parallelism (1 by default)
SET pipeline.auto-watermark-interval = 200; --optional: interval for periodic
watermarks
```

```
SET pipeline.max-parallelism = 10; -- optional: Flink's maximum parallelism
SET table.exec.state.ttl = 1000; -- optional: table program's idle state time
SET restart-strategy = fixed-delay;

SET table.optimizer.join-reorder-enabled = true;
SET table.exec.spill-compression.enabled = true;
SET table.exec.spill-compression.block-size = 128kb;

SET execution.savepoint.path = tmp/flink-savepoints/savepoint-cca7bc-
bb1e257f0dab; -- restore from the specific savepoint path
-- 执行一组SQL指令
BEGIN STATEMENT SET;
  -- one or more INSERT INTO statements
  { INSERT INTO|OVERWRITE <select_statement>; }+
END;
```




## 批量读取

> 在FlinkSQL执行SET后再执行查询，总是报错：
>
> ;
>
> 所以需要在执行SQL Client之前设置一些参数

修改 `conf/sql-client-defaults.yaml`

execution.type=batch

```
catalogs:
# A typical catalog definition looks like:
  - name: myhive
    type: hive
    hive-conf-dir: /home/anxin/apache-hive-3.1.2-bin/conf
    #    default-database: ...
  - name: hadoop_catalog
    type: iceberg
    warehouse: hdfs://node37:8020/user/hadoop
    catalog-type: hadoop

#==============================================================================
# Modules
#==============================================================================
```

```yaml
# Define modules here.

#modules: # note the following modules will be of the order they are specified
#  - name: core
#    type: core

#==============================================================================
# Execution properties
#==============================================================================

# Properties that change the fundamental execution behavior of a table program.

execution:
  # select the implementation responsible for planning table programs
  # possible values are 'blink' (used by default) or 'old'
  planner: blink
  # 'batch' or 'streaming' execution
  type: batch
  # allow 'event-time' or only 'processing-time' in sources
  time-characteristic: event-time
  # interval in ms for emitting periodic watermarks
  periodic-watermarks-interval: 200
  # 'changelog', 'table' or 'tableau' presentation of results
  result-mode: table
  # maximum number of maintained rows in 'table' presentation of results
  max-table-result-rows: 1000000
  # parallelism of the program
  # parallelism: 1
  # maximum parallelism
  max-parallelism: 128
  # minimum idle state retention in ms
  min-idle-state-retention: 0
  # maximum idle state retention in ms
  max-idle-state-retention: 0
  # current catalog ('default_catalog' by default)
  current-catalog: default_catalog
  # current database of the current catalog (default database of the catalog by
default)
  current-database: default_database
  # controls how table programs are restarted in case of a failures
  # restart-strategy:
    # strategy type
    # possible values are "fixed-delay", "failure-rate", "none", or "fallback"
(default)
    # type: fallback

#==============================================================================
# Configuration options
#==============================================================================

# Configuration options for adjusting and tuning table programs.

# A full list of options and their default values can be found
# on the dedicated "Configuration" web page.

# A configuration can look like:
configuration:
   table.exec.spill-compression.enabled: true
```

```
    table.exec.spill-compression.block-size: 128kb
    table.optimizer.join-reorder-enabled: true
    #   execution.checkpointing.interval: 10s
    table.dynamic-table-options.enabled: true
```

### 流式读取

修改 `conf/sql-client-defaults.yaml`

execution.type=streaming

execution.checkpointing.interval: 10s

table.dynamic-table-options.enabled: true // 开启动态表（Dynamic Table）选项

```
-- Submit the flink job in streaming mode for current session.
SET execution.type = streaming ;

-- Enable this switch because streaming read SQL will provide few job options in
flink SQL hint options.
SET table.dynamic-table-options.enabled=true;

-- Read all the records from the iceberg current snapshot, and then read
incremental data starting from that snapshot.
SELECT * FROM iota_raw /*+ OPTIONS('streaming'='true', 'monitor-
interval'='10s')*/ ;

-- Read all incremental data starting from the snapshot-id '3821550127947089987'
(records from this snapshot will be excluded).
SELECT * FROM iota_raw /*+ OPTIONS('streaming'='true', 'monitor-interval'='10s',
'start-snapshot-id'='3821550127947089987')*/ ;
```

### 通过外部Hive表查询

```
-- HIVE SHELL
add jar /tmp/iceberg-hive-runtime-0.12.1.jar;

use iceberg_dba;

SET engine.hive.enabled=true;
SET iceberg.engine.hive.enabled=true;
SET iceberg.mr.catalog=hive;

 CREATE EXTERNAL TABLE iceberg_dba.iota_rawe(
     `userId` STRING,
    `dimensionId` STRING,
    `dimCapId` STRING,
    `scheduleId` STRING,
    `jobId` STRING,
    `jobRepeatId` STRING,
    `thingId` STRING ,
    `deviceId` STRING,
```

```
    `taskId` STRING,
    `triggerTime` TIMESTAMP,
    `day` STRING,
    `seq` STRING,
    `result` STRING,
     `data` STRING
)
  STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
  LOCATION '/user/hadoop/iceberg_db/iota_raw'
  TBLPROPERTIES (
      'iceberg.mr.catalog'='hadoop',

'iceberg.mr.catalog.hadoop.warehouse.location'='hdfs://node37:8020/user/hadoop/iceberg_db/iota_raw'
 );
```

## 处理小文件的三种方式

https://zhuanlan.zhihu.com/p/349420627

1. Iceberg表中设置 write.distribution-mode=hash

```
CREATE TABLE sample (
    id BIGINT,
    data STRING
) PARTITIONED BY (data) WITH (
    'write.distribution-mode'='hash'
);
```

2. 定期对 Apache Iceberg 表执行 Major Compaction 来合并 Apache iceberg 表中的小文件。这个作业目前是一个 Flink 的批作业，提供 Java API 的方式来提交作业，使用姿势可以参考文档[8]。
3. 在每个 Flink Sink 流作业之后，外挂算子用来实现小文件的自动合并。这个功能目前暂未 merge 到社区版本，由于涉及到 format v2 的 compaction 的一些讨论，我们会在 0.12.0 版本中发布该功能。

> Iceberg provides API to rewrite small files into large files by submitting flink batch job. The behavior of this flink action is the same as the spark's rewriteDataFiles.

```
import org.apache.iceberg.flink.actions.Actions;

TableLoader tableLoader =
TableLoader.fromHadoopTable("hdfs://nn:8020/warehouse/path");
Table table = tableLoader.loadTable();
RewriteDataFilesActionResult result = Actions.forTable(table)
        .rewriteDataFiles()
        .execute();
```

For more doc about options of the rewrite files action, please see RewriteDataFilesAction

## 插播“ Flink操作HUDI

流式读取

```
CREATE TABLE t1(
  uuid VARCHAR(20),
  name VARCHAR(10),
  age INT,
  ts TIMESTAMP(3),
  `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
  'connector' = 'hudi',
  'path' = 'oss://vvr-daily/hudi/t1',
  'table.type' = 'MERGE_ON_READ',
  'read.streaming.enabled' = 'true',  -- this option enable the streaming read
  'read.streaming.start-commit' = '20210316134557' -- specifies the start commit
instant time
  'read.streaming.check-interval' = '4' -- specifies the check interval for
finding new source commits, default 60s.
);

-- Then query the table in stream mode
select * from t1;
```

## 报错记录：

1. java.lang.ClassNotFoundException: org.apache.hadoop.conf.Configurable

2. 执行 Flink SQL 报错 `[ERROR] Could not execute SQL statement. Reason: java.net.ConnectException: Connection refused`

   启动flink集群：

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

3. 执行batch查询时:

   ```
   val bsSetting =
   EnvironmentSettings.newInstance().useBlinkPlanner().inBatchMode().build()
   val tenv = TableEnvironment.create(bsSetting)
   ```

   ```
   Error:(22, 43) Static methods in interface require -target:jvm-1.8
           val tenv = TableEnvironment.create(bsSetting)
   ```

   未解决：继续使用StreamTableEnvironment

4. MiniCluster is not yet running or has already been shut down

   本地同时调试写入和查询两个Flink程序。不能同时调试两个程序

   ？？

5. flink SQL 程序执行报错 Job client must be a CoordinationRequestGateway. This is a bug

   通过命令行提交执行：

   ```
   ./bin/flink run -c com.fs.OfficialRewriteData -p 1 ./flink-iceberg-1.0-SNAPSHOT-shaded.jar --host localhost --port 8081
   ```

6. 任务提交时，Unable to instantiate java compiler

   ```
   Unable to instantiate java compiler:  calcite依赖冲突
   ```

   参考：https://blog.csdn.net/weixin_44056920/article/details/118110262

7. Flink报错OOM

   放大Flink内存

   ```
   jobmanager.memory.process.size: 2600m
   taskmanager.memory.jvm-metaspace.size: 1000m
   jobmanager.memory.jvm-metaspace.size: 1000m
   ```

8. 网路上的问题汇总帖

   > IceBerg+Kafka+FlinkSQL https://blog.csdn.net/qq_33476283/article/details/119138610

## 大数据湖最佳实践

实施数据湖的路线图

- 建设基础设施（Hadoop集群）
- 组织好数据湖的各个区域（给不同的用户群创建各种区域，并导入数据）
- 设置好数据湖的自助服务（创建数据资产目录、访问控制机制、准备分析师使用的工具）
- 将数据湖开放给用户

规划数据湖：

- 原始区：保存采集的数据
- 产品区：清洗处理后的数据
- 工作区：数据科学家在此分析数据，一般按用户、项目、主题划分。投产后迁移至产品区
- 敏感区.

传统数据库是基于Schema On Write，数据湖（Hadoop等）是Schema On Read.

Michael Hausenblas:

> 数据湖一般与静态数据相关联。其基本思想是引入数据探索的自助服务方法，使相关的业务数据集可以在组织中共享

- 数据存储 HDFS HBase Cassandra Kafka
- 处理引擎 Spark Flink Beam
- 交互 Zeppelin/Spark noteboook,Tableau/Datameer

## 附录

1. hive-site.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><?xml-stylesheet
type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>123456</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://10.8.30.37:3306/metastore_db?
createDatabaseIfNotExist=true</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property>
        <name>hive.metastore.schema.verification</name>
        <value>false</value>
    </property>
    <property>
        <name>hive.cli.print.current.db</name>
        <value>true</value>
    </property>
    <property>
        <name>hive.cli.print.header</name>
        <value>true</value>
    </property>

    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
    </property>

    <property>
        <name>hive.metastore.local</name>
        <value>false</value>
    </property>

    <property>
        <name>hive.metastore.uris</name>
        <value>thrift://10.8.30.37:9083</value>
    </property>
```

```
    <!-- hiveserver2 -->
    <property>
        <name>hive.server2.thrift.port</name>
        <value>10000</value>
    </property>
    <property>
        <name>hive.server2.thrift.bind.host</name>
        <value>10.8.30.37</value>
    </property>
</configuration>
```