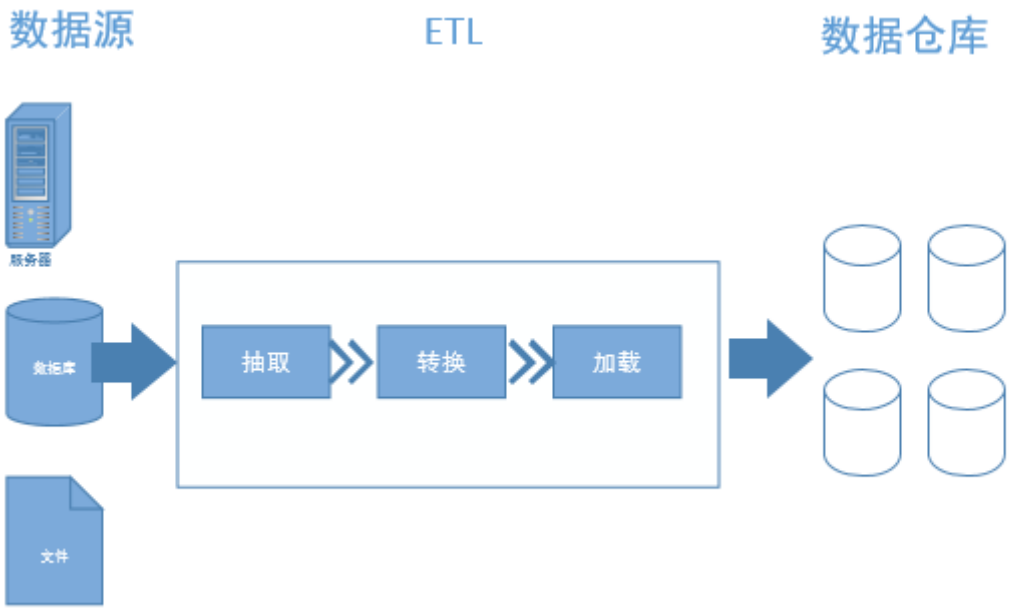
# 数仓ETL工具sqoop和可视化工具Hue

- Hive端口10000, matestore端口9083 , hdfs 端口9870 ,yarn 端口19888,hue端口8889或8888 , CM端口7180, hdfs内部通信端口8020， yarn端口8088,yarn历史端口19888,spark端口18080,spark运行时端口4040

## 2.[掌握]ETL的概念和设计

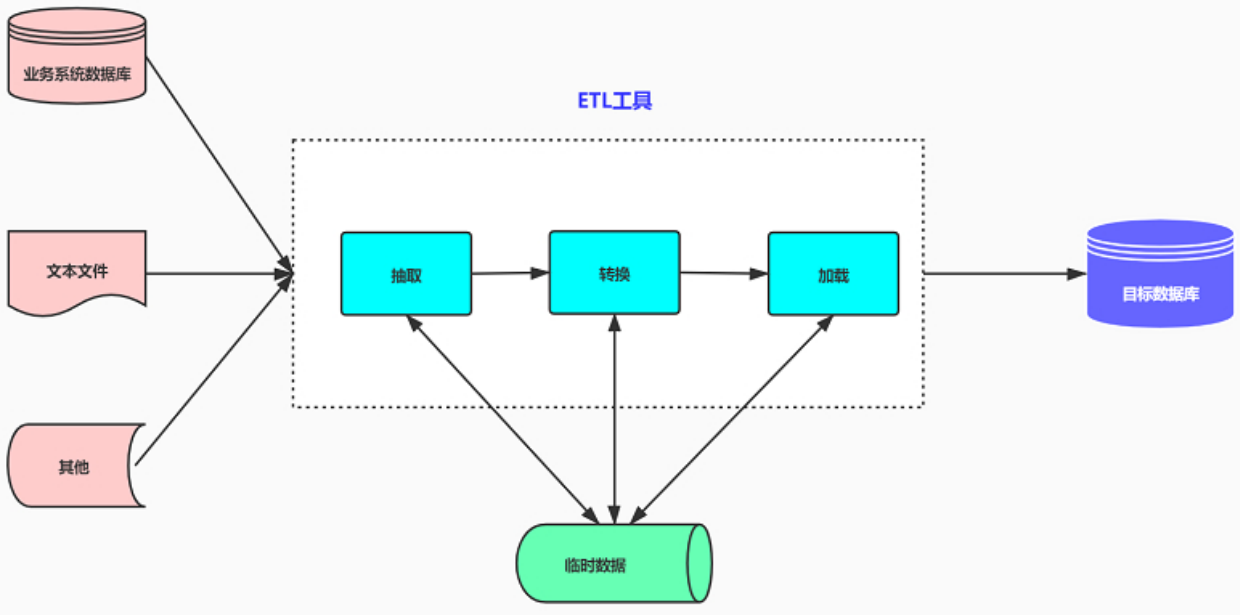
- **ETL定义**



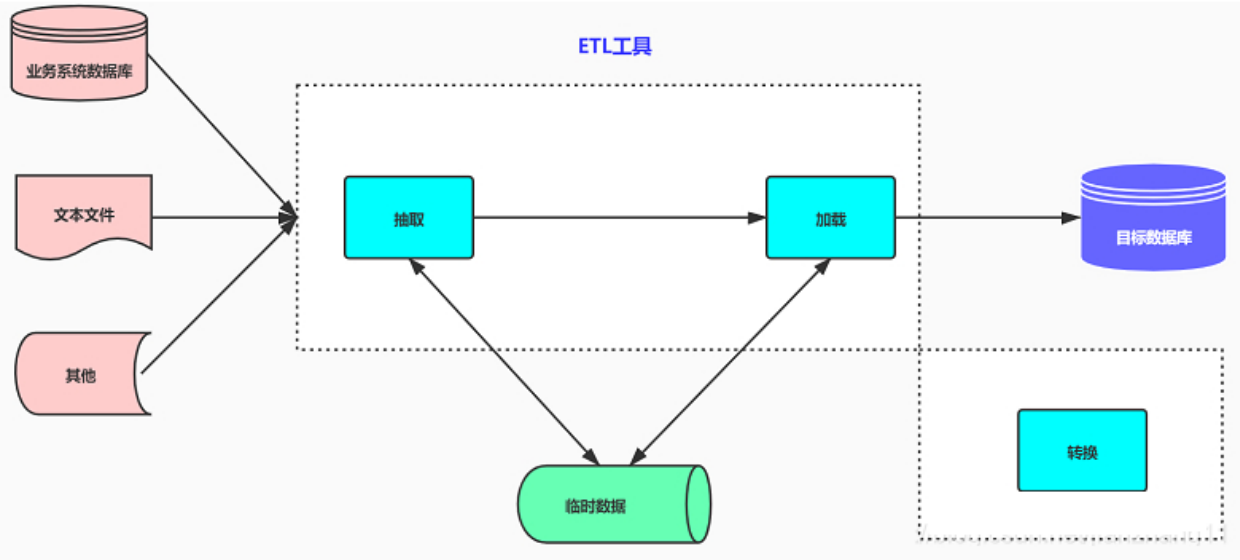

  - ETL是将数据从来源端经过抽取(Extract)、转换(Transform)、加载(Load)至目的端的过程
- 一般常用于数据仓库领域,将数据进行采集并做转换处理,最终将转换好的数据加载至数据仓库中
- **ETL功能**
  - **抽取**/采集: extract
    - 将不同数据来源的数据进行抽取
    - 数据来源: RDBMS、文件系统、数据流端口
    - 抽取方式: 全量、增量
  - **过滤/处理** : Tranform
    - 对抽取到的数据进行转换处理,数据清洗
    - **过滤**: 将非法的数据并行剔除过滤
    - **转换** : 将数据格式进行转换或者数据类型进行转换
      - 日期转换
      - 数据: 18/Aug/2021 12:30:30
      - 需要: 2021-08-18 12:30:30
    - **补全**: 利用已有的数据,将需要的但当前缺少的数据进行补全
      - 数据: IP地址
      - 需求: 得到用户的国家、省份、城市信息

- **加载** : Load
  - 将转换处理好的数据加载到目的地中
  - 加载方式:全量、增量

- ETL与ELT的区别
  - ELT:先加载再转换,适用于数据业务比较单一的场景,直接处理加载后得到结果
  -




  - ELT:先加载再转换,适用于数据业务比较复杂的场景,加载后根据不同的业务需求进行处理再应用
  -




  - 大数据中的ETL
    - 传统数据平台的ETL: 传统数据业务比较单一,主要实现数据清洗的功能,将非法数据进行过滤处理等
    - 大数据系统中的ETL: 大数据业务逻辑比较复杂,所有与数仓相关的计算处理过程统称为ETL
    - 数据仓库工程师 = ETL 工程师

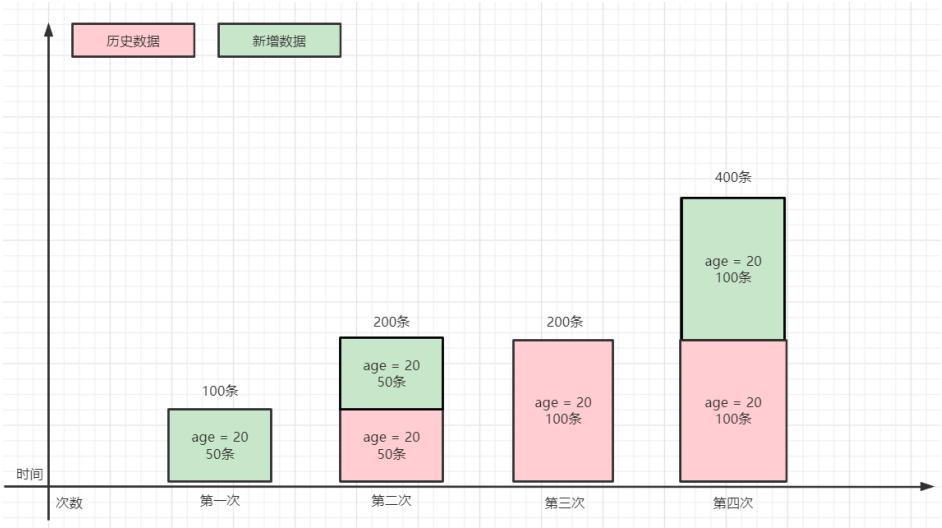
## 3.[了解]数仓ETL工具

- 招聘信息出现以下软件 : 不偏大数据,偏传统数据库

- 传统ETL工具
  - 特点 : 所有功能集于一身
  - Datastage:IBM公司的商业软件,最专业的ETL工具,但专业的ETL工具,但同时价格不菲,适合大规模的ETL应用
  - informatica: 同为商业的专业ETL工具,类似于Datastage,也适合大规模的ETL应用
  - kettle:业界最有名的开源ETL工具,用纯Java编写的ETL工具,只需要JVM环境即可部署,可跨平台,扩展性好
- 大数据平台的ETL工具
  - 特点:不同的功能由不同的工具来实现
  - 工具
    - 抽取/采集: Sqoop 、 Flume 、 Logstash 、 Beats 、 Canal
    - 转换/处理 : HiveSQL 、 SparkSQL 、 Impala 、 Presto 、 Kylin
    - 调度: Oozie 、 AirFlow 、 DS 、 Azkaban

## 4.[掌握]sqoop的功能及应用(抽取工具)

- **功能**: 用于实现MySQL等RDBMS数据库与HDFS[Hive/Hbase] 之间的数据导入与导出
-



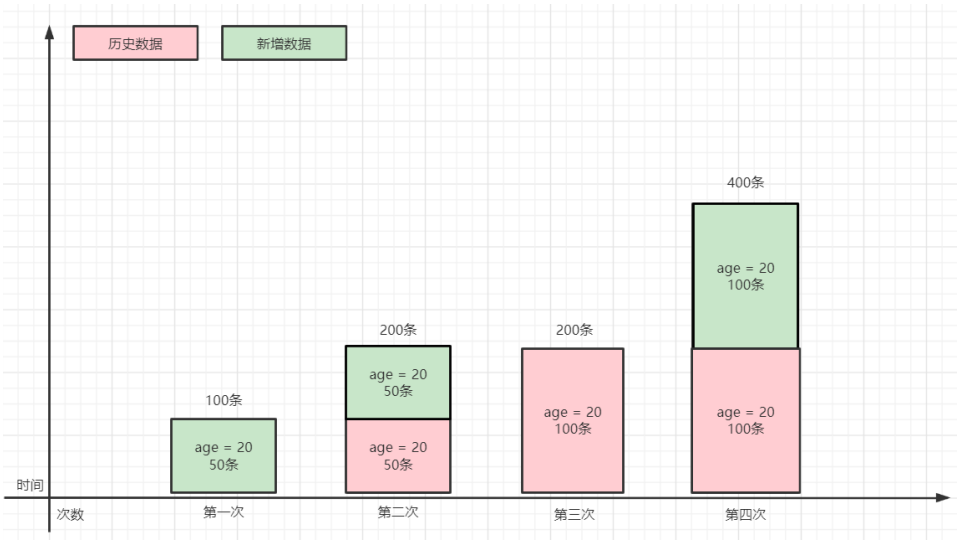

  - **导入**: 从MySQL导入到HDFS
  - **导出**: 从HDFS导出到MySQL
  - **抽取**: 将A的数据同步到B里面
    - A : 读取A数据(MySQL或HDFS)
    - B : 写入B数据(MySQL或HDFS)
- **本质**
  - **底层就是mapreduce程序**
    - **input**: 负责读取数据,由输入类决定 : TextinputFormat
      1. **功能一** : 分片 ,按照128M的1.1倍来分片(如果块130M,超过1.1倍按128M算,如果没有超过1.1倍则全部读取 ,即130M) -- 不适合处理小数据所有 [源码: getsplits]
      2. **功能二** : 将每个分片的数据转换成KV结构 ,K1V1[源码 : lineRecordReader.nexKeyValue]
    - **Map** : 按照分片个数来启动MapTask进程 , 每个MapTask调用map方法对自己负责的分片的数据进行处

理

- 处理逻辑由map方法来决定 ,输出K2,V2

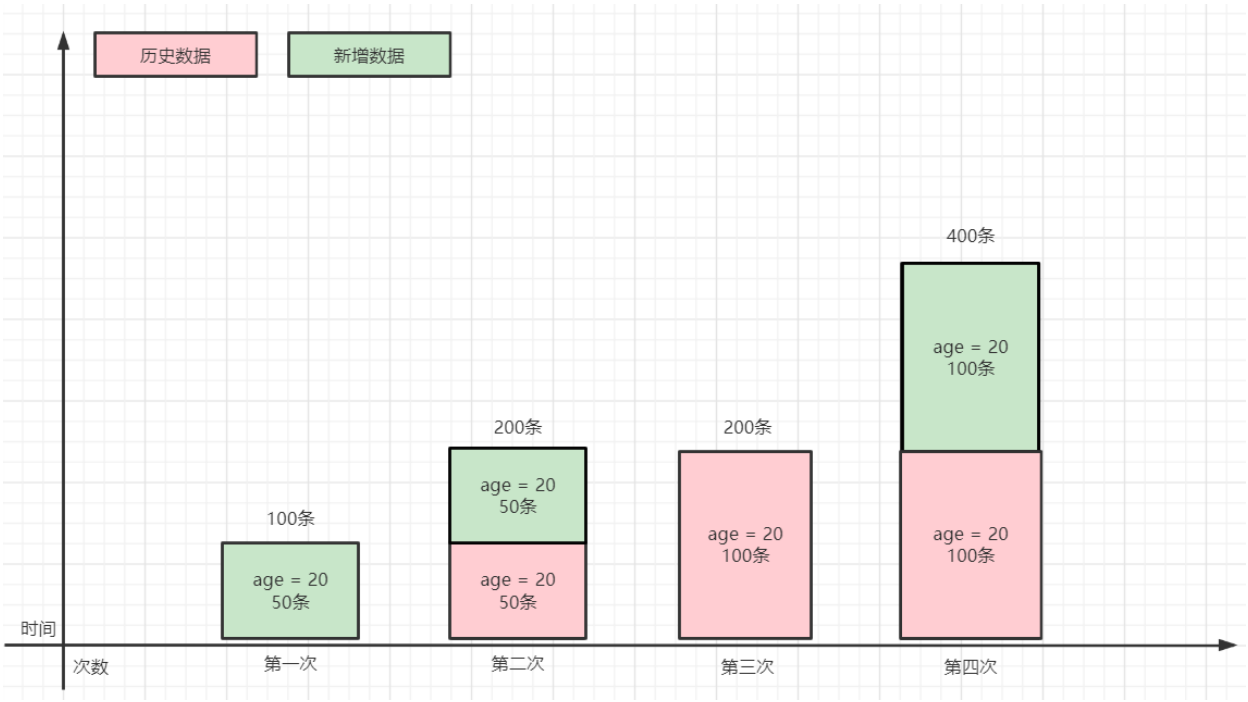
- **Shuffle** : 分区、 排序、分组
  - K2 List<V2...>
- **Reduce** :默认启动一个ReduceTask今夕来对Shuffle输出的数据进行处理,调用reduce方法
  - 处理逻辑由reduce方法来决定,输出K3 ,V3
- **Output** :负责保存结果,由输出类:TextOutputFormat
  - 将K3,V3保存到文件中,并且用制表符分隔
  - 要求:输出目录不能提前存在

- **将sqoop的程序转换成mapreduce程序,提交个yarn运行,实现分布式采集**
-




- **sqoop程序转换成的MR程序中一般没有Shuffle 和Reduce**
- **导入原理**
  - input : DBInputFormat 类(MySQL数据库)
  - Map:读写
  - Output:TextOutput
- **特点**
  - **必须依赖与Hadoop: MapReduce+ YARN**
  - **MapReduce是离线计算框架,**Sqoop离线数据采集工具,只能适合于离线业务平台
- **应用**
  - 数据同步:定期将离线的数据进行采集同步到数据仓库中
  - 数据迁移: 将历史数据[MySQL、Oracle]存储到HDFS中
- **Sqoop应用场景**
  - **功能**: 实现将RDBMS数据与HDFS之间实现导入和导出
  - **原理**: 底层是MR程序
  - **场景**: 离线数据库的数据同步和数据迁移

## 5.[掌握]数据全量、增量、条件概念




- **全量数据**: 从开始到目前开始为止所有数据的集合为全量数据集
  - 第一次: 100条
  - 第二次: 200条
  - 第三天: 200条
  - 第四次: 400条
  - 特点: 每次覆盖
- **增量数据** : 从上一次之后到这一次开始之间数据集合为增量数据集
  - 第一次:100条
  - 第二次:100条
  - 第三次:0条
  - 第四次:200条
  - 特点:每次追加
- **条件数据**:符合某种条件的数据集合为条件数据集
  - 覆盖/追加: 对每次处理的数据中进行条件过滤

## 6.[掌握]sqoop的开发规则

- **版本**
  - sqoop1版本:单纯的工具,直接实现程序的转换的
    - 没有服务端进程,不需要启动进程
  - sqoop2版本: 基于1.x的功能之上,引入cs模式[**如果用的MySQL版本比较高,建议使用sqoop2**] -- 可以识别中文列名

```
crate table tbname(    name string)comment '表的中文注释'
```

- 命令: ${SQOOP_HOMW}/bin/sqoop1
- 语法: sqoop help

```
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  import-mainframe   Import datasets from a mainframe server to HDFS
  job                Work with saved jobs
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  merge              Merge results of incremental imports
  metastore          Run a standalone Sqoop metastore
  version            Display version information

See 'sqoop help COMMAND' for information on a specific command.
```

### 6.1sqoop导入HDFS

```
#MySQL选项
--connect: 指定连接的RDBMS数据库的地址
--username: 指定数据库的用户名
--password: 指定数据库的密码
--table: 指定数据库中的表
--columns: 指定导入的列名
--where: 指定导入行的条件
-e/--query: 指定导入数据的SQL语句，不能与--table一起使用，必须指定where条件，where中必须指定$CONDITIONS

#HDFS选项
--target-dir: 指定导入的HDFS路径
```

--delete-target-dir: 指定如果导入的HDFS路径已存在就提前删除
--fields-terminated-by: 指定导入HDFS的文件中列的分隔符

#其他选项
-m: 指定底层MapReduce程序的MapTask的个数
--split-by: MySQL没有主键时，指定多个MapTask划分数据的方式
sqoop import ’org.apache.sqoop.splitter.allow_text_splitter=true' D表示修改sqoop属性
Dorg.apache.sqoop.splitter.allow_text_splitter=true：如果--split-by指定的是文本类型，需要开启该参数
sqoop-import 并行抽数及数据倾斜解决
通过num-mappers 参数控制并行度，split-by参数控制数据分割字段，就可以做到抽数并行化。
--num-mappers：启动N个map来并行导入数据，默认4个；
--split-by：按照某一列来切分表的工作单元。
根据不同的参数类型split-by有不同的切分方法，如比较简单的int型，sqoop会取最大和最小split-by字段值，然后根据传入的num-mappers来确定划分几个区域。

使用4个Map任务，过程中有2个任务失败这是导致不一样了
#sqoop会先把数据导入进这张临时表中，所有任务都成功了。才会导入
注意： -direct导入时staging方式是不可用的，使用了—update-key选项时staging方式也不能用。
--staging-table mysql表名_tmp
注意：临时表需要在MySQL中创建和导出的一样的表
#使用了--clear-staging-table选项,sqoop执行导出任务前会删除staging表中所有的数据。失败也会清空
--clear-staging-table

空值问题，hive底层用/N MySQL用null
1）导出数据时采用--input-null-string和--input-null-non-string两个参数。
--input-null-string ’\\N’
--input-null-non-string ’\\N’
2）导入数据时采用--null-string和--null-non-string。
--null-string ’\\N’
--null-non-string ’\\N’

#Hive选项
--hive-import: 指定导入数据到Hive表
--hive-overwrite ： 覆盖写入
#方式一
--hive-database: 指定原生方式导入Hive的数据库名称
--hive-table: 指定原生方式导入的Hive的表名
#方式二-推荐
--hcatalog-database: 指定使用hcatalog方式导入Hive的数据库名称
--hcatalog-table: 指定使用hcatalog方式导入Hive的数据库名称

--fields-terminated-by: 指定导入HDFS的文件中列的分隔符

导出数据库任务失败--数据一致性问题
多个Map任务时，采用–staging-table方式，仍然可以解决数据一致性问题。
1、指定mappers的数量（数量最好不要超过节点的个数）
sqoop job --exec gp1813_user - --num-mappers 8;
2、调整jvm的内存，缺点:
-Dmapreduce.map.memory.mb=6000
-Dmapreduce.map.java.opts=
-Xmx1600m-Dmapreduce.task.io.sort.mb=4800 \
3、设置mysql的读取数据的方式，不要一次性将所有数据都fetch到内存?
dontTrackOpenResources=true&defaultFetchSize=10000&useCursorFetch=true

5.如果导入的是分区表：
--hive-partition-key 'dt' \
--hive-partition-value '20200201' \

特殊问题

- 因oracle表特殊字段类型，导致sqoop导数据任务失败
- oracle字段类型为： clob或date等特殊类型
- 解决方案：在sqoop命令中添加参数，指定特殊类型字段列(SERIAL_NUM)的数据类型为string
  - `--map-column-java SERIAL_NUM=String`

| 参数 | 说明 |
| --- | --- |

| | |
|---|---|
| --connect | 连接关系型数据库的URL |
| --username | 连接数据库的用户名 |
| --password | 连接数据库的密码 |
| --driver | JDBC的driver class |
| --query或--e <statement> | 将查询结果的数据导入，使用时必须伴随参--target-dir，--hcatalog-table，如果查询中有where条件，则条件后必须加上CONDITIONS关键字。 如果使用双引号包含sql，则CONDITIONS前要加上\以完成转义：\$CONDITIONS |
| --hcatalog-database | 指定HCatalog表的数据库名称。如果未指定，default则使用默认数据库名称。提供 --hcatalog-database不带选项--hcatalog-table是错误的。 |
| --hcatalog-table | 此选项的参数值为HCatalog表名。该--hcatalog-table选项的存在表示导入或导出作业是使用HCatalog表完成的，并且是HCatalog作业的必需选项。 |
| --create-hcatalog-table | 此选项指定在导入数据时是否应自动创建HCatalog表。表名将与转换为小写的数据库表名相同。 |
| --hcatalog-storage-stanza 'stored as orc tblproperties ("orc.compress"="SNAPPY")' \ | 建表时追加存储格式到建表语句中，tblproperties修改表的属性，这里设置orc的压缩格式为SNAPPY |
| -m | 指定并行处理的MapReduce任务数量。 -m不为1时，需要用split-by指定分片字段进行并行导入，尽量指定int型。 |
| --split-by id | 如果指定-split by, 必须使用$CONDITIONS关键字, 双引号的查询语句还要加\ |
| --hcatalog-partition-keys --hcatalog-partition-values | keys和values必须同时存在，相当于指定静态分区。允许将多个键和值提供为静态分区键。多个选项值之间用，（逗号）分隔。比如： --hcatalog-partition-keys year,month,day --hcatalog-partition-values 1999,12,31 |
| --null-string '\N' --null-non-string '\N' | 指定mysql数据为空值时用什么符号存储，null-string针对string类型的NULL值处理，--null-non-string针对非string类型的NULL值处理 |
| --hive-drop-import-delims | 设置无视字符串中的分割符（hcatalog默认开启） |
| | |

| | |
|---|---|
| --fields-terminated-by '\t' | 设置字段分隔符 |

- 注意:指定多个MapTask时,分为两种情况
  - 情况一: 如果MySQL表有主键,可以直接指定
  - 情况二: 如果MySQL没有主键,会报错,加上--split-by指定按照哪一列进行划分数据即可
- 注意:采集到Hive时,默认分隔符时\001, 如果Hive表不是默认分隔符,采集时一定要指定
- Hcatalog

```
Apache HCatalog是基于 Apache Hadoop 之上的数据表和存储管理服务
-- 提供一个共享的模式和数据类型的机制
-- 抽象出表,使用户不必关心他们的数据怎么存储,底层什么格式
-- 提供可操作性的跨数据处理工具,如Pig ,MapReduce, Streaming 和 Hive
```

  - 与原生方式的区别(--hive 和 --hcatalog)

| 区别 | 原生方式 | Hcatalog方式 |
|---|---|---|
| 数据格式 | 较少 | 支持多种特殊格式:orc/rcfile/squencefile/json等 |
| 导入方式 | 允许覆盖 | 不允许覆盖，只能追加 |
| 字段匹配 | 顺序匹配，字段名可以不相等 | 字段名匹配，名称必须相等 |

### 6.2sqoop全量导入

- 导入HDFS
  - 需求1:将MySQL中tb_tohdfs表的数据导入HDFS的/sqoop/import/test01目录中

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--target-dir /sqoop/import/test01
```

  - 需求2：将tb_tohdfs表的id和name导入HDFS的/sqoop/import/test01目录,指定Map个数为1个

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
```

```
--columns id,name \
--delete-target-dir \
--target-dir /sqoop/import/test01 \
-m 1
```

- 需求3 : 需求3：将tb_tohdfs表的id和name导入HDFS的/sqoop/import/test01目录,指定Map个数为2个

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--columns id,name \
--delete-target-dir \
--target-dir /sqoop/import/test01 \
-m 2
```

- 需求4：将tb_tohdfs表中的id >4的数据导入HDFS的/sqoop/import/test01目录中，并且用制表符分隔

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--where 'id > 4' \
--delete-target-dir \
--target-dir /sqoop/import/test01 \
--fields-terminated-by '\t' \
-m 1
```

- 需求5：将tb_tohdfs表中的id>4的数据中id和name两列导入/sqoop/import/test01目录中
- #方式一

```
#方式一
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--columns id,name \
--where id>4 \
--delete-target-dir \
```

```
--target-dir /sqoop/import/tast01 \
--fields-terminated-by '\t' \
-m 1

#方式二
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
-e 'select id,name from tb_tohdfs where id>4 and $CONDITIONS' \
--delete-target-dir \
--target-dir /sqoop/import/test01 \
--fields-terminated-by '\t' \
-m 1

```

### 6.3sqoop全量导入Hive

- 需求:将MySQL中tb_tohdfs同步到Hive的default数据库下的fromsqoop表中,fromsqoop表不存在
- 方式一:使用sqoop创建表结构,再将数据同步到Hive表【工作中不会选用】

```
#建表
sqoop create-hive-table \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--hive-database default \
--hive-table fromsqoop1
#同步
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--hive-import \
--hive-database default \
--hive-table fromsqoop01 \
-m 1
```

- 方式二 :自己建表,将数据采集到Hive表中【不常用】

```
#Hive中建表
use default ;
drop table if exists fromsqoop02;
create table fromsqoop02 (
    id int ,
    name string ,
    age int
) row format delimited fields terminated by '\t';
#同步
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--hive-import \
--hive-database default \
--hive-table fromsqoop02 \
--fields-terminated-by '\t' \
-m 1

```

- 注意：采集到Hive时，默认分隔符是\001，如果Hive表不是默认分隔符，采集时一定要指定
- 方式三:使用Hcatalog方式,将数据同步到Hive表【主要使用的方式】

```
#Hive中建表
use default ;
drop table if exists fromsqoop3;
create table fromsqoop3(
    id int ,
    name string ,
    age int ,
    )row format delimited fields terminated by '\t';

#同步
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
```

```
--hcatalog-database default \
--hcatalog-table fromsqoop3 \
--m 1
```

| 区别 | 原生方式【--hive】 | Hcatalog方式[--hcatalog] |
| --- | --- | --- |
| 数据格式 | 较少 | 支持多种特殊格式：orc/rcfile/squencefile/json等 |
| 导入方式 | 允许覆盖 | 不允许覆盖，只能追加 |
| 字段匹配 | 顺序匹配，字段名可以不相等 | 字段名匹配，名称必须相等 |

### 6.4sqoop增量导入

- 增量采集
  - 用某一列的值基于上一次的采集来做比较
  - 设计:每次记录这一次采集这一列的值,用于对某一列值进行判断,只要大于上一次的值就会被导入
- sqoop自带方式
  - --check-column: 以哪一列的值作为增量的基准
  - --last-value : 指定上一次这一列的值是什么
  - --incremental : 指定增量的方式 append 和 lastmodified
- append方式测试
  - 要求:必须有一列自增的值,按照自增的int值进行判断
  - 特点 : 只能导入增加的数据,无法导入更新的数据
  - 场景:数据只会新增,不会发生更新的场景
  - 测试
    - 第一次导入

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--target-dir /sqoop/import/test02 \
--fields-terminated-by '\t' \
--check-column id \
--incremental append \
```

```
--last-value 0 \
-m 1
```

- 第二次导入

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_tohdfs \
--target-dir /sqoop/import/test02 \
--fields-terminated-by '\t' \
--incremental append \
--check-cloumn id \
--last-value 8 \
-m 1
```

- lastmodified方式测试
  - 要求:必须包含动态时间变化这一列,按照数据变化的时间进行判断
  - 特点 : 既导入新增的数据也导入更新的数据
  - 测试
    - MySQL中创建测试数据

```
-- MySQL中创建测试表
create table tb_lastmode (
    id  int not null auto_increment ,
    word varchar(255) not null ,
    lastmode timestamp not null default ,
    current_timestamp on update current_timestamp ,
    primary key (id)
)charset=utf8;
insert into tb_lastmode values(null,'hadoop',null);
insert into tb_lastmode values(null,'spark',null);
insert into tb_lastmode values(null,'hbase',null);
```

- 第一次采集

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_lastmode \
```

```
--target-dir /sqoop/import/test03 \
--fields-terminated-by '\t' \
--incremental lastmodified \
--check-column lastmode \
--last-value '2022-01-01 00:00:00' \
-m 1
```

- 数据发生变化

```
insert into tb_lastmode values(null,'hive',null);
update tb_lastmode set word = 'sqoop' where id = 1;
```

- 第二次采集

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_lastmode \
--target-dir /sqoop/import/test03 \
--fields-terminated-by '\t' \
--merge-key id \
--incremental lastmodified \
--check-column lastmode \
--last-value '2022-01-06 20:26:10' \
-m 1
```

- --merge-key : 将相同的ID的数据进行合并

- 条件过滤方式测试
  - 由于append和lastmodified的场景有限,工作中会见到使用条件过滤的方式来实现增量采集
  - 数据内容
  - id name age …… createTime updateTime
    - createTime: 数据产生的时间
    - updateTime : 数据更新的时间
  - 实现采集:今天采集昨天的数据

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
-e 'select * from tb_tohdfs where substr(create_time,1,10) = 昨天的日期 or
substr(update_time,1,10)=昨天的日期 and $CONDITIONS' \
```

```
--target-dir /sqoop/import/昨天的日期目录 \
--fields-terminated-by '\t' \
-m 1
```

## 6.5sqoop全量导出

- HDFS选项
  - --export-dir : 指定导出的HDFS路径
  - --input-fields-terminated-by : 指定导出的HDFS文件的列的分隔符
- Hive选项
  - --hcatalog-database: 指定使用Hcatalog方式导入Hive的数据库名称
  - --hcatalog-table: 指定使用hcatalog法师导入Hive的数据库名称
- 准备数据
  - MySQL中建表

```
use sqoopTest;
create table tb_url (
id int not null ,
url varchar(200) not null
primary key (id)
)charset=utf8;
```

  - Linux上创建测试文件

```
mkdir -p /export/data
vim /export/data/url.txt
1       http://facebook.com/path/p1.php?query=1
2       http://www.baidu.com/news/index.jsp?uuid=frank
3       http://www.jd.com/index?source=baidu
```

  - Hive中建表

```
-- Hive中建表
use default;
create table tb_url(
id int,
url string
) row format delimited fields terminated by '\t';
-- 加载数据到Hive表
load data local inpath '/export/data/url.txt' into table tb_url;
```

- 测试案例

- 需求1:将HDFS的某个目录的数据导出到MySQL

```
sqoop export \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_url \
--export-dir /user/hive/warehouse/tb_url \
--input-fields-terminated-by '\t' \
-m 1
```

- 需求2:将Hive的某张表的数据导出到MySQL

```
sqoop import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_url \
--hcatalog-database default \
--hcatalog-table tb_url \
--input-fields-terminated-by '\t' \
-m 1
```

## 6.6sqoop增量导出

- 增量导出场景
  - 增量导出方式
    - updateonly: 只增量导出更新的数据,不能同步新增的数据
    - allowerinsert : 既导出更新的数据,也导出新增的数据
  - updateonly
    - 修改url.txt数据

```
1       http://www.itcast.com/path/p1.php?query=1
2       http://www.baidu.com/news/index.jsp?uuid=frank
3       http://www.jd.com/index?source=baidu
4       http://www.heima.com
```

    - 重新加载覆盖

```
load data local inpath '/export/data/url.txt' overwrite into table tb_url ;
```

    - 增量导出

```
sqoop export \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_url \
--export-dir /user/hive/warehouse/tb_url \
--input-fields-terminated-by '\t' \
--update-key id \
--update-mode updateonly \
-m 1
```

- allowerinsert
  - 修改url.txt

```
1       http://bigdata.itcast.com/path/p1.php?query=1
2       http://www.baidu.com/news/index.jsp?uuid=frank
3       http://www.jd.com/index?source=baidu
4       http://www.heima.com
```

  - 覆盖表中数据

```
load data local inpath '/export/data/url.txt' overwrite into table tb_url;
```

  - 增量导出

```
sqoop export \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password 123456 \
--table tb_url \
--export-dir /user/hive/warehouse/tb_url \
--input-fields-terminated-by '\t' \
--update-key id \
--update-mode allowinsert \
-m 1
```

### 6.7sqoop Job的使用

- 创建job

```
sqoop job --create job01 \
-- import \
--connect jdbc:mysql://hadoop:3306/sqoopTest \
```

```
--username root \
--password 123456 \
--table tb_tohdfs \
--target-dir /sqoop/import/tset04 \
--fields-terminated-by '\t' \
--incremental append \
--check-column id \
--last-value 8 \
-m 1
```

- 创建job，不会运行程序，只是在元数据中记录信息
  - 列举job

```
sqoop job --list
```

  - 查看job的信息

```
sqoop job --show jobName
```

  - 运行job

```
sqoop job --exec jobName
```

  - 删除job

```
sqoop job --delete jobName
```

- 6.8sqoop密码解决
  - 方式一:在sqoop的sqoop-site-xml中配置将密码存储在客户端中
  - 方式二: 将密码存储在文件中,通过文件的权限来管理密码

```
sqoop job --create job02 \
-- import \
--connect jdbc:mysql://hadoop01:3306/sqoopTest \
--username root \
--password-file file:///export/data/sqoop.passwd \
--table tb_tohdfs \
--target-dir /sqoop/import/test05 \
--fields-terminated-by '\t' \
--incremental append \
--check-column id \
--last-value 4 \
-m 1
```

- --password-file：密码文件的路径，默认读取的是HDFS文件，这个文件中只能有一行密码
- 注意: vim创建的密码有换行符,需要去掉

### 6.8sqoop脚本使用

- 1.将代码封装到一个文件中

```
vim /export/data/test.sqoop
```

```
import
--connect
jdbc:mysql://hadoop01:3306/sqoopTest
--username
root
--password-file
file:///export/data/sqoop.passwd
--table
tb_tohdfs
--target-dir
/sqoop/import/test05
--fields-terminated-by
'\t'
-m
1
```

  - 要求:一行只放一个参数

- 2.运行这个文件

```
sqoop --options-file /export/data/test.sqoop
```

### sell脚本说明

```
问题1: 如何在shell中 获取上一天的时间呢?

date +'%Y-%m-%d %H:%M:%S' : 表示 让日期按照特定格式输出
结果: 2021-02-26 09:23:05

date -d '-1 day' +'%Y-%m-%d %H:%M:%S': 表示 获取上一天的时间

其他操作:
```

```
    date -d '-1 hour' +'%Y-%m-%d %H:%M:%S' : 获取上一个小时
    date -d '-1 week' +'%Y-%m-%d %H:%M:%S' : 获取上一周时间

wait 挂起
前面的没有执行完可以先挂起，执行下面的语句
```

增量shell脚本

```
#!/bin/bash

export SQOOP_HOME=/usr/bin/sqoop
#如果参数个数=1
if [ $# -eq 1 ]
then
    #如果传入了参数，则赋值

        time_str=$1
    yearAndMonth=`date -d $1 +'%Y_%m'`

else
    time_str=`date -d '-1 day' +'%Y-%m-%d'`
    yearAndMonth=`date -d '-1 day' +'%Y_%m'`

fi

jdbcUrl='jdbc:mysql://192.168.52.150:3306/nev'
username='root'
password='123456'

#sqoop脚本
wait

```

增量2shell脚本

```
#!/bin/bash

#step1：先获取要采集的数据时间，规则：如果没有给参数，就默认处理昨天的日期，如果给了参数，就参数对应的日期
if [ $# -ne 0 ]
then
        #参数个数不为0
```

```
        if [ $# -ne 1 ]
        then
                echo "参数至多只能有一个，为处理的日期，请重新运行！"
                exit 100
        else
                #参数个数只有1个，就用第一个参数作为处理的日期
                yesterday=$1
        fi
else
        #参数个数为0，默认处理昨天的日期
        yesterday=`date -d '-1 day' +%Y-%m-%d`
fi
echo "step1：要处理的日期是：${yesterday}"

echo "step2：开始运行分析"
#step2：运行分析程序
hive --hiveconf yest=${yesterday}  -f hdfs://hadoop01:8020/user/oozie/shell/04.export.sql

echo "step2：分析的程序运行结束"

echo "step3：开始运行导出的程序"
#step2：运行增量导出
sqoop export \
--connect jdbc:mysql://hadoop01:3306/db_order \
--username root \
--password-file hdfs://hadoop01:8020/user/oozie/shell/sqoop.passwd \
--table tb_order_rs \
--hcatalog-database default \
--hcatalog-table tb_order_rs \
--input-fields-terminated-by '\t' \
--update-key daystr \
--update-mode allowinsert \
-m 1

echo "step3：导出的程序运行结束"


```

如何保证sqoop数据同步后2边数据量一致

```
#1-sqoop导入数据
/export/server/sqoop/bin/sqoop import \
--connect jdbc:mysql://node1:3306/insurance \
--username root \
--password 123456 \
--table dd_table \
--hive-table insurance_ods.dd_table \
--hive-import \
--hive-overwrite \
--fields-terminated-by '\t' \
--delete-target-dir \
-m 1

#2-统计mysql表dd_table的条数
mysql_log=`/export/server/sqoop/bin/sqoop eval \
--connect jdbc:mysql://node1:3306/insurance \
--username root \
--password 123456 \
--query "select count(*) from dd_table"`
# | 管道，将左边的结果输出 给到 右边作为输入，  awk -F表示按什么来切分，'{print $4}'表示返回切完后的第4个值
mysql_cnt=`echo $mysql_log | awk -F '|' '{print $4}' | awk -F ' ' '{print $1}'`

#3-统计hive表dd_table的条数
hive_log=`spark-sql -e 'select count(*) from insurance_ods.dd_table'`
hive_cnt=`echo $hive_log | awk -F ' ' '{print $2}'`
#4-比较2边的条数是否一致。
if [ $mysql_cnt -eq $hive_cnt ] ; then
  echo "mysql表的条数是$mysql_cnt,hive表的条数是$hive_cnt，2边一致"
else
  echo "mysql表的条数是$mysql_cnt,hive表的条数是$hive_cnt，2边不一致"
fi
```