尚硅谷大数据技术之 DataX

(作者：尚硅谷大数据研究院)

版本：V1.0

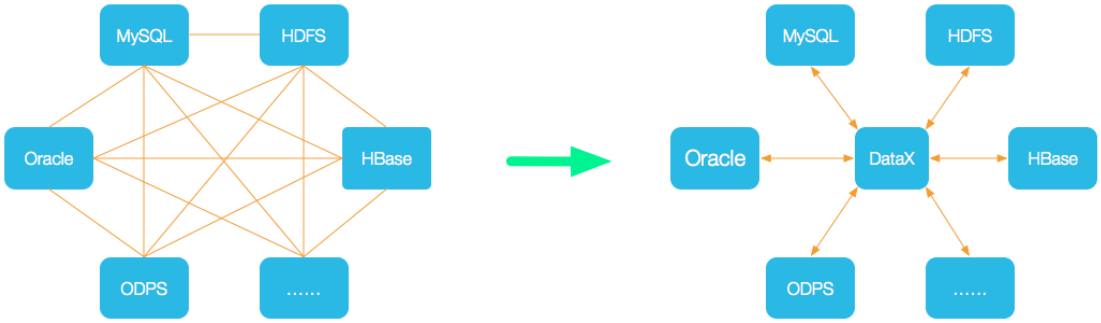
第1章 概述

1.1 什么是 DataX

DataX 是阿里巴巴开源的一个异构数据源离线同步工具，致力于实现包括关系型数据库(MySQL、Oracle 等)、HDFS、Hive、ODPS、HBase、FTP 等各种异构数据源之间稳定高效的数据同步功能。

1.2 DataX 的设计

为了解决异构数据源同步问题，DataX 将复杂的网状的同步链路变成了星型数据链路，DataX 作为中间传输载体负责连接各种数据源。当需要接入一个新的数据源的时候，只需要将此数据源对接到 DataX，便能跟已有的数据源做到无缝数据同步。



1.3 支持的数据源

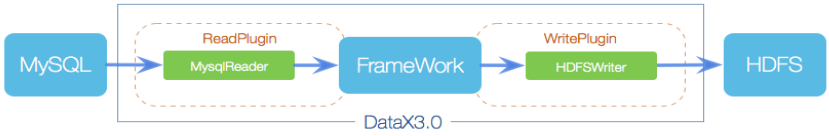
DataX 目前已经有了比较全面的插件体系，主流的 RDBMS 数据库、NOSQL、大数据

计算系统都已经接入。

类型	数据源	Reader(读)	Writer(写)	文档
RDBMS 关系型数据库	MySQL	√	√	读、写
	Oracle	√	√	读、写
	SQLServer	√	√	读、写
	PostgreSQL	√	√	读、写
	DRDS	√	√	读、写
	通用RDBMS(支持所有关系型数据库)	√	√	读、写
阿里云数仓数据存储	ODPS	√	√	读、写
	ADS		√	写
	OSS	√	√	读、写
	OCS	√	√	读、写
NoSQL数据存储	OTS	√	√	读、写
	Hbase0.94	√	√	读、写
	Hbase1.1	√	√	读、写
	Phoenix4.x	√	√	读、写
	Phoenix5.x	√	√	读、写
	MongoDB	√	√	读、写
	Hive	√	√	读、写
	Cassandra	√	√	读、写
无结构化数据存储	TxtFile	√	√	读、写
	FTP	√	√	读、写
	HDFS	√	√	读、写
	Elasticsearch		√	写
时间序列数据库	OpenTSDB	√		读
	TSDB	√	√	读、写

1.4 框架设计

DataX框架

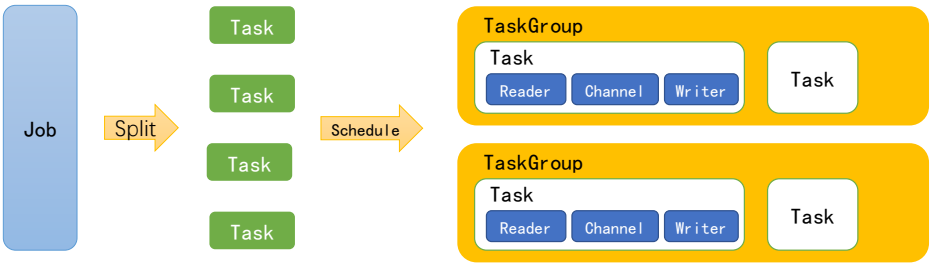


Reader：数据采集模块，负责采集数据源的数据，将数据发送给Framework。

Writer：数据写入模块，负责不断向Framework取数据，并将数据写入到目的端。

Framework：用于连接reader和writer，作为两者的数据传输通道，并处理缓冲，流控，并发，数据转换等核心技术问题。

1.5 运行原理



Job：单个作业的管理节点，负责数据清理、子任务划分、TaskGroup监控管理。

Task：由Job切分而来，是DataX作业的最小单元，每个Task负责一部分数据的同步工作。

Schedule：将Task组成TaskGroup，单个TaskGroup的并发数量为5。

TaskGroup：负责启动Task。

举例来说，用户提交了一个 DataX 作业，并且配置了 20 个并发，目的是将一个 100 张分表的 mysql 数据同步到 odps 里面。 DataX 的调度决策思路是：

1）DataXJob 根据分库分表切分成了 100 个 Task。

2）根据 20 个并发，DataX 计算共需要分配 4 个 TaskGroup。

3）4 个 TaskGroup 平分切分好的 100 个 Task，每一个 TaskGroup 负责以 5 个并发共计运行 25 个 Task。

1.6 与 Sqoop 的对比

功能	DataX	Sqoop
运行模式	单进程多线程	MR
MySQL 读写	单机压力大； 读写粒度容易控制	MR 模式重，写出错处理麻烦
Hive 读写	单机压力大	很好
文件格式	orc 支持	orc 不支持，可添加
分布式	不支持，可以通过调度系统规避	支持
流控	有流控功能	需要定制
统计信息	已有一些统计，上报需定制	没有，分布式的数据收集不方便
数据校验	在 core 部分有校验功能	没有，分布式的数据收集不方便
监控	需要定制	需要定制
社区	开源不久，社区不活跃	一直活跃，核心部分变动很少

第2章 快速入门

2.1 官方地址

下载地址：http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz

源码地址：https://github.com/alibaba/DataX

2.2 前置要求

- Linux
- JDK(1.8 以上，推荐 1.8)
- Python(推荐 Python2.6.X)

2.3 安装

1）将下载好的 datax.tar.gz 上传到 hadoop102 的/opt/software

2）解压 datax.tar.gz 到/opt/module

```
[atguigu@hadoop102 software]$ tar -zxvf datax.tar.gz -C /opt/module/
```

3）运行自检脚本

```
[atguigu@hadoop102 bin]$ cd /opt/module/datax/bin/
[atguigu@hadoop102 bin]$ python datax.py /opt/module/datax/job/job.json
```

```
2019-05-08 22:08:18.271 [main] INFO  JobContainer - Set jobId = 0
2019-05-08 22:08:18.291 [job-0] INFO  JobContainer - jobContainer starts to do prepare ...
2019-05-08 22:08:18.292 [job-0] INFO  JobContainer - DataX Reader.Job [streamreader] do prepare work .
2019-05-08 22:08:18.292 [job-0] INFO  JobContainer - DataX Writer.Job [streamwriter] do prepare work .
2019-05-08 22:08:18.292 [job-0] INFO  JobContainer - jobContainer starts to do split ...
2019-05-08 22:08:18.293 [job-0] INFO  JobContainer - Job set Max-Byte-Speed to 10485760 bytes.
2019-05-08 22:08:18.294 [job-0] INFO  JobContainer - DataX Reader.Job [streamreader] splits to [1] tasks.
2019-05-08 22:08:18.295 [job-0] INFO  JobContainer - DataX Writer.Job [streamwriter] splits to [1] tasks.
2019-05-08 22:08:18.322 [job-0] INFO  JobContainer - jobContainer starts to do schedule ...
2019-05-08 22:08:18.327 [job-0] INFO  JobContainer - Scheduler starts [1] taskGroups.
2019-05-08 22:08:18.329 [job-0] INFO  JobContainer - Running by standalone Mode.
2019-05-08 22:08:18.344 [taskGroup-0] INFO  TaskGroupContainer - taskGroupId=[0] start [1] channels for [1] tasks.
2019-05-08 22:08:18.347 [taskGroup-0] INFO  Channel - Channel set byte_speed_limit to -1, No bps activated.
2019-05-08 22:08:18.348 [taskGroup-0] INFO  Channel - Channel set record_speed_limit to -1, No tps activated.
2019-05-08 22:08:18.361 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] taskId[0] attemptCount[1] is started
2019-05-08 22:08:18.665 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] taskId[0] is successed, used[306]ms
2019-05-08 22:08:18.666 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] completed it's tasks.
2019-05-08 22:08:28.357 [job-0] INFO  StandAloneJobContainerCommunicator - Total 100000 records, 2600000 bytes | Speed 253.91KB/s, 10000 records/s | Error 0 records, 0 bytes |  All Task
WaitWriterTime 0.021s |  All Task WaitReaderTime 0.043s | Percentage 100.00%
2019-05-08 22:08:28.357 [job-0] INFO  AbstractScheduler - Scheduler accomplished all tasks.
2019-05-08 22:08:28.358 [job-0] INFO  JobContainer - DataX Writer.Job [streamwriter] do post work.
2019-05-08 22:08:28.358 [job-0] INFO  JobContainer - DataX Reader.Job [streamreader] do post work.
2019-05-08 22:08:28.359 [job-0] INFO  JobContainer - DataX jobId [0] completed successfully.
2019-05-08 22:08:28.359 [job-0] INFO  HookInvoker - No hook invoked, because base dir not exists or is a file: /opt/module/datax/hook
2019-05-08 22:08:28.362 [job-0] INFO  JobContainer -
         [total cpu info] =>
                averageCpu                     | maxDeltaCpu                    | minDeltaCpu
                -1.00%                         | -1.00%                         | -1.00%


         [total gc info] =>
                 NAME                 | totalGCCount       | maxDeltaGCCount    | minDeltaGCCount    | totalGCTime        | maxDeltaGCTime     | minDeltaGCTime
                 PS MarkSweep         | 0                  | 0                  | 0                  | 0.000s             | 0.000s             | 0.000s
                 PS Scavenge          | 0                  | 0                  | 0                  | 0.000s             | 0.000s             | 0.000s

2019-05-08 22:08:28.362 [job-0] INFO  JobContainer - PerfTrace not enable!
2019-05-08 22:08:28.363 [job-0] INFO  StandAloneJobContainerCommunicator - Total 100000 records, 2600000 bytes | Speed 253.91KB/s, 10000 records/s | Error 0 records, 0 bytes |  All Task
WaitWriterTime 0.021s |  All Task WaitReaderTime 0.043s | Percentage 100.00%
2019-05-08 22:08:28.365 [job-0] INFO  JobContainer -
任务启动时刻                    : 2019-05-08 22:08:18
任务结束时刻                    : 2019-05-08 22:08:28
任务总计耗时                    :                 10s
任务平均流量                    :          253.91KB/s
记录写入速度                    :          10000rec/s
读出记录总数                    :              100000
读写失败总数                    :                   0
```

第3章 使用案例

3.1 从 stream 流读取数据并打印到控制台

1）查看配置模板

```
[atguigu@hadoop102 bin]$ python datax.py -r streamreader -w streamwriter
DataX (DATAX-OPENSOURCE-3.0), From Alibaba !
Copyright (C) 2010-2017, Alibaba Group. All Rights Reserved.


Please refer to the streamreader document:
     https://github.com/alibaba/DataX/blob/master/streamreader/doc/streamreader.md

Please refer to the streamwriter document:
     https://github.com/alibaba/DataX/blob/master/streamwriter/doc/streamwriter.md

Please save the following configuration as a json file and  use
     python {DATAX_HOME}/bin/datax.py {JSON_FILE_NAME}.json
to run the job.

{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "streamreader",
                    "parameter": {
                        "column": [],
                        "sliceRecordCount": ""
                    }
                },
                "writer": {
                    "name": "streamwriter",
                    "parameter": {
                        "encoding": "",
                        "print": true
                    }
```

```
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": ""
            }
        }
    }
}
```

2）根据模板编写配置文件

```
[atguigu@hadoop102 job]$ vim stream2stream.json
```

填写以下内容：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
                "value": "hello，DataX"
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "encoding": "UTF-8",
            "print": true
```

```
            }
        }
    }
],
"setting": {
    "speed": {
        "channel": 1
    }
}
}
}
```

3）运行

```
[atguigu@hadoop102 job]$
/opt/module/datax/bin/datax.py /opt/module/datax/job/stream2stream.json
```

3.2 读取 MySQL 中的数据存放到 HDFS

3.2.1 查看官方模板

```
[atguigu@hadoop102 ~]$ python /opt/module/datax/bin/datax.py -r mysqlreader -w
hdfswriter
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "column": [],
                        "connection": [
                            {
                                "jdbcUrl": [],
                                "table": []
                            }
                        ],
                        "password": "",
                        "username": "",
                        "where": ""
```

```
                        }
                    },
                    "writer": {
                        "name": "hdfswriter",
                        "parameter": {
                            "column": [],
                            "compress": "",
                            "defaultFS": "",
                            "fieldDelimiter": "",
                            "fileName": "",
                            "fileType": "",
                            "path": "",
                            "writeMode": ""
                        }
                    }
                }
            ],
            "setting": {
                "speed": {
                    "channel": ""
                }
            }
        }
}
```

mysqlreader 参数解析：

mysqlreader参数解析

尚硅谷

```
"reader": {
            "name": "mysqlreader",
            "parameter": {
              "column": [],
              "connection": [
                {
                  "jdbcUrl": [],
                  "table": [ ]
                  【"querySql:[]】
                }
              ],
              "password": "",
              "username": "",
              【"where": ""】
              【"splitPk": ""】
            }
          }
```

name：reader名

column：需要同步的列名集合，使用JSON数组描述自带信息，*代表所有列

jdbcUrl：对数据库的JDBC连接信息，使用JSON数组描述，支持多个连接地址

table：需要同步的表，支持多个

querySql：自定义SQL，配置它后，mysqlreader直接忽略table、column、where

password：数据库用户名对应的密码

username：数据库用户名

where：筛选条件

splitPK：数据分片字段，一般是主键，仅支持整型

注意：【】中的参数为可选参数

hdfswriter 参数解析：


```
"writer": {
            "name": "hdfswriter",
            "parameter": {
               "column": [],
               "compress": "",
               "defaultFS": "",
               "fieldDelimiter": "",
               "fileName": "",
               "fileType": "",
               "path": "",
               "writeMode": ""
            }
         }
```

name：writer名

column：写入数据的字段，其中name指定字段名，type指定类型

compress：hdfs文件压缩类型，默认不填写意味着没有压缩。

defaultFS：hdfs文件系统namenode节点地址，格式：hdfs://ip:端口

fieldDelimiter：字段分隔符

fileName：写入文件名

fileType：文件的类型，目前只支持用户配置为"text"或"orc"

path：存储到Hadoop hdfs文件系统的路径信息

writeMode：hdfswriter写入前数据清理处理模式：
(1)append：写入前不做任何处理，DataX hdfswriter直接使用filename写入，并保证文件名不冲突。
(2)nonConflict：如果目录下有fileName前缀的文件，直接报错。

3.2.2 准备数据

1）创建 student 表

```
mysql> create database datax;
mysql> use datax;
mysql> create table student(id int,name varchar(20));
```

2）插入数据

```
mysql> insert into student values(1001,'zhangsan'),(1002,'lisi'),(1003,'wangwu');
```

3.2.3 编写配置文件

```
[atguigu@hadoop102 datax]$ vim /opt/module/datax/job/mysql2hdfs.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "column": [
                            "id",
                            "name"
                        ],
                        "connection": [
```

```
                        {
                            "jdbcUrl": [
                                "jdbc:mysql://hadoop102:3306/datax"
                            ],
                            "table": [
                                "student"
                            ]
                        }
                    ],
                    "username": "root",
                    "password": "000000"
                }
            },
            "writer": {
                "name": "hdfswriter",
                "parameter": {
                    "column": [
                        {
                            "name": "id",
                            "type": "int"
                        },
                        {
                            "name": "name",
                            "type": "string"
                        }
                    ],
                    "defaultFS": "hdfs://hadoop102:9000",
                    "fieldDelimiter": "\t",
                    "fileName": "student.txt",
                    "fileType": "text",
                    "path": "/",
                    "writeMode": "append"
                }
            }
        }
    ],
    "setting": {
        "speed": {
```

```
                        "channel": "1"
                    }
                }
            }
        }
}
```

3.2.4 执行任务

```
[atguigu@hadoop102 datax]$ bin/datax.py job/mysql2hdfs.json
2019-05-17 16:02:16.581 [job-0] INFO    JobContainer -
任务启动时刻                    : 2019-05-17 16:02:04
任务结束时刻                    : 2019-05-17 16:02:16
任务总计耗时                    :                 12s
任务平均流量                    :               3B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   3
读写失败总数                    :                   0
```

3.2.5 查看 hdfs

Browse Directory

/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxrwx	atguigu	supergroup	0 B	2019/5/7 下午5:12:10	0	0 B	flume
drwxr-xr-x	atguigu	supergroup	0 B	2019/5/4 下午5:46:36	0	0 B	hbase
-rw-r--r--	atguigu	supergroup	230 B	2019/5/4 下午6:43:30	3	128 MB	hive.txt
-rw-r--r--	wei	supergroup	83 B	2019/5/4 下午5:43:33	3	128 MB	out.txt
-rw-r--r--	atguigu	supergroup	36 B	2019/5/17 下午4:02:07	3	128 MB	student.txt__5961743b_8440_4bdc_8d41_e2ec06f4544a
drwxrwx---	atguigu	supergroup	0 B	2019/4/26 下午2:45:09	0	0 B	tmp
drwxr-xr-x	atguigu	supergroup	0 B	2019/4/26 下午2:43:55	0	0 B	user

注意：HdfsWriter 实际执行时会在该文件名后添加随机的后缀作为每个线程写入实际文件名。

3.2.6 关于 HA 的支持

```
"hadoopConfig":{
    "dfs.nameservices": "ns",
    "dfs.ha.namenodes.ns": "nn1,nn2",
    "dfs.namenode.rpc-address.ns.nn1": "主机名:端口",
    "dfs.namenode.rpc-address.ns.nn2": "主机名:端口",
```

```
    "dfs.client.failover.proxy.provider.ns":
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
}
```

3.3 读取 HDFS 数据写入 MySQL

1）将上个案例上传的文件改名

```
[atguigu@hadoop102 datax]$ hadoop fs -mv /student.txt* /student.txt
```

2）查看官方模板

```
[atguigu@hadoop102 datax]$ python bin/datax.py -r hdfsreader -w mysqlwriter
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "hdfsreader",
                    "parameter": {
                        "column": [],
                        "defaultFS": "",
                        "encoding": "UTF-8",
                        "fieldDelimiter": ",",
                        "fileType": "orc",
                        "path": ""
                    }
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": [],
                        "connection": [
                            {
                                "jdbcUrl": "",
                                "table": []
                            }
                        ],
                        "password": "",
```

```
                                "preSql": [],
                                "session": [],
                                "username": "",
                                "writeMode": ""
                            }
                        }
                    }
                ],
                "setting": {
                    "speed": {
                        "channel": ""
                    }
                }
            }
}
```

3）创建配置文件

```
[atguigu@hadoop102 datax]$ vim job/hdfs2mysql.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "hdfsreader",
                    "parameter": {
                        "column": ["*"],
                        "defaultFS": "hdfs://hadoop102:9000",
                        "encoding": "UTF-8",
                        "fieldDelimiter": "\t",
                        "fileType": "text",
                        "path": "/student.txt"
                    }
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": [
                            "id",
                            "name"
```

```
                                ],
                                "connection": [
                                    {
                                        "jdbcUrl": "jdbc:mysql://hadoop102:3306/datax",
                                        "table": ["student2"]
                                    }
                                ],
                                "password": "000000",
                                "username": "root",
                                "writeMode": "insert"
                            }
                        }
                    }
                ],
                "setting": {
                    "speed": {
                        "channel": "1"
                    }
                }
            }
}
```

4）在 MySQL 的 datax 数据库中创建 student2

```
mysql> use datax;
mysql> create table student2(id int,name varchar(20));
```

5）执行任务

```
[atguigu@hadoop102 datax]$ bin/datax.py job/hdfs2mysql.json
```

```
2019-05-17 16:21:53.616 [job-0] INFO    JobContainer -
任务启动时刻                    : 2019-05-17 16:21:41
任务结束时刻                    : 2019-05-17 16:21:53
任务总计耗时                    :                 11s
任务平均流量                    :               3B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                  3
读写失败总数                    :                  0
```

6）查看 student2 表

```
mysql> select * from student2;
+------+----------+
| id   | name     |
+------+----------+
| 1001 | zhangsan |
| 1002 | lisi     |
| 1003 | wangwu   |
+------+----------+
3 rows in set (0.00 sec)
```

第4章 Oracle 数据库

以下操作使用 root 账号。

4.1 oracle 数据库简介

Oracle Database，又名 Oracle RDBMS，或简称 Oracle。是甲骨文公司的一款关系数据库管理系统。它是在数据库领域一直处于领先地位的产品。可以说 Oracle 数据库系统是目前世界上流行的关系数据库管理系统，系统可移植性好、使用方便、功能强，适用于各类大、中、小、微机环境。它是一种高效率、可靠性好的、适应高吞吐量的数据库解决方案。

4.2 安装前的准备

4.2.1 安装依赖

```
yum install -y bc binutils compat-libcap1 compat-libstdc++33 elfutils-libelf elfutils-libelf-devel fontconfig-devel glibc glibc-devel ksh libaio libaio-devel libX11 libXau libXi libXtst libXrender libXrender-devel libgcc libstdc++ libstdc++-devel libxcb make smartmontools sysstat kmod* gcc-c++ compat-libstdc++-33
```

4.2.2 配置用户组

Oracle 安装文件不允许通过 root 用户启动，需要为 oracle 配置一个专门的用户。

1）创建 sql 用户组

```
[root@hadoop102 software]#groupadd sql
```

2）创建 oracle 用户并放入 sql 组中

```
[root@hadoop102 software]#useradd oracle -g sql
```

3）修改 oracle 用户登录密码，输入密码后即可使用 oracle 用户登录系统

```
[root@hadoop102 software]#passwd oracle
```

4.2.3 上传安装包并解压

注意:19c 需要把软件包直接解压到 ORACLE_HOME 的目录下

```
[root@hadoop102 software]# mkdir -p /home/oracle/app/oracle/product/19.3.0/dbhome_1
[root@hadoop102 software]# unzip LINUX.X64_193000_db_home.zip -d
/home/oracle/app/oracle/product/19.3.0/dbhome_1
```

修改所属用户和组

```
[root@hadoop102 dbhome_1]# chown -R oracle:sql /home/oracle/app/
```

4.2.4 修改配置文件 sysctl.conf

```
[root@hadoop102 module]# vim /etc/sysctl.conf
```

删除里面的内容，添加如下内容：

```
net.ipv4.ip_local_port_range = 9000 65500
fs.file-max = 6815744
kernel.shmall = 10523004
kernel.shmmax = 6465333657
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
net.core.rmem_default=262144
net.core.wmem_default=262144
net.core.rmem_max=4194304
net.core.wmem_max=1048576
fs.aio-max-nr = 1048576
```

参数解析：

net.ipv4.ip_local_port_range ： 可使用的 IPv4 端口范围

fs.file-max ： 该参数表示文件句柄的最大数量。文件句柄设置表示在 linux 系统中可以打开的文件数量。

kernel.shmall ： 该参数表示系统一次可以使用的共享内存总量（以页为单位）

kernel.shmmax ： 该参数定义了共享内存段的最大尺寸（以字节为单位）

kernel.shmmni ： 这个内核参数用于设置系统范围内共享内存段的最大数量

kernel.sem ： 该参数表示设置的信号量。

net.core.rmem_default：默认的 TCP 数据接收窗口大小（字节）。

net.core.wmem_default：默认的 TCP 数据发送窗口大小（字节）。

net.core.rmem_max：最大的 TCP 数据接收窗口（字节）。

net.core.wmem_max：最大的 TCP 数据发送窗口（字节）。

fs.aio-max-nr ：同时可以拥有的的异步 IO 请求数目。

4.2.5 修改配置文件 limits.conf

```
[root@hadoop102 module]# vim /etc/security/limits.conf
```

在文件末尾添加：

```
oracle soft nproc 2047
oracle hard nproc 16384
oracle soft nofile 1024
oracle hard nofile 65536
```

重启机器生效。

4.3 安装 Oracle 数据库

4.3.1 设置环境变量

```
[oracle@hadoop102 dbhome_1]# vim /home/oracle/.bash_profile
```

添加：

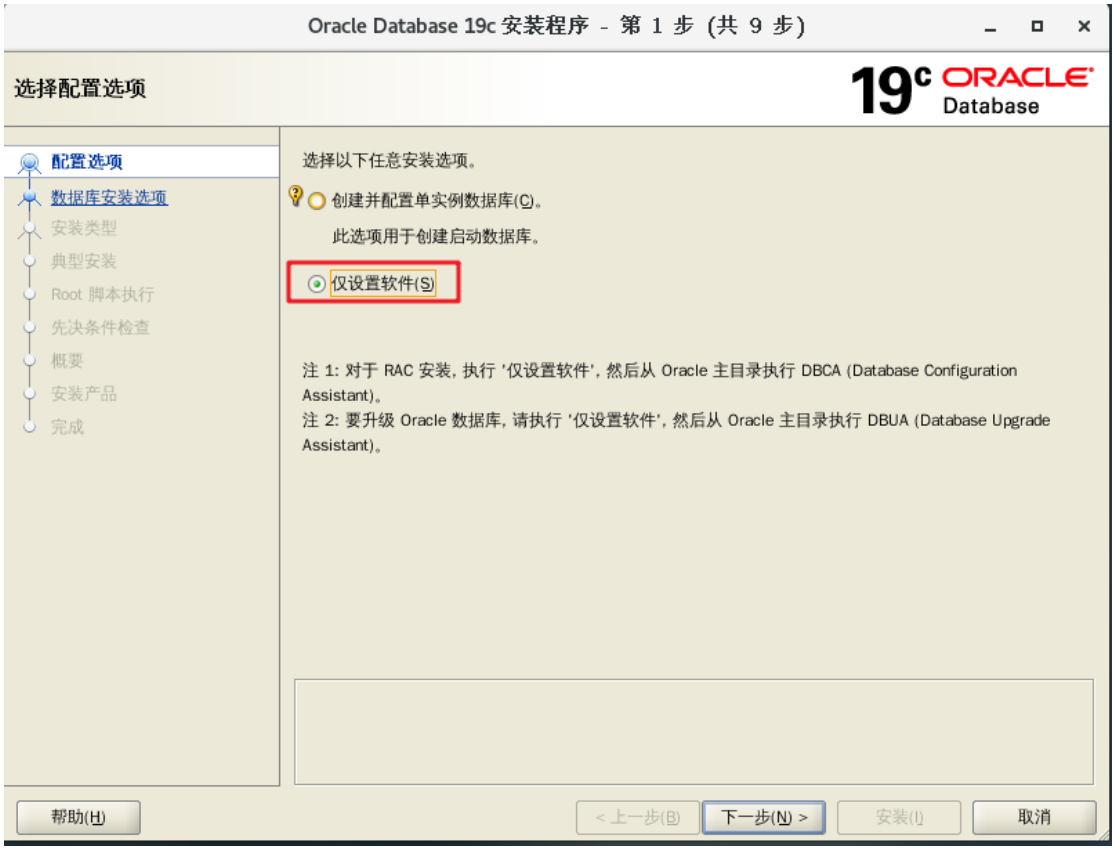
```
#ORACLE_HOME
export ORACLE_BASE=/home/oracle/app/oracle
export ORACLE_HOME=/home/oracle/app/oracle/product/19.3.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export ORACLE_SID=orcl
export NLS_LANG=AMERICAN_AMERICA.ZHS16GBK

[oracle@hadoop102 ~]$ source /home/oracle/.bash_profile
```

4.3.2 进入虚拟机图像化页面操作

```
[oracle@hadoop102 ~]# cd /opt/module/oracle
[oracle@hadoop102 database]# ./runInstaller
```

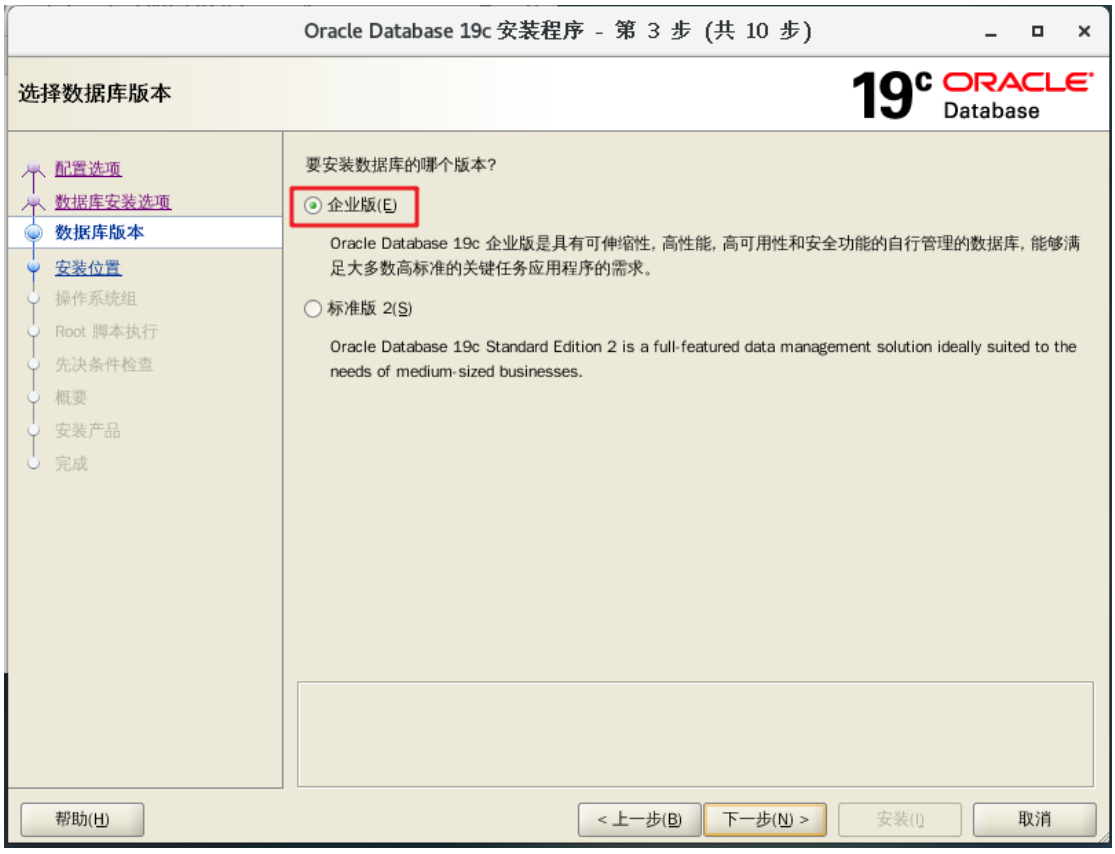
4.3.3 安装数据库

1）选择仅安装数据库软件



2）选择单实例数据库安装

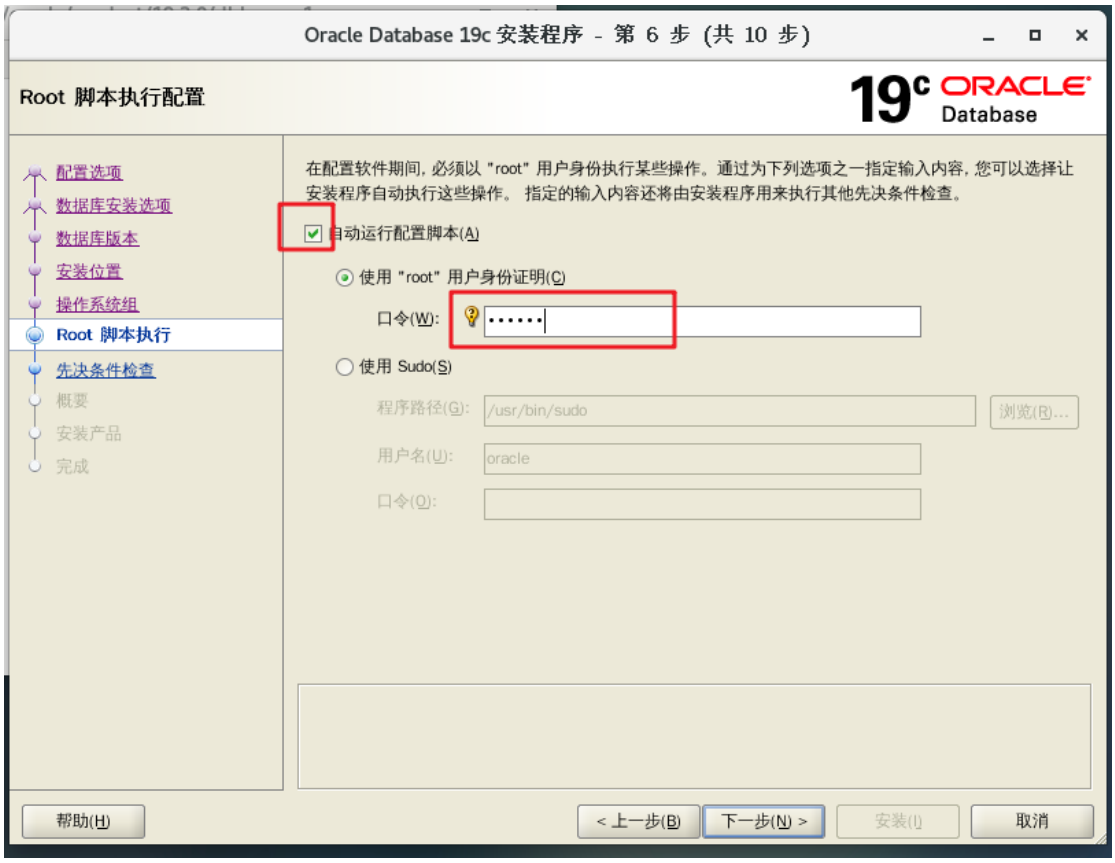
3）选择企业版，默认



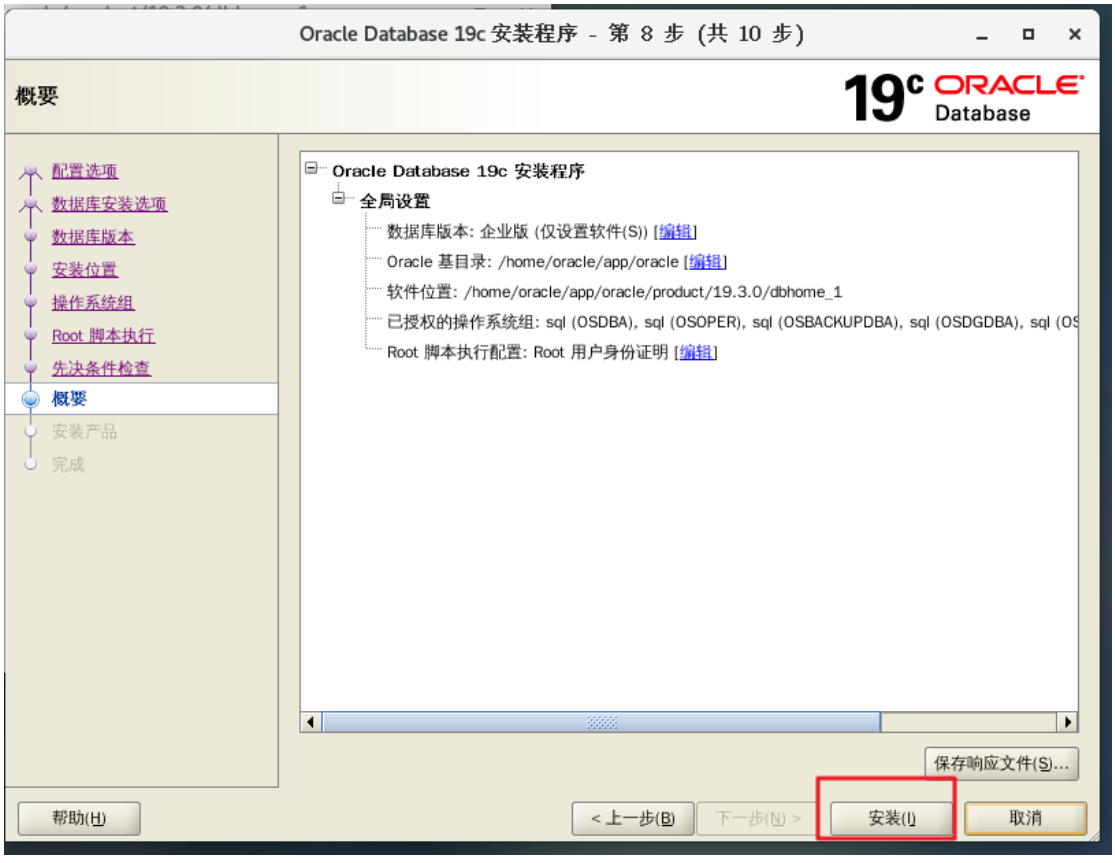
4）设置安装位置

5）操作系统组设置

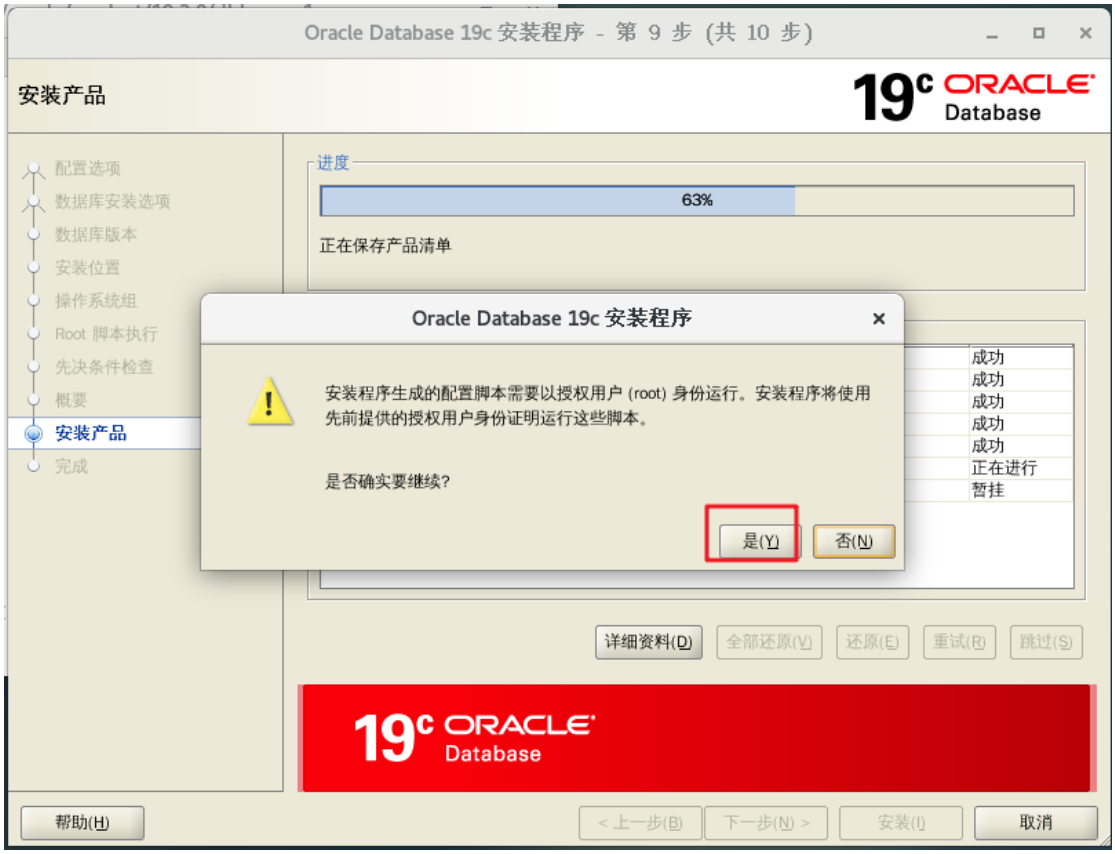
6）配置 root 脚本自动执行



7）条件检查通过后，选择开始安装



8）运行 root 脚本


Oracle Database 19c 安装程序 - 第 9 步 (共 10 步)
安装产品
19c ORACLE Database
配置选项
数据库安装选项
数据库版本
安装位置
操作系统组
Root 脚本执行
先决条件检查
概要
安装产品
完成
进度
63%
正在保存产品清单
Oracle Database 19c 安装程序
安装程序生成的配置脚本需要以授权用户 (root) 身份运行。安装程序将使用先前提供的授权用户身份证明运行这些脚本。
是否确实要继续?
是(Y)
否(N)
成功
成功
成功
成功
成功
正在进行
暂挂
详细资料(D)
全部还原(V)
还原(E)
重试(R)
跳过(S)
帮助(H)
< 上一步(B)
下一步(N) >
安装(I)
取消

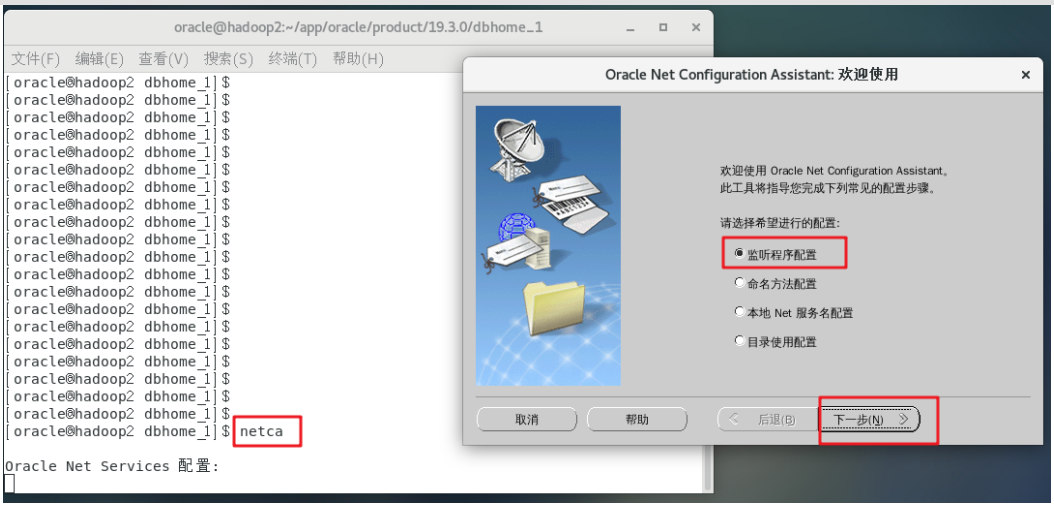
9）安装完成


Oracle Database 19c 安装程序 - 第 10 步 (共 10 步)
完成
19c ORACLE Database
配置选项
数据库安装选项
数据库版本
安装位置
操作系统组
Root 脚本执行
先决条件检查
概要
安装产品
完成
Oracle Database 的 注册 已成功。
帮助(H)
< 上一步(B)
下一步(N) >
安装(I)
关闭(C)

4.4 设置 Oracle 监听

4.4.1 命令行输入以下命令

```
[oracle@hadoop2 ~]$ netca
```



4.4.2 选择添加

4.4.3 设置监听名，默认即可

4.4.4 选择协议，默认即可

4.4.5 设置端口号，默认即可

4.4.6 配置更多监听，默认

4.4.7 完成


Oracle Net Configuration Assistant: 监听程序配置完成
监听程序配置完成!
取消
帮助
后退(B)
下一步(N)

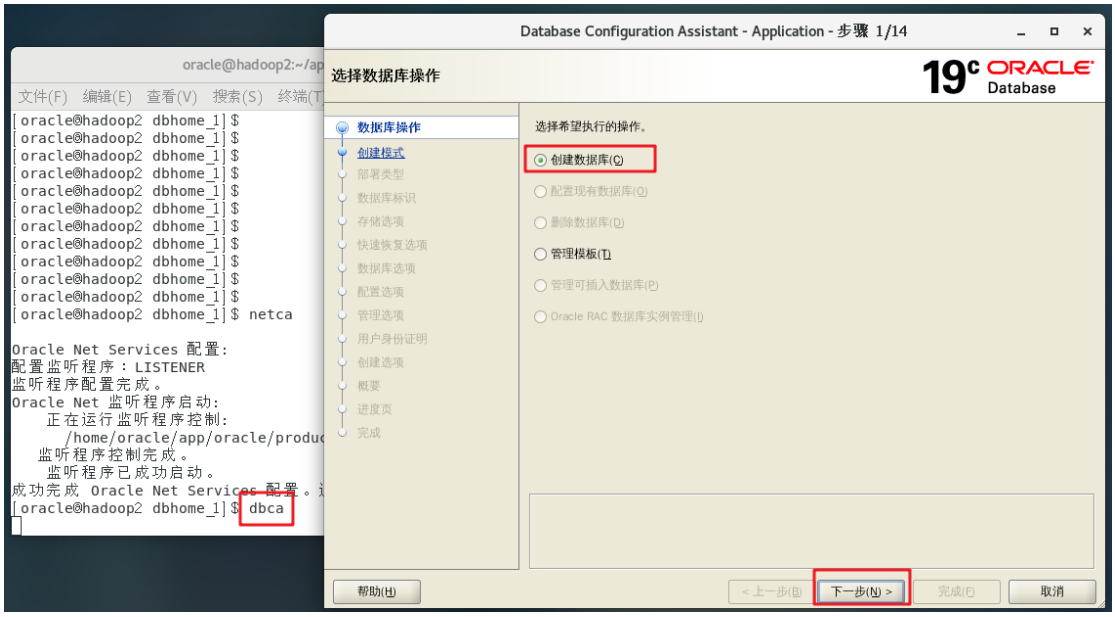

Oracle Net Configuration Assistant: 欢迎使用
欢迎使用 Oracle Net Configuration Assistant。
此工具将指导您完成下列常见的配置步骤。
请选择希望进行的配置:
监听程序配置
命名方法配置
本地 Net 服务名配置
目录使用配置
取消
帮助
后退(B)
下一步(N)
完成(F)

4.5 创建数据库

4.5.1 进入创建页面

```
[oracle@hadoop2 ~]$ dbca
```

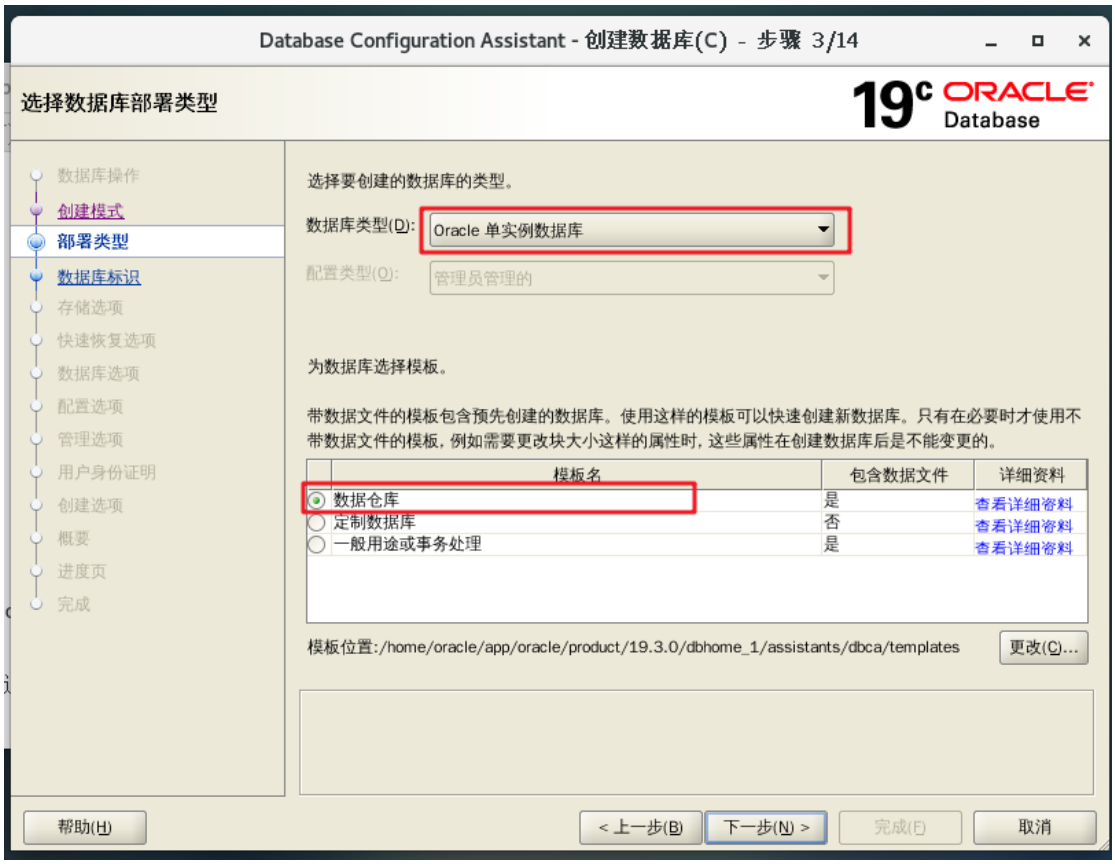
4.5.2 选择创建数据库



4.5.3 选择高级配置


Database Configuration Assistant - 创建数据库(C) - 步骤 2/14
选择数据库创建模式
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
数据库选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
典型配置(T)
全局数据库名(G):
orcl
存储类型(S):
文件系统
数据库文件位置(D):
{ORACLE_BASE}/oradata/{DB_UNIQUE_NAME}
浏览(R)...
快速恢复区 (FRA)(A):
{ORACLE_BASE}/fast_recovery_area/{DB_UNIQUE_NAME}
浏览(O)...
数据库字符集(C):
AL32UTF8 - Unicode UTF-8 通用字符集
管理口令(I):
确认口令(P):
创建为容器数据库(E)
可插入数据库名(L):
高级配置(V)
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.4 选择数据仓库

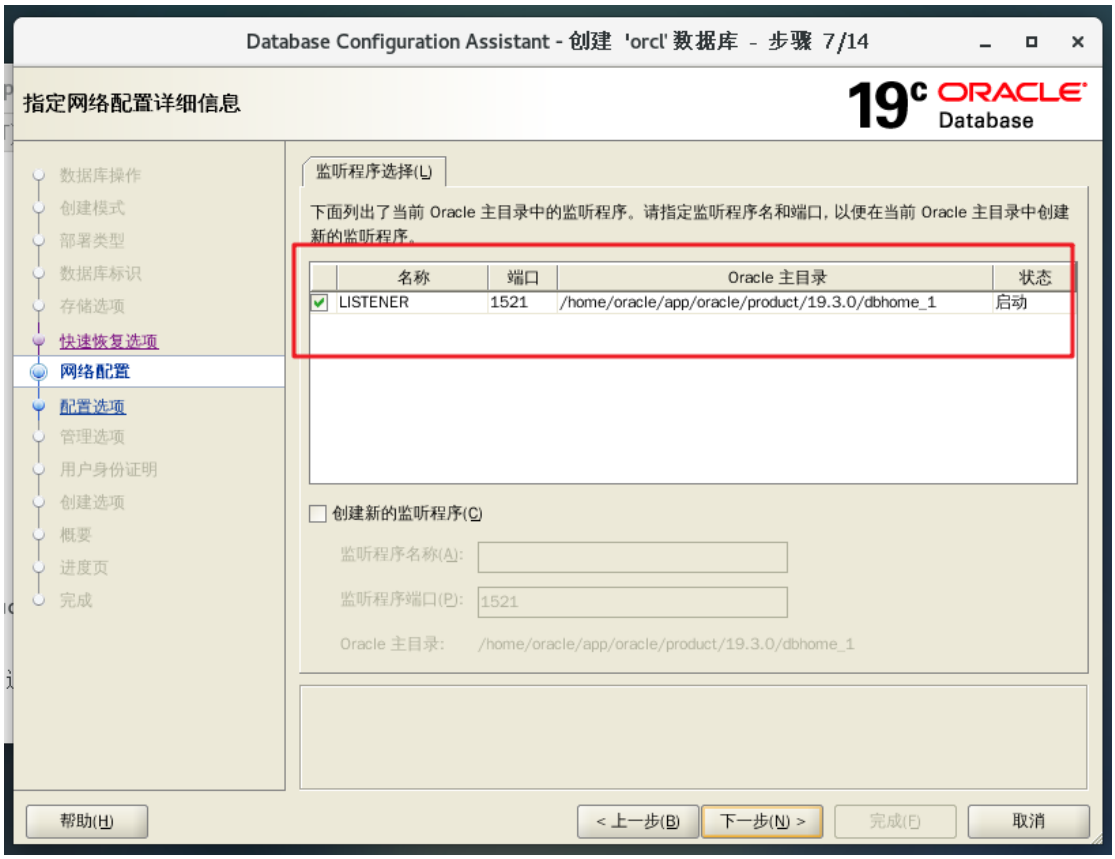

Database Configuration Assistant - 创建数据库(C) - 步骤 3/14
选择数据库部署类型
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
数据库选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
选择要创建的数据库的类型。
数据库类型(D):
Oracle 单实例数据库
配置类型(O):
管理员管理的
为数据库选择模板。
带数据文件的模板包含预先创建的数据库。使用这样的模板可以快速创建新数据库。只有在必要时才使用不带数据文件的模板，例如需要更改块大小这样的属性时，这些属性在创建数据库后是不能变更的。
模板名
包含数据文件
详细资料
数据仓库
是
查看详细资料
定制数据库
否
查看详细资料
一般用途或事务处理
是
查看详细资料
模板位置:/home/oracle/app/oracle/product/19.3.0/dbhome_1/assistants/dbca/templates
更改(C)...
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.5 将图中所示对勾去掉

4.5.6 存储选项

4.5.7 快速恢复选项

4.5.8 选择监听程序



4.5.9 如图设置

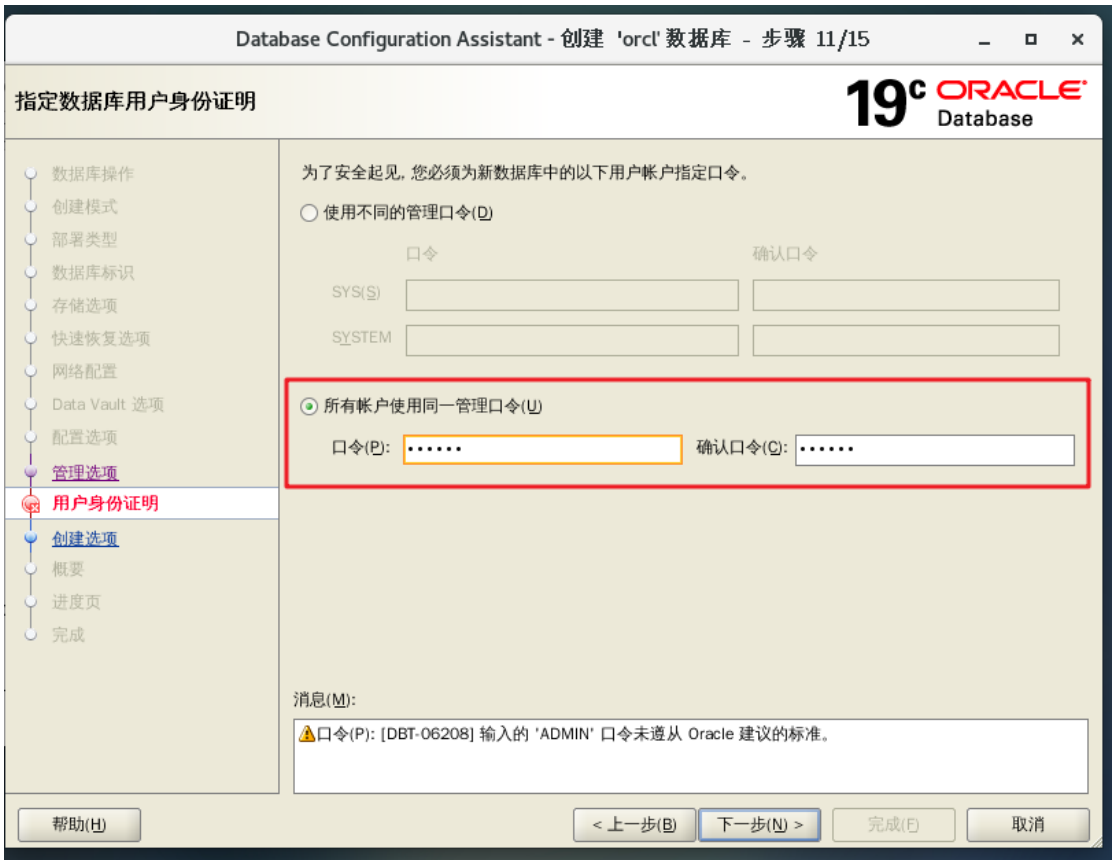

Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 8/15
选择 Oracle Data Vault 配置选项
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
配置 Oracle Database Vault(V)
Database Vault 所有者(O):
口令(P):
确认口令(F):
创建单独的帐户管理员(C)
帐户管理员(A):
口令(S):
确认口令(R):
配置 Oracle Label Security(L)
配置具有以下 OID 的 Oracle Label Security(I)
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.10 使用自动内存管理

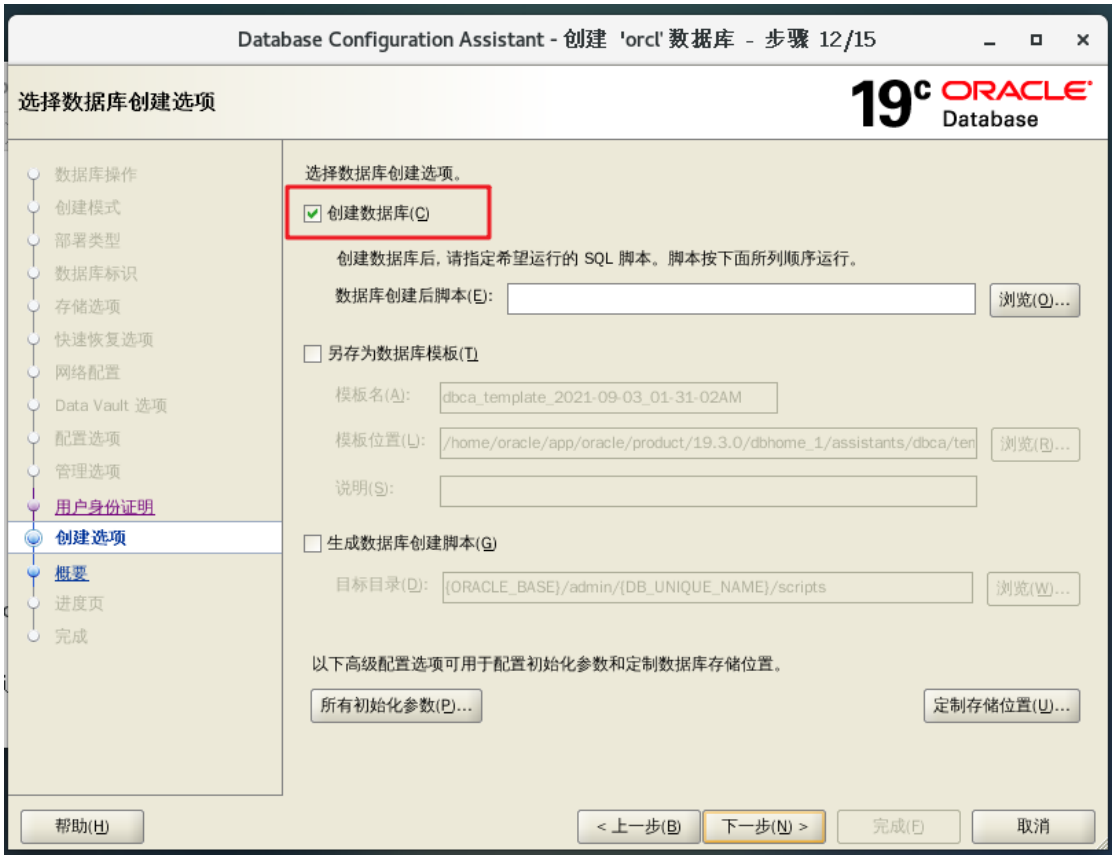
4.5.11 管理选项，默认


Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 10/15
指定管理选项
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
指定数据库的管理选项。
配置 Enterprise Manager (EM) Database Express(C)
EM Database Express 端口(E): 5500
注册到 Enterprise Manager (EM) Cloud Control(R)
OMS 主机(O):
OMS 端口(M):
EM 管理员用户名(U):
EM 管理员口令(A):
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.12 设置统一密码


Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 11/15
指定数据库用户身份证明
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
为了安全起见，您必须为新数据库中的以下用户帐户指定口令。
使用不同的管理口令(D)
口令
确认口令
SYS(S)
SYSTEM
所有帐户使用同一管理口令(U)
口令(P):
确认口令(C):
消息(M):
口令(P): [DBT-06208] 输入的 'ADMIN' 口令未遵从 Oracle 建议的标准。
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.13 创建选项，选择创建数据库



4.5.14 概要，点击完成


Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 13/15
概要
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
Database Configuration Assistant
全局设置
全局数据库名(G): orcl
配置类型(O): Oracle 单实例数据库
SID: orcl
创建为容器数据库(E): 否
内存配置类型: 自动共享内存管理
模板名: Data Warehouse
初始化参数
audit_file_dest: {ORACLE_BASE}/admin/{DB_UNIQUE_NAME}/adump
audit_trail: db
compatible: 19.0.0
control_files: ("{ORACLE_BASE}/oradata/{DB_UNIQUE_NAME}/control01.ctl", "{ORACLE_BAS
db_block_size: 8192 BYTES
db_name: orcl
db_recovery_file_dest: {ORACLE_BASE}/fast_recovery_area/{DB_UNIQUE_NAME}
db_recovery_file_dest_size: 8256 MB
diagnostic_dest: {ORACLE_BASE}
保存响应文件(S)...
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消

4.5.15 等待安装


Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 14/15
进度页
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
进度
40%
正在创建并启动 Oracle 实例：正在进行
状态
数据库创建 正在进行
• 准备执行数据库操作 成功
• 复制数据库文件 成功
• 正在创建并启动 Oracle 实例 正在进行
• 正在进行数据库创建 暂挂
详细资料(D)
全部还原(V)
还原(E)
重试(R)
跳过(S)
DBCA 日志位置:
/home/oracle/app/oracle/cfgtoollogs/dbca/orcl/trace.log_2021-09-03_01-31-02AM
数据库预警日志位置:
/home/oracle/app/oracle/diag/rdbms/orcl/orcl/trace/alert_orcl.log
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
取消


Database Configuration Assistant - 创建 'orcl' 数据库 - 步骤 15/15
完成
19c ORACLE Database
数据库操作
创建模式
部署类型
数据库标识
存储选项
快速恢复选项
网络配置
Data Vault 选项
配置选项
管理选项
用户身份证明
创建选项
概要
进度页
完成
数据库创建完成。有关详细信息，请查看以下位置的日志文件:
/home/oracle/app/oracle/cfgtoollogs/dbca/orcl。
数据库信息:
全局数据库名: orcl
系统标识符 (SID): orcl
服务器参数文件名: /home/oracle/app/oracle/product/19.3.0/dbhome_1/dbs/spfileorcl.ora
EM Database Express URL: https://hadoop2:5500/em
注: 除 SYS 和 SYSTEM 帐户外，所有数据库帐户都已锁定。请选择"口令管理"按钮查看锁定帐户的完整列表或管理数据库帐户。从"口令管理"窗口中，仅解锁您将使用的帐户。Oracle 强烈建议您在解锁帐户后立即更改默认口令。
口令管理...(A)
帮助(H)
< 上一步(B)
下一步(N) >
完成(F)
关闭(C)

4.6 简单使用

4.6.1 开启，关闭监听服务

开启服务：

```
[oracle@hadoop102 ~]$ lsnrctl start
```

关闭服务：

```
[oracle@hadoop102 ~]$ lsnrctl stop
```

4.6.2 进入命令行

```
[oracle@hadoop102 ~]$ sqlplus
SQL*Plus: Release 19.0.0.0.0 - Production on Fri Sep 3 01:44:30 2021
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.
Enter user-name: system
Enter password:  （这里输入之前配置的统一密码）

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL>
```

4.6.3 创建用户并授权

```
SQL> create user atguigu identified by 000000;
User created.
SQL> grant create session,create table,create view,create sequence,unlimited tablespace to
atguigu;
Grant succeeded.
```

4.6.4 进入 atguigu 账号，创建表

```
SQL>create TABLE student(id INTEGER,name VARCHAR2(20));
SQL>insert into student values (1,'zhangsan');
SQL> select * from student;
        ID  NAME
```

```
---------- ----------------------------------------
         1  zhangsan
```

注意：安装完成后重启机器可能出现 ORACLE not available 错误，解决方法如下：

```
[oracle@hadoop102 ~]$ sqlplus / as sysdba
SQL>startup
SQL>conn atguigu
Enter password:
```

4.7 Oracle 与 MySQL 的 SQL 区别

类型	Oracle	MySQL
整型	number(N)/integer	int/integer
浮点型	float	float/double
字符串类型	varchar2(N)	varchar(N)
NULL	''	null 和''不一样
分页	rownum	limit
" "	限制很多，一般不让用	与单引号一样
价格	闭源，收费	开源，免费
主键自动增长	×	√
if not exists	×	√
auto_increment	×	√
create database	×	√
select * from table as t	×	√

4.8 DataX 案例

4.8.1 从 Oracle 中读取数据存到 MySQL

1）MySQL 中创建表

```
[oracle@hadoop102 ~]$ mysql -uroot -p000000
mysql> create database oracle;
mysql> use oracle;
mysql> create table student(id int,name varchar(20));
```

2）编写 datax 配置文件

```
[oracle@hadoop102 ~]$ vim /opt/module/datax/job/oracle2mysql.json


{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "oraclereader",
                    "parameter": {
                        "column": ["*"],
                        "connection": [
                            {
                                "jdbcUrl":
["jdbc:oracle:thin:@hadoop102:1521:orcl"],
                                "table": ["student"]
                            }
                        ],
                        "password": "000000",
                        "username": "atguigu"
                    }
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": ["*"],
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://hadoop102:3306/oracle",
                                "table": ["student"]
                            }
                        ],
                        "password": "000000",
                        "username": "root",
                        "writeMode": "insert"
                    }
                }
            }
```

```
            ],
            "setting": {
                "speed": {
                    "channel": "1"
                }
            }
        }
}
```

3）执行命令

```
[oracle@hadoop102 ~]$
/opt/module/datax/bin/datax.py /opt/module/datax/job/oracle2mysql.json
```

查看结果:

```
mysql> select * from student;
+------+----------+
| id   | name     |
+------+----------+
|    1 | zhangsan |
+------+----------+
```

4.8.2 读取 Oracle 的数据存入 HDFS 中

1）编写配置文件

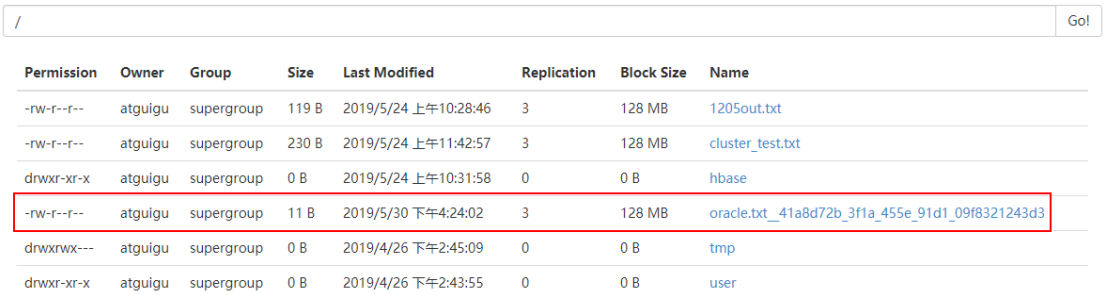
```
[oracle@hadoop102 datax]$ vim job/oracle2hdfs.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "oraclereader",
                    "parameter": {
                        "column": ["*"],
                        "connection": [
                            {
                                "jdbcUrl":
["jdbc:oracle:thin:@hadoop102:1521:orcl"],
                                "table": ["student"]
                            }
                        ],
```

```
                        "password": "000000",
                        "username": "atguigu"
                    }
                },
                "writer": {
                    "name": "hdfswriter",
                    "parameter": {
                        "column": [
                            {
                                "name": "id",
                                "type": "int"
                            },
                            {
                                "name": "name",
                                "type": "string"
                            }

                        ],
                        "defaultFS": "hdfs://hadoop102:9000",
                        "fieldDelimiter": "\t",
                        "fileName": "oracle.txt",
                        "fileType": "text",
                        "path": "/",
                        "writeMode": "append"
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}
```

2）执行

```
[oracle@hadoop102 datax]$ bin/datax.py job/oracle2hdfs.json
```

3）查看 HDFS 结果



/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	atguigu	supergroup	119 B	2019/5/24 上午10:28:46	3	128 MB	1205out.txt
-rw-r--r--	atguigu	supergroup	230 B	2019/5/24 上午11:42:57	3	128 MB	cluster_test.txt
drwxr-xr-x	atguigu	supergroup	0 B	2019/5/24 上午10:31:58	0	0 B	hbase
-rw-r--r--	atguigu	supergroup	11 B	2019/5/30 下午4:24:02	3	128 MB	oracle.txt__41a8d72b_3f1a_455e_91d1_09f8321243d3
drwxrwx---	atguigu	supergroup	0 B	2019/4/26 下午2:45:09	0	0 B	tmp
drwxr-xr-x	atguigu	supergroup	0 B	2019/4/26 下午2:43:55	0	0 B	user

第5章 MongoDB

5.1 什么是 MongoDB

MongoDB 是由 C++语言编写的，是一个基于分布式文件存储的开源数据库系统。MongoDB 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

5.2 MongoDB 优缺点

MongoDB优缺点

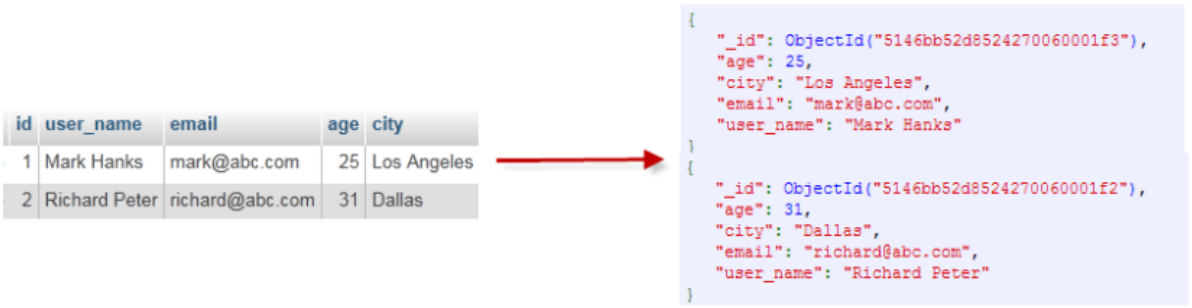
优点： 1. MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易；
2.内置GridFS，支持大容量的存储；
3. 可以在MongoDB记录中设置任何属性的索引；
4. MongoDB支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言；
5.安装简单；
6.复制（复制集）和支持自动故障恢复；
7. MapReduce 支持复杂聚合。

缺点： 1.不支持事务；
2.占用空间过大；
3.不能进行表关联；
4.复杂聚合操作通过MapReduce创建，速度慢；
5. MongoDB 在你删除记录后不会在文件系统回收空间。除非你删掉数据库。

5.3 基础概念解析

SQL 术语/概念	MongoDB 术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins	不支持	表连接,MongoDB 不支持
primary key	primary key	主键,MongoDB 自动将_id 字段设置为主键

通过下图实例，我们也可以更直观的了解 Mongo 中的一些概念：



id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas


```
{
   "_id": ObjectId("5146bb52d8524270060001f3"),
   "age": 25,
   "city": "Los Angeles",
   "email": "mark@abc.com",
   "user_name": "Mark Hanks"
}
{
   "_id": ObjectId("5146bb52d8524270060001f2"),
   "age": 31,
   "city": "Dallas",
   "email": "richard@abc.com",
   "user_name": "Richard Peter"
}
```


5.4 安装

5.4.1 下载地址

https://www.mongodb.com/download-center#community

5.4.2 安装

1）上传压缩包到虚拟机中,解压

```
[atguigu@hadoop102 software]$ tar -zxvf mongodb-linux-x86_64-rhel70-5.0.2.tgz -C /opt/module/
```

2）重命名

```
[atguigu@hadoop102 module]$ mv mongodb-linux-x86_64- rhel70-5.0.2/ mongodb
```

3）创建数据库目录

MongoDB 的数据存储在 data 目录的 db 目录下，但是这个目录在安装过程不会自动创建，所以需要手动创建 data 目录，并在 data 目录中创建 db 目录。

```
[atguigu@hadoop102 module]$ sudo mkdir -p /data/db
[atguigu@hadoop102 mongodb]$ sudo chmod 777 -R /data/db/
```

5）启动 MongoDB 服务

```
[atguigu@hadoop102 mongodb]$ bin/mongod
```

6）进入 shell 页面

```
[atguigu@hadoop102 mongodb]$ bin/mongo
```

5.5 基础概念详解

5.5.1 数据库

一个 mongodb 中可以建立多个数据库。MongoDB 的默认数据库为"db"，该数据库存储

在 data 目录中。MongoDB 的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

1）显示所有数据库

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
```

- admin：从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
- local：这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
- config：当 Mongo 用于分片设置时，config 数据库在内部使用，用于保存分片的相关信息。

2）显示当前使用的数据库

```
> db
test
```

3）切换数据库

```
> use local
switched to db local
> db
local
```

5.5.2 集合

集合就是 MongoDB 文档组，类似于 MySQL 中的 table。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

MongoDB 中使用 createCollection() 方法来创建集合。下面我们来看看如何创建集合：

语法格式：

```
db.createCollection(name, options)
```

参数说明：

name: 要创建的集合名称

options: 可选参数，指定有关内存大小及索引的选项，有以下参数：

字段	类型	描述
capped	布尔	（可选）如果为 true，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。 **当该值为 true 时，必须指定 size 参数。**
autoIndexId	布尔	（可选）如为 true，自动在 _id 字段创建索引。默认为 false。
size	数值	（可选）为固定集合指定一个最大值（以字节计）。 **如果 capped 为 true，也需要指定该字段。**
max	数值	（可选）指定固定集合中包含文档的最大数量。

案例 1：在 test 库中创建一个 atguigu 的集合

```
> use test
switched to db test
> db.createCollection("atguigu")
{ "ok" : 1 }
> show collections
Atguigu

//插入数据
> db.atguigu.insert({"name":"atguigu","url":"www.atguigu.com"})
WriteResult({ "nInserted" : 1 })
//查看数据
> db.atguigu.find()
{ "_id" : ObjectId("5d0314ceecb77ee2fb2d7566"), "name" : "atguigu", "url" :
"www.atguigu.com" }
```

说明：

ObjectId 类似唯一主键，可以很快的去生成和排序，包含 12 bytes，由 24 个 16 进制数字组成的字符串（每个字节可以存储两个 16 进制数字）,含义是：

- 前 4 个字节表示创建 unix 时间戳
- 接下来的 3 个字节是机器标识码
- 紧接的两个字节由进程 id 组成 PID
- 最后三个字节是随机数

案例 2：创建一个固定集合 mycol

```
> db.createCollection("mycol",{ capped : true,autoIndexId : true,size : 6142800, max :
1000})
> show tables;
atguigu
mycol
```

案例 3：自动创建集合

在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

```
> db.mycol2.insert({"name":"atguigu"})
WriteResult({ "nInserted" : 1 })
> show collections
atguigu
mycol
mycol2
```

案例 4：删除集合

```
> db.mycol2.drop()
True
> show tables;
atguigu
mycol
```

5.5.3 文档(Document)

文档是一组键值(key-value)对组成。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

一个简单的例子：

```
{"name":"atguigu"}
```

注意：

1、文档中的键/值对是有序的。

2、MongoDB 区分类型和大小写。

3、MongoDB 的文档不能有重复的键。

4、文档的键是字符串。除了少数例外情况，键可以使用任意 UTF-8 字符。

5.6 DataX 导入导出案例

5.6.1 读取 MongoDB 的数据导入到 HDFS

1）编写配置文件

```
[atguigu@hadoop102 datax]$ vim job/mongdb2hdfs.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mongodbreader",
                    "parameter": {
                        "address": ["127.0.0.1:27017"],
                        "collectionName": "atguigu",
                        "column": [
                            {
                                "name":"name",
                                "type":"string"
                            },
                            {
                                "name":"url",
                                "type":"string"
                            }
                        ],
                        "dbName": "test",
                    }
                },
                "writer": {
                    "name": "hdfswriter",
                    "parameter": {
                        "column": [
                            {
                                "name":"name",
                                "type":"string"
                            },
                            {
                                "name":"url",
```

```
                                    "type":"string"
                                }
                            ],
                            "defaultFS": "hdfs://hadoop102:9000",
                            "fieldDelimiter": "\t",
                            "fileName": "mongo.txt",
                            "fileType": "text",
                            "path": "/",
                            "writeMode": "append"
                        }
                    }
                }
            ],
            "setting": {
                "speed": {
                    "channel": "1"
                }
            }
        }
}
```

2）mongodbreader 参数解析

- address： MongoDB 的数据地址信息，因为 MonogDB 可能是个集群，则 ip 端口信息需要以 Json 数组的形式给出。【必填】
- userName：MongoDB 的用户名。【选填】
- userPassword： MongoDB 的密码。【选填】
- collectionName： MonogoDB 的集合名。【必填】
- column：MongoDB 的文档列名。【必填】
- name：Column 的名字。【必填】
- type：Column 的类型。【选填】
- splitter：因为 MongoDB 支持数组类型，但是 Datax 框架本身不支持数组类型，所以 mongoDB 读出来的数组类型要通过这个分隔符合并成字符串。【选填】

3）执行

```
[atguigu@hadoop102 datax]$ bin/datax.py job/mongdb2hdfs.json
```

4）查看结果

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	atguigu	supergroup	0 B	Tue Apr 30 09:03:52 +0800 2019	0	0 B	hbase
-rw-r--r--	atguigu	supergroup	24 B	Fri Jun 14 11:53:22 +0800 2019	3	128 MB	mongo.txt__1080895d_ec8a_4287_a665_5c0ceba5700f
-rw-r--r--	atguigu	supergroup	110 B	Tue Apr 30 08:57:40 +0800 2019	3	128 MB	out.txt
-rw-r--r--	root	supergroup	27 B	Wed Jun 12 14:09:00 +0800 2019	3	128 MB	student.csv
drwxrwx---	atguigu	supergroup	0 B	Fri Apr 26 14:45:09 +0800 2019	0	0 B	tmp
drwxr-xr-x	atguigu	supergroup	0 B	Fri Apr 26 14:43:55 +0800 2019	0	0 B	user

5.6.2 读取 MongoDB 的数据导入 MySQL

1）在 MySQL 中创建表

```
mysql> create table atguigu(name varchar(20),url varchar(20));
```

2）编写 DataX 配置文件

```
[atguigu@hadoop102 datax]$ vim job/mongodb2mysql.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mongodbreader",
                    "parameter": {
                        "address": ["127.0.0.1:27017"],
                        "collectionName": "atguigu",
                        "column": [
                            {
                                "name":"name",
                                "type":"string"
                            },
                            {
                                "name":"url",
                                "type":"string"
                            }
                        ],
                        "dbName": "test",
                    }
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
```

```
                                "column": ["*"],
                                "connection": [
                                    {
                                        "jdbcUrl": "jdbc:mysql://hadoop102:3306/test",
                                        "table": ["atguigu"]
                                    }
                                ],
                                "password": "000000",
                                "username": "root",
                                "writeMode": "insert"
                            }
                        }
                    }
                ],
                "setting": {
                    "speed": {
                        "channel": "1"
                    }
                }
            }
        }
```

3）执行

```
[atguigu@hadoop102 datax]$ bin/datax.py job/mongodb2mysql.json
```

4）查看结果

```
mysql> select * from atguigu;
+---------+-----------------+
| name    | url             |
+---------+-----------------+
| atguigu | www.atguigu.com |
+---------+-----------------+
```

第6章 SQLServer

6.1 什么是 SQLServer

美国 Microsoft 公司推出的一种关系型数据库系统。SQL Server 是一个可扩展的、高性

能的、为分布式客户机/服务器计算所设计的数据库管理系统，实现了与 WindowsNT 的有机结合，提供了基于事务的企业级信息管理系统方案。SQL Server 的基本语法和 MySQL 基本相同。

（1）高性能设计，可充分利用 WindowsNT 的优势。

（2）系统管理先进，支持 Windows 图形化管理工具，支持本地和远程的系统管理和配置。

（3）强壮的事务处理功能，采用各种方法保证数据的完整性。

（4）支持对称多处理器结构、存储过程、ODBC，并具有自主的 SQL 语言。 SQLServer 以其内置的数据复制功能、强大的管理工具、与 Internet 的紧密集成和开放的系统结构为广大的用户、开发人员和系统集成商提供了一个出众的数据库平台。

6.2 安装

6.2.1 安装要求

系统要求：

1、centos 或 redhat7.0 以上系统

2、内存 2G 以上

说明：

linux 下安装 sqlserver 数据库有 2 种办法：

- 使用 rpm 安装包安装

  rpm 安装包地址：https://packages.microsoft.com/rhel/7/mssql-server-2017/

  安装时缺少什么依赖，就使用 yum 进行安装补齐

- 使用 yum 镜像安装

6.2.2 安装步骤

1）下载 Microsoft SQL Server 2017 Red Hat 存储库配置文件

```
sudo curl -o /etc/yum.repos.d/mssql-server.repo
https://packages.microsoft.com/config/rhel/7/mssql-server-2017.repo
```

2）执行安装

```
yum install -y mssql-server
```

3）完毕之后运行做相关配置

```
sudo /opt/mssql/bin/mssql-conf setup
```

6.2.3 安装配置

1）执行配置命令

```
sudo /opt/mssql/bin/mssql-conf setup
```

2）选择安装的版本

```
[atguigu@hadoop1 mssql]$ sudo /opt/mssql/bin/mssql-conf setup
选择 SQL Server 的一个版本:
  1) Evaluation (免费，无生产许可，180 天限制)
  2) Developer (免费，无生产许可)
  3) Express (免费)
  4) Web (付费版)
  5) Standard (付费版)
  6) Enterprise (付费版)
  7) Enterprise Core (付费版)
  8) 我通过零售渠道购买了许可证并具有要输入的产品密钥。

可在以下位置找到有关版本的详细信息:
https://go.microsoft.com/fwlink/?LinkId=852748&clcid=0x804

使用此软件的付费版本需要通过以下途径获取单独授权
Microsoft 批量许可计划。
选择付费版本即表示你具有适用的
要安装和运行此软件的就地许可证数量。

输入版本(1-8): 2
```

3）接受许可条款

```
可以在以下位置找到此产品的许可条款:
/usr/share/doc/mssql-server 或从以下位置下载:
https://go.microsoft.com/fwlink/?LinkId=855862&clcid=0x804

可以从以下位置查看隐私声明:
https://go.microsoft.com/fwlink/?LinkId=853010&clcid=0x804

接受此许可条款吗? [Yes/No]:yes
```

3）选择语言

```
选择 SQL Server 的语言:
(1) English
(2) Deutsch
(3) Español
(4) Français
(5) Italiano
(6) 日本語
(7) 한 국 어
(8) Português
(9) Русский
(10) 中文 - 简体
(11) 中文 (繁体)
输入选项 1-11:10
```

4）配置系统管理员密码

```
指定的密码不符合 SQL Server 密码策略要求，因为它不够复杂。密码必须至少包含 8 个字符，并包含以下四种字符集中的任意三种：大写字母、小写字母、数字和符号。
输入 SQL Server 系统管理员密码:
确认 SQL Server 系统管理员密码:
正在配置 SQL Server...
```

5）完成

```
正在配置 SQL Server...

ForceFlush is enabled for this instance.
ForceFlush feature is enabled for log durability.
Created symlink from /etc/systemd/system/multi-user.target.wants/mssql-server.service to /usr/lib/systemd/system/mssql-server.service.
安装程序已成功完成。SQL Server 正在启动。
[atguigu@hadoop1 mssql]$
```

6.2.4 安装命令行工具

1）下载存储库配置文件

```
sudo curl -o /etc/yum.repos.d/msprod.repo
https://packages.microsoft.com/config/rhel/7/prod.repo
```

2）执行安装

```
sudo yum remove mssql-tools unixODBC-utf16-devel
sudo yum install mssql-tools unixODBC-devel
```

3）配置环境变量

```
sudo vim /etc/profile.d/my_env.sh
#添加环境变量
export PATH="$PATH:/opt/mssql-tools/bin


source /etc/profile.d/my_env.sh
```

4）进入命令行

```
sqlcmd -S localhost -U SA -P 密码 # 用命令行连接
```

6.3 简单使用

6.3.1 启停命令

```
#启动
systemctl start mssql-server
#重启
systemctl restart mssql-server
#停止
systemctl stop mssql-server
#查看状态
systemctl status mssql-server
#具体配置路径
/opt/mssql/bin/mssql-conf
```

6.3.2 创建数据库

1) 建库

```
> create database datax
> go
```

(2) 看当前数据库列表

```
> select * from SysDatabases
> go
```

(3) 看当前数据表

```
> use 库名
> select * from sysobjects where xtype='u'
> go
```

(4) 看表的内容

```
> select * from 表名;
> go
```

6.4 DataX 导入导出案例

创建表并插入数据

```
create table student(id int,name varchar(25))
go

insert into student values(1,'zhangsan')
go
```

6.4.1 读取 SQLServer 的数据导入到 HDFS

1）编写配置文件

```
[atguigu@hadoop102 datax]$ vim job/sqlserver2hdfs.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "sqlserverreader",
                    "parameter": {
                        "column": [
```

```
                                "id",
                                "name"
                            ],
                            "connection": [
                                {
                                    "jdbcUrl": [
"jdbc:sqlserver://hadoop2:1433;DatabaseName=datax"
                                    ],
                                    "table": [
                                        "student"
                                    ]
                                }
                            ],
                            "username": "root",
                            "password": "000000"
                        }
                    },
                    "writer": {
                        "name": "hdfswriter",
                        "parameter": {
                            "column": [
                                {
                                    "name": "id",
                                    "type": "int"
                                },
                                {
                                    "name": "name",
                                    "type": "string"
                                }
                            ],
                            "defaultFS": "hdfs://hadoop102:9000",
                            "fieldDelimiter": "\t",
                            "fileName": "sqlserver.txt",
                            "fileType": "text",
                            "path": "/",
                            "writeMode": "append"
                        }
```

```
                    }
                }
            ],
            "setting": {
                "speed": {
                    "channel": "1"
                }
            }
        }
}
```

6.4.2 读取 SQLServer 的数据导入 MySQL

```
[atguigu@hadoop102 datax]$ vim job/sqlserver2mysql.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "sqlserverreader",
                    "parameter": {
                        "column": [
                            "id",
                            "name"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": [

"jdbc:sqlserver://hadoop2:1433;DatabaseName=datax"
                                ],
                                "table": [
                                    "student"
                                ]
                            }
                        ],
                        "username": "root",
                        "password": "000000"
                    }
```

```
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": ["*"],
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://hadoop102:3306/datax",
                                "table": ["student"]
                            }
                        ],
                        "password": "000000",
                        "username": "root",
                        "writeMode": "insert"
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}
```

第7章 DB2

7.1 什么是 db2

DB2 是 IBM 公司于 1983 年研制的一种关系型数据库系统(Relational Database Management System)，主要应用于大型应用系统，具有较好的可伸缩性 。DB2 是 IBM 推出的第二个关系型数据库，所以称为 db2。DB2 提供了高层次的数据利用性、完整性、安全性 、并行性、可恢复性，以及小规模到大规模应用程序的执行能力，具有与平台无关的基本功能和 SQL 命令运行环境。可以同时在不同操作系统使用，包括 Linux、UNIX 和 Windows。

7.2 db2 数据库对象关系

1、instance，同一台机器上可以安装多个 DB2 instance。

2、database，同一个 instance 下面可以创建有多个 database。

3、schema，同一个 database 下面可以配置多个 schema。

4、table，同一个 schema 下可以创建多个 table。

7.3 安装前的准备

7.3.1 安装依赖

```
yum install -y bc binutils compat-libcap1 compat-libstdc++33 elfutils-libelf elfutils-libelf-devel fontconfig-devel glibc glibc-devel ksh libaio libaio-devel libX11 libXau libXi libXtst libXrender libXrender-devel libgcc libstdc++ libstdc++-devel libxcb make smartmontools sysstat kmod* gcc-c++ compat-libstdc++-33 libstdc++.so.6 kernel-devel pam-devel.i686 pam.i686 pam32*
```

7.3.2 修改配置文件 sysctl.conf

```
[root@hadoop102 module]# vim /etc/sysctl.conf
```

删除里面的内容，添加如下内容：

```
net.ipv4.ip_local_port_range = 9000 65500
fs.file-max = 6815744
kernel.shmall = 10523004
kernel.shmmax = 6465333657
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
net.core.rmem_default=262144
net.core.wmem_default=262144
net.core.rmem_max=4194304
net.core.wmem_max=1048576
fs.aio-max-nr = 1048576
```

7.3.3 修改配置文件 limits.conf

```
[root@hadoop102 module]# vim /etc/security/limits.conf
```

在文件末尾添加：

```
* soft nproc 65536
* hard nproc 65536
* soft nofile 65536
* hard nofile 65536
```

重启机器生效。

7.3.4 上传安装包并解压

```
[root@hadoop102 software]# tar -zxvf v11.5.4_linuxx64_server_dec.tar.gz -C /opt/module/
[root@hadoop102 module]# chmod 777 server_dec
```

7.4 安装

在 root 用户下操作

7.4.1 执行预检查命令

```
./db2prereqcheck -l -s //检查环境
```

7.4.2 执行安装

```
./db2_install
```

1）接受许可条款

```
[root@hadoop2 server_dec]# ./db2_install
请阅读 db2/license 目录中的许可协议文件。

************************************************************
要接受这些条款，请输入“是”。否则，请输入“否”以取消安装过程。[是/否]
是
```

可能会出现两次询问是否接受条款，都选“是”即可。

2）确认安装路径，默认

```
安装产品的缺省目录 - /opt/ibm/db2/v11.5

************************************************************
是否安装到缺省目录 (/opt/ibm/db2/v11.5) 中？[是/否]
是
```

3）选择安装 SERVER

```
指定下列其中一个关键字以安装 Db2 产品。

   SERVER
   CONSV
   CLIENT
   RTCL

输入 "help" 以重新显示产品名称。

输入 "quit" 以退出。

************************************************************
SERVER
```

4）不安装 pureScale

```
************************************************************
要安装 Db2 pureScale Feature 吗？[是/否]
否
```

等待安装完成即可

5）查看许可

```
/opt/ibm/db2/V11.5/adm/db2licm -l
```

7.4.3 添加组和用户

```
groupadd -g 2000 db2iadm1
groupadd -g 2001 db2fadm1
useradd -m -g db2iadm1 -d /home/db2inst1 db2inst1
useradd -m -g db2fadm1 -d /home/db2fenc1 db2fenc1
passwd db2inst1
passwd db2fenc1
```

- db2inst1：实例所有者
- db2fenc1：受防护用户

7.4.4 创建实例

```
cd /opt/ibm/db2/V11.5/instance
./db2icrt -p 50000 -u db2fenc1 db2inst1
```

7.4.5 创建样本数据库、开启服务

```
su - db2inst1
db2sampl
db2start
```

7.4.6 连接

```
db2
conncet to sample #连接到某个数据库
select * from staff
```

7.4.7 创建表、插入数据

```
CREATE TABLE STUDENT(ID int ,NAME varchar(20));
INSERT INTO STUDENT VALUES(11, 'lisi');
commit;
```

7.5 DataX 导入导出案例

7.5.1 注册 db2 驱动

datax 暂时没有独立插件支持 db2，需要使用通用的使用 rdbmsreader 或 rdbmswriter。

1）注册 reader 的 db2 驱动

```
[atguigu@hadoop102 datax]$ vim /opt/module/datax/plugin/reader/rdbmsreader/plugin.json
#在 drivers 里添加 db2 的驱动类
"drivers":["dm.jdbc.driver.DmDriver", "com.sybase.jdbc3.jdbc.SybDriver",
"com.edb.Driver","com.ibm.db2.jcc.DB2Driver"]
```

2）注册 writer 的 db2 驱动

```
[atguigu@hadoop102 datax]$ vim /opt/module/datax/plugin/writer/rdbmswriter/plugin.json
#在 drivers 里添加 db2 的驱动类
"drivers":["dm.jdbc.driver.DmDriver", "com.sybase.jdbc3.jdbc.SybDriver",
"com.edb.Driver","com.ibm.db2.jcc.DB2Driver"]
```

7.5.2 读取 DB2 的数据导入到 HDFS

1）编写配置文件

```
[atguigu@hadoop102 datax]$ vim job/db2-2-hdfs.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "rdbmsreader",
```

```
                "parameter": {
                    "column": [
                        "ID",
                        "NAME"
                    ],
                    "connection": [
                        {
                            "jdbcUrl": [
                                "jdbc:db2://hadoop2:50000/sample"
                            ],
                            "table": [
                                "STUDENT"
                            ]
                        }
                    ],
                    "username": "db2inst1",
                    "password": "atguigu"
                }
            },
            "writer": {
                "name": "hdfswriter",
                "parameter": {
                    "column": [
                        {
                            "name": "id",
                            "type": "int"
                        },
                        {
                            "name": "name",
                            "type": "string"
                        }
                    ],
                    "defaultFS": "hdfs://hadoop102:9000",
                    "fieldDelimiter": "\t",
                    "fileName": "db2.txt",
                    "fileType": "text",
                    "path": "/",
                    "writeMode": "append"
```

```
                        }
                    }
                }
            ],
            "setting": {
                "speed": {
                    "channel": "1"
                }
            }
        }
}
```

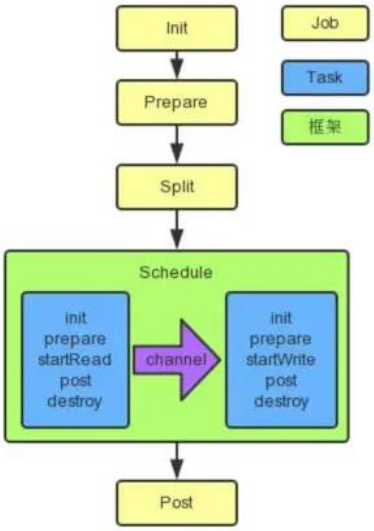
7.5.3 读取 DB2 的数据导入 MySQL

```
[atguigu@hadoop102 datax]$ vim job/db2-2-mysql.json
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "rdbmsreader",
                    "parameter": {
                        "column": [
                            "ID",
                            "NAME"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": [
                                    "jdbc:db2://hadoop2:50000/sample"
                                ],
                                "table": [
                                    "STUDENT"
                                ]
                            }
                        ],
                        "username": "db2inst1",
                        "password": "atguigu"
                    }
```

```
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": ["*"],
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://hadoop102:3306/datax",
                                "table": ["student"]
                            }
                        ],
                        "password": "000000",
                        "username": "root",
                        "writeMode": "insert"
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}
```

第8章 执行流程源码分析

8.1 总体流程



- 黄色： Job 部分的执行阶段，
- 蓝色： Task 部分的执行阶段，
- 绿色：框架执行阶段。

8.2 程序入口

datax.py

```
……
ENGINE_COMMAND = "java -server ${jvm} %s -classpath %s    ${params}
com.alibaba.datax.core.Engine -mode ${mode} -jobid ${jobid} -job ${job}" % (
        DEFAULT_PROPERTY_CONF, CLASS_PATH)
……
```

Engine.java

```
    public void start(Configuration allConf) {

        ……

        //JobContainer 会在 schedule 后再行进行设置和调整值
        int channelNumber =0;
        AbstractContainer container;
```

```
            long instanceId;
            int taskGroupId = -1;
            if (isJob) {
                allConf.set(CoreConstant.DATAX_CORE_CONTAINER_JOB_MODE,
RUNTIME_MODE);
                container = new JobContainer(allConf);
                instanceId = allConf.getLong(
                        CoreConstant.DATAX_CORE_CONTAINER_JOB_ID, 0);

            } else {
                container = new TaskGroupContainer(allConf);
                instanceId = allConf.getLong(
                        CoreConstant.DATAX_CORE_CONTAINER_JOB_ID);
                taskGroupId = allConf.getInt(
                        CoreConstant.DATAX_CORE_CONTAINER_TASKGROUP_ID);
                channelNumber = allConf.getInt(

CoreConstant.DATAX_CORE_CONTAINER_TASKGROUP_CHANNEL);
            }
            ……
            container.start();
        }
```

JobContainer.java

```
    /**
     * jobContainer 主要负责的工作全部在 start()里面，包括 init、prepare、split、
scheduler、
     * post 以及 destroy 和 statistics
     */
    @Override
    public void start() {
        LOG.info("DataX jobContainer starts job.");

        boolean hasException = false;
        boolean isDryRun = false;
        try {
            this.startTimeStamp = System.currentTimeMillis();
            isDryRun =
configuration.getBool(CoreConstant.DATAX_JOB_SETTING_DRYRUN, false);
```

```
                if(isDryRun) {
                    LOG.info("jobContainer starts to do preCheck ...");
                    this.preCheck();
                } else {
                    userConf = configuration.clone();
                    LOG.debug("jobContainer starts to do preHandle ...");
                    //Job 前置操作
                    this.preHandle();

                    LOG.debug("jobContainer starts to do init ...");
                    //初始化 reader 和 writer
                    this.init();
                    LOG.info("jobContainer starts to do prepare ...");
                    //全局准备工作，比如 odpswriter 清空目标表
                    this.prepare();
                    LOG.info("jobContainer starts to do split ...");
                    //拆分 Task
                    this.totalStage = this.split();
                    LOG.info("jobContainer starts to do schedule ...");
                    this.schedule();
                    LOG.debug("jobContainer starts to do post ...");
                    this.post();

                    LOG.debug("jobContainer starts to do postHandle ...");
                    this.postHandle();
                    LOG.info("DataX jobId [{}] completed successfully.", this.jobId);

                    this.invokeHooks();
                }
            } ……
        }
```

8.3 Task 切分逻辑

JobContainer.java

```
    private int split() {
        this.adjustChannelNumber();
```

```
        if (this.needChannelNumber <= 0) {
            this.needChannelNumber = 1;
        }

        List<Configuration> readerTaskConfigs = this
                .doReaderSplit(this.needChannelNumber);
        int taskNumber = readerTaskConfigs.size();
        List<Configuration> writerTaskConfigs = this
                .doWriterSplit(taskNumber);

        List<Configuration> transformerList =
this.configuration.getListConfiguration(CoreConstant.DATAX_JOB_CONTENT_TRANSF
ORMER);

        LOG.debug("transformer configuration: "+ JSON.toJSONString(transformerList));
        /**
         * 输入是 reader 和 writer 的 parameter list，输出是 content 下面元素的 list
         */
        List<Configuration> contentConfig = mergeReaderAndWriterTaskConfigs(
                readerTaskConfigs, writerTaskConfigs, transformerList);


        LOG.debug("contentConfig configuration: "+
JSON.toJSONString(contentConfig));

        this.configuration.set(CoreConstant.DATAX_JOB_CONTENT, contentConfig);

        return contentConfig.size();
    }
```

8.3.1 并发数的确定

```
    private void adjustChannelNumber() {
        int needChannelNumberByByte = Integer.MAX_VALUE;
        int needChannelNumberByRecord = Integer.MAX_VALUE;

        boolean isByteLimit = (this.configuration.getInt(
                CoreConstant.DATAX_JOB_SETTING_SPEED_BYTE, 0) > 0);
```

```
        if (isByteLimit) {
            long globalLimitedByteSpeed = this.configuration.getInt(
                    CoreConstant.DATAX_JOB_SETTING_SPEED_BYTE, 10 * 1024
* 1024);

            // 在 byte 流控情况下，单个 Channel 流量最大值必须设置，否则报错！
            Long channelLimitedByteSpeed = this.configuration
                    .getLong(CoreConstant.DATAX_CORE_TRANSPORT_CHANNE
L_SPEED_BYTE);
            if (channelLimitedByteSpeed == null || channelLimitedByteSpeed <= 0) {
                throw DataXException.asDataXException(
                        FrameworkErrorCode.CONFIG_ERROR,
                        "在有总 bps 限速条件下，单个 channel 的 bps 值不能为
空，也不能为非正数");
            }

            needChannelNumberByByte =
                    (int) (globalLimitedByteSpeed / channelLimitedByteSpeed);
            needChannelNumberByByte =
                    needChannelNumberByByte > 0 ? needChannelNumberByByte : 1;
            LOG.info("Job set Max-Byte-Speed to " + globalLimitedByteSpeed + "
bytes.");
        }

        boolean isRecordLimit = (this.configuration.getInt(
                CoreConstant.DATAX_JOB_SETTING_SPEED_RECORD, 0)) > 0;
        if (isRecordLimit) {
            long globalLimitedRecordSpeed = this.configuration.getInt(
                    CoreConstant.DATAX_JOB_SETTING_SPEED_RECORD,
100000);

            Long channelLimitedRecordSpeed = this.configuration.getLong(

CoreConstant.DATAX_CORE_TRANSPORT_CHANNEL_SPEED_RECORD);
            if (channelLimitedRecordSpeed == null || channelLimitedRecordSpeed <= 0)
{
                throw
DataXException.asDataXException(FrameworkErrorCode.CONFIG_ERROR,
```

```
                    "在有总 tps 限速条件下，单个 channel 的 tps 值不能为空，
也不能为非正数");
                }

                needChannelNumberByRecord =
                        (int) (globalLimitedRecordSpeed / channelLimitedRecordSpeed);
                needChannelNumberByRecord =
                        needChannelNumberByRecord > 0 ?
needChannelNumberByRecord : 1;
                LOG.info("Job set Max-Record-Speed to " + globalLimitedRecordSpeed + "
records.");
            }

            // 取较小值
            this.needChannelNumber = needChannelNumberByByte <
needChannelNumberByRecord ?
                    needChannelNumberByByte : needChannelNumberByRecord;

            // 如果从 byte 或 record 上设置了 needChannelNumber 则退出
            if (this.needChannelNumber < Integer.MAX_VALUE) {
                return;
            }

            boolean isChannelLimit = (this.configuration.getInt(
                    CoreConstant.DATAX_JOB_SETTING_SPEED_CHANNEL, 0) > 0);
            if (isChannelLimit) {
                this.needChannelNumber = this.configuration.getInt(
                        CoreConstant.DATAX_JOB_SETTING_SPEED_CHANNEL);

                LOG.info("Job set Channel-Number to " + this.needChannelNumber
                        + " channels.");

                return;
            }

            throw DataXException.asDataXException(
                    FrameworkErrorCode.CONFIG_ERROR,
                    "Job 运行速度必须设置");
```

```
    }
```

8.4 调度

JobContainer.java

```
    private void schedule() {
        /**
         * 这里的全局 speed 和每个 channel 的速度设置为 B/s
         */
        int channelsPerTaskGroup = this.configuration.getInt(

CoreConstant.DATAX_CORE_CONTAINER_TASKGROUP_CHANNEL, 5);
        int taskNumber = this.configuration.getList(
                CoreConstant.DATAX_JOB_CONTENT).size();
        //确定的 channel 数和切分的 task 数取最小值，避免浪费
        this.needChannelNumber = Math.min(this.needChannelNumber, taskNumber);
        PerfTrace.getInstance().setChannelNumber(needChannelNumber);

        /**
         * 通过获取配置信息得到每个 taskGroup 需要运行哪些 tasks 任务
         */

        List<Configuration> taskGroupConfigs =
JobAssignUtil.assignFairly(this.configuration,
                this.needChannelNumber, channelsPerTaskGroup);

        LOG.info("Scheduler starts [{}] taskGroups.", taskGroupConfigs.size());

        ExecuteMode executeMode = null;
        AbstractScheduler scheduler;
        try {
            //可以看到 3.0 进行了阉割，只有 STANDALONE 模式
            executeMode = ExecuteMode.STANDALONE;
            scheduler = initStandaloneScheduler(this.configuration);

            //设置 executeMode
            for (Configuration taskGroupConfig : taskGroupConfigs) {
```

```
taskGroupConfig.set(CoreConstant.DATAX_CORE_CONTAINER_JOB_MODE,
executeMode.getValue());
                }

                if (executeMode == ExecuteMode.LOCAL || executeMode ==
ExecuteMode.DISTRIBUTE) {
                    if (this.jobId <= 0) {
                        throw
DataXException.asDataXException(FrameworkErrorCode.RUNTIME_ERROR,
                                "在[ local | distribute ]模式下必须设置 jobId，并且其
值 > 0 .");
                    }
                }

                LOG.info("Running by {} Mode.", executeMode);

                this.startTransferTimeStamp = System.currentTimeMillis();

                scheduler.schedule(taskGroupConfigs);

                this.endTransferTimeStamp = System.currentTimeMillis();
            } catch (Exception e) {
                LOG.error("运行 scheduler 模式[{}]出错.", executeMode);
                this.endTransferTimeStamp = System.currentTimeMillis();
                throw DataXException.asDataXException(
                        FrameworkErrorCode.RUNTIME_ERROR, e);
            }

            /**
             * 检查任务执行情况
             */
            this.checkLimit();
        }
```

8.4.1 确定组数和分组

assignFairly 方法：

1）确定 taskGroupNumber，

2）做分组分配，

3）做分组优化

```
    public static List<Configuration> assignFairly(Configuration configuration, int channelNumber, int channelsPerTaskGroup) {
        Validate.isTrue(configuration != null, "框架获得的 Job 不能为 null.");

        List<Configuration> contentConfig = configuration.getListConfiguration(CoreConstant.DATAX_JOB_CONTENT);
        Validate.isTrue(contentConfig.size() > 0, "框架获得的切分后的 Job 无内容.");

        Validate.isTrue(channelNumber > 0 && channelsPerTaskGroup > 0,
                "每个 channel 的平均 task 数[averTaskPerChannel]，channel 数目[channelNumber]，每个 taskGroup 的平均 channel 数[channelsPerTaskGroup]都应该为正数");

        //TODO 确定 taskgroup 的数量
        int taskGroupNumber = (int) Math.ceil(1.0 * channelNumber / channelsPerTaskGroup);

        Configuration aTaskConfig = contentConfig.get(0);

        String readerResourceMark = aTaskConfig.getString(CoreConstant.JOB_READER_PARAMETER + "." +
                CommonConstant.LOAD_BALANCE_RESOURCE_MARK);
        String writerResourceMark = aTaskConfig.getString(CoreConstant.JOB_WRITER_PARAMETER + "." +
                CommonConstant.LOAD_BALANCE_RESOURCE_MARK);

        boolean hasLoadBalanceResourceMark = StringUtils.isNotBlank(readerResourceMark) ||
                StringUtils.isNotBlank(writerResourceMark);

        if (!hasLoadBalanceResourceMark) {
            // fake 一个固定的 key 作为资源标识（在 reader 或者 writer 上均可，此处选择在 reader 上进行 fake）
```

```
                for (Configuration conf : contentConfig) {
                    conf.set(CoreConstant.JOB_READER_PARAMETER + "." +
                            CommonConstant.LOAD_BALANCE_RESOURCE_MARK,
"aFakeResourceMarkForLoadBalance");
                }
                // 是为了避免某些插件没有设置 资源标识 而进行了一次随机打乱操作
                Collections.shuffle(contentConfig, new
Random(System.currentTimeMillis()));
            }

            LinkedHashMap<String, List<Integer>> resourceMarkAndTaskIdMap =
parseAndGetResourceMarkAndTaskIdMap(contentConfig);
            List<Configuration> taskGroupConfig = doAssign(resourceMarkAndTaskIdMap,
configuration, taskGroupNumber);

            // 调整 每个 taskGroup 对应的 Channel 个数（属于优化范畴）
            adjustChannelNumPerTaskGroup(taskGroupConfig, channelNumber);
            return taskGroupConfig;
        }
```

8.4.2 调度实现

AbstractScheduler.java

```
    public void schedule(List<Configuration> configurations) {
        Validate.notNull(configurations,
                "scheduler 配置不能为空");
        int jobReportIntervalInMillSec = configurations.get(0).getInt(

CoreConstant.DATAX_CORE_CONTAINER_JOB_REPORTINTERVAL, 30000);
        int jobSleepIntervalInMillSec = configurations.get(0).getInt(
                CoreConstant.DATAX_CORE_CONTAINER_JOB_SLEEPINTERVAL,
10000);

        this.jobId = configurations.get(0).getLong(
                CoreConstant.DATAX_CORE_CONTAINER_JOB_ID);

        errorLimit = new ErrorRecordChecker(configurations.get(0));
```

```
        /**
         * 给 taskGroupContainer 的 Communication 注册
         */
        this.containerCommunicator.registerCommunication(configurations);

        int totalTasks = calculateTaskCount(configurations);
        startAllTaskGroup(configurations);

        Communication lastJobContainerCommunication = new Communication();

        long lastReportTimeStamp = System.currentTimeMillis();
        try {
            while (true) {
                /**
                 * step 1: collect job stat
                 * step 2: getReport info, then report it
                 * step 3: errorLimit do check
                 * step 4: dealSucceedStat();
                 * step 5: dealKillingStat();
                 * step 6: dealFailedStat();
                 * step 7: refresh last job stat, and then sleep for next while
                 *
                 * above steps, some ones should report info to DS
                 *
                 */
                ……
            }
        }
    ……
    }
```

ProcessInnerScheduler.java

```
    public void startAllTaskGroup(List<Configuration> configurations) {
        this.taskGroupContainerExecutorService = Executors
                .newFixedThreadPool(configurations.size());

        for (Configuration taskGroupConfiguration : configurations) {
            TaskGroupContainerRunner taskGroupContainerRunner =
newTaskGroupContainerRunner(taskGroupConfiguration);
```

```
this.taskGroupContainerExecutorService.execute(taskGroupContainerRunner);
            }

            this.taskGroupContainerExecutorService.shutdown();
        }
```

8.5 数据传输

接 8.3.2 丢到线程池执行

TaskGroupContainer.start()

-> taskExecutor.doStart()

可以看到调用插件的 start 方法

```
public void doStart() {
    this.writerThread.start();

    // reader 没有起来，writer 不可能结束
    if (!this.writerThread.isAlive() || this.taskCommunication.getState() == State.FAILED) {
        throw DataXException.asDataXException(
                FrameworkErrorCode.RUNTIME_ERROR,
                this.taskCommunication.getThrowable());
    }

    this.readerThread.start();
    ……
}
```

可以看看 generateRunner()

ReaderRunner.java

```
    public void run() {
        ……
        try {
            channelWaitWrite.start();
            ……
            initPerfRecord.start();
            taskReader.init();
            initPerfRecord.end();
```

```
                ……
                preparePerfRecord.start();
                taskReader.prepare();
                preparePerfRecord.end();

                ……
                dataPerfRecord.start();
                taskReader.startRead(recordSender);
                recordSender.terminate();

                ……
                postPerfRecord.start();
                taskReader.post();
                postPerfRecord.end();
                // automatic flush
                // super.markSuccess(); 这里不能标记为成功，成功的标志由
writerRunner 来标志（否则可能导致 reader 先结束，而 writer 还没有结束的严重
bug）
            } catch (Throwable e) {
                LOG.error("Reader runner Received Exceptions:", e);
                super.markFail(e);
            } finally {
                LOG.debug("task reader starts to do destroy ...");
                PerfRecord desPerfRecord = new PerfRecord(getTaskGroupId(), getTaskId(),
PerfRecord.PHASE.READ_TASK_DESTROY);
                desPerfRecord.start();
                super.destroy();
                desPerfRecord.end();


channelWaitWrite.end(super.getRunnerCommunication().getLongCounter(CommunicationTo
ol.WAIT_WRITER_TIME));

                long transformerUsedTime =
super.getRunnerCommunication().getLongCounter(CommunicationTool.TRANSFORMER_
USED_TIME);
                if (transformerUsedTime > 0) {
```

```
                    PerfRecord transformerRecord = new PerfRecord(getTaskGroupId(),
getTaskId(), PerfRecord.PHASE.TRANSFORMER_TIME);
                    transformerRecord.start();
                    transformerRecord.end(transformerUsedTime);
                }
            }
        }
```

8.5.1 限速的实现

比如看 MysqlReader 的 startReader 方法

-》CommonRdbmsReaderTask.startRead()

-》transportOneRecord()

-》sendToWriter()

-》BufferedRecordExchanger. flush()

-》Channel.pushAll()

-》Channel. statPush()

```
private void statPush(long recordSize, long byteSize) {

currentCommunication.increaseCounter(CommunicationTool.READ_SUCCEED_RECORD
S,
                recordSize);

currentCommunication.increaseCounter(CommunicationTool.READ_SUCCEED_BYTES,
                byteSize);
    //在读的时候进行统计 waitCounter 即可，因为写（pull）的时候可能正在阻塞，
但读的时候已经能读到这个阻塞的 counter 数

    currentCommunication.setLongCounter(CommunicationTool.WAIT_READER_TIME,
waitReaderTime);
    currentCommunication.setLongCounter(CommunicationTool.WAIT_WRITER_TIME,
waitWriterTime);

    boolean isChannelByteSpeedLimit = (this.byteSpeed > 0);
    boolean isChannelRecordSpeedLimit = (this.recordSpeed > 0);
    if (!isChannelByteSpeedLimit && !isChannelRecordSpeedLimit) {
        return;
```

```
        }

        long lastTimestamp = lastCommunication.getTimestamp();
        long nowTimestamp = System.currentTimeMillis();
        long interval = nowTimestamp - lastTimestamp;
        if (interval - this.flowControlInterval >= 0) {
            long byteLimitSleepTime = 0;
            long recordLimitSleepTime = 0;
            if (isChannelByteSpeedLimit) {
                long currentByteSpeed =
(CommunicationTool.getTotalReadBytes(currentCommunication) -
                        CommunicationTool.getTotalReadBytes(lastCommunication)) *
1000 / interval;
                if (currentByteSpeed > this.byteSpeed) {
                    // 计算根据 byteLimit 得到的休眠时间
                    byteLimitSleepTime = currentByteSpeed * interval / this.byteSpeed
                            - interval;
                }
            }

            if (isChannelRecordSpeedLimit) {
                long currentRecordSpeed =
(CommunicationTool.getTotalReadRecords(currentCommunication) -
                        CommunicationTool.getTotalReadRecords(lastCommunication)) *
1000 / interval;
                if (currentRecordSpeed > this.recordSpeed) {
                    // 计算根据 recordLimit 得到的休眠时间
                    recordLimitSleepTime = currentRecordSpeed * interval /
this.recordSpeed
                            - interval;
                }
            }

            // 休眠时间取较大值
            long sleepTime = byteLimitSleepTime < recordLimitSleepTime ?
                    recordLimitSleepTime : byteLimitSleepTime;
            if (sleepTime > 0) {
                try {
```

```
                    Thread.sleep(sleepTime);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        ……
        }
}
```

第9章 DataX 使用优化

9.1 关键参数

- job.setting.speed.channel : channel 并发数
- job.setting.speed.record : 2 全局配置 channel 的 record 限速
- job.setting.speed.byte：全局配置 channel 的 byte 限速
- core.transport.channel.speed.record：单个 channel 的 record 限速
- core.transport.channel.speed.byte：单个 channel 的 byte 限速

9.2 优化 1：提升每个 channel 的速度

在 DataX 内部对每个 Channel 会有严格的速度控制，分两种，一种是控制每秒同步的记录数，另外一种是每秒同步的字节数，默认的速度限制是 1MB/s，可以根据具体硬件情况设置这个 byte 速度或者 record 速度，一般设置 byte 速度，比如：我们可以把单个 Channel 的速度上限配置为 5MB

9.3 优化 2：提升 DataX Job 内 Channel 并发数

并发数 = taskGroup 的数量 * 每个 TaskGroup 并发执行的 Task 数 (默认为 5)。

提升 job 内 Channel 并发有三种配置方式：

9.3.1 配置全局 Byte 限速以及单 Channel Byte 限速

Channel 个数 = 全局 Byte 限速 / 单 Channel Byte 限速

```
{
    "core": {
        "transport": {
            "channel": {
                "speed": {
                    "byte": 1048576
                }
            }
        }
    },
    "job": {
        "setting": {
            "speed": {
                "byte" : 5242880
            }
        },
        ...
    }
}
```

core.transport.channel.speed.byte=1048576，job.setting.speed.byte=5242880，所以 Channel 个数 = 全局 Byte 限速 / 单 Channel Byte 限速=5242880/1048576=5 个

9.3.2 配置全局 Record 限速以及单 Channel Record 限速

Channel 个数 = 全局 Record 限速 / 单 Channel Record 限速

```
{
    "core": {
        "transport": {
            "channel": {
                "speed": {
                    "record": 100
                }
            }
        }
    },
    "job": {
        "setting": {
            "speed": {
```

```
                "record" : 500
            }
        },
        ...
    }
}
```

core.transport.channel.speed.record=100 ， job.setting.speed.record=500, 所以配置全局 Record 限速以及单 Channel Record 限速，Channel 个数 = 全局 Record 限速 / 单 Channel Record 限速=500/100=5

9.3.3 直接配置 Channel 个数

只有在上面两种未设置才生效，上面两个同时设置是取值小的作为最终的 channel 数。

```
{
    "job": {
        "setting": {
            "speed": {
                "channel" : 5
            }
        },
        ...
    }
}
```

直接配置 job.setting.speed.channel=5，所以 job 内 Channel 并发=5 个

9.4 优化 3：提高 JVM 堆内存

当提升 DataX Job 内 Channel 并发数时，内存的占用会显著增加，因为 DataX 作为数据交换通道，在内存中会缓存较多的数据。例如 Channel 中会有一个 Buffer，作为临时的数据交换的缓冲区，而在部分 Reader 和 Writer 的中，也会存在一些 Buffer，为了防止 OOM 等错误，调大 JVM 的堆内存。

建议将内存设置为 4G 或者 8G，这个也可以根据实际情况来调整。

调整 JVM xms xmx 参数的两种方式：一种是直接更改 datax.py 脚本；另一种是在启动的时候，加上对应的参数，如下：

```
python datax/bin/datax.py --jvm="-Xms8G -Xmx8G" XXX.json
```