Addax

发布 *4.0.2*

unknown

2021 年 08 月 06 日

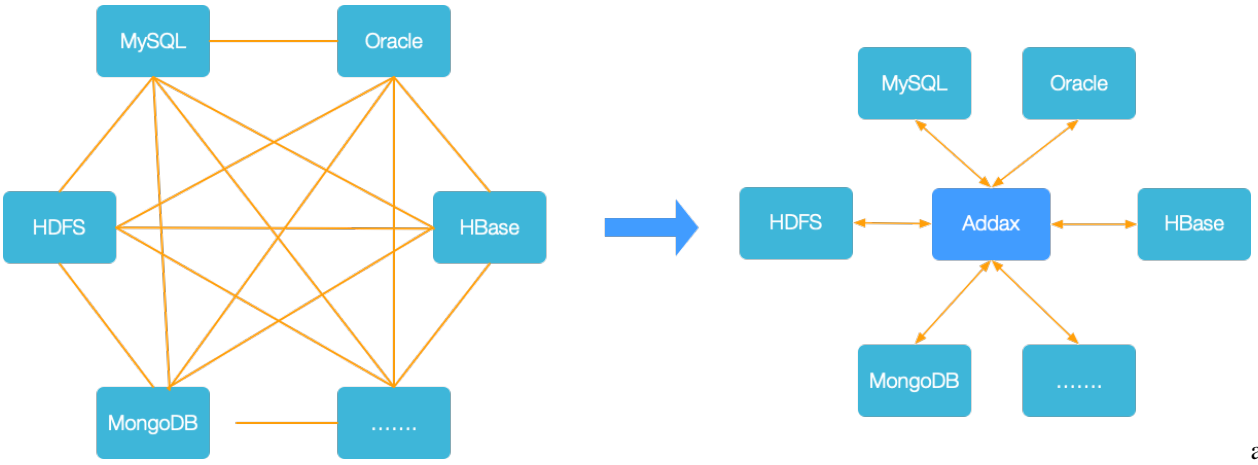
Contents

CHAPTER 1

Addax 介绍

1.1 一、Addax 概览

Addax 是一个异构数据源离线同步工具（最初来源于阿里的 DataX），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。



addax_why_new

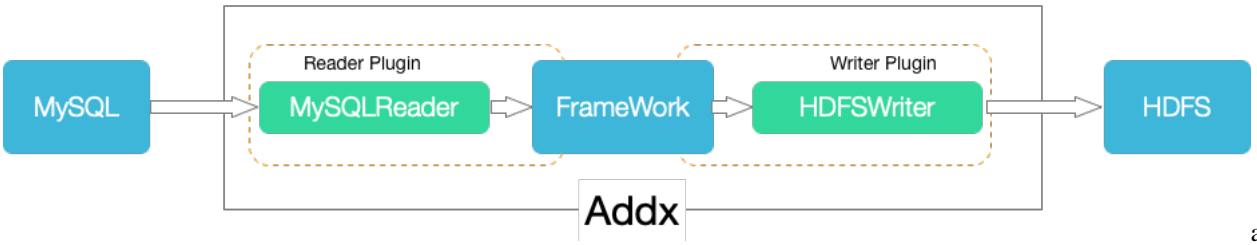
1.1.1 设计理念

为了解决异构数据源同步问题，Addax 将复杂的网状的同步链路变成了星型数据链路，Addax 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 Addax，便能跟已有的数据源做到无缝数据同步。

1.1.2 当前使用现状

Addax 在阿里巴巴集团内被广泛使用，承担了所有大数据的离线同步业务，并已持续稳定运行了 6 年之久。目前每天完成同步 8w 多道作业，每日传输数据量超过 300TB。

1.2 二、Addax 框架设计



addax_framework_

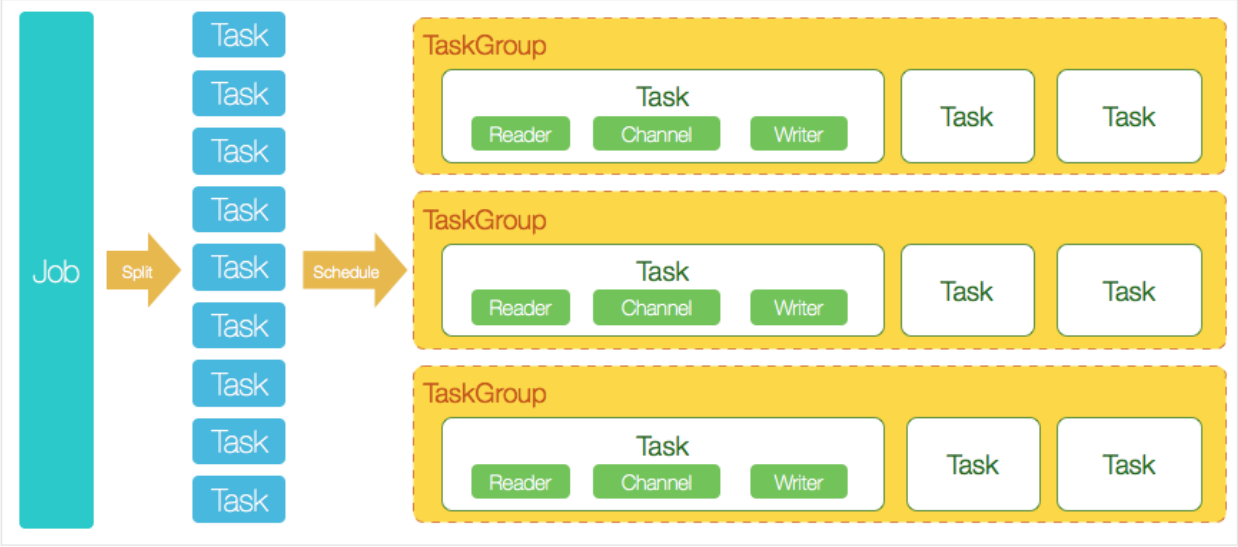
Addax 本身作为离线数据同步框架，采用 Framework + plugin 架构构建。将数据源读取和写入抽象成为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader：Reader 为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer：Writer 为数据写入模块，负责不断向 Framework 取数据，并将数据写入到目的端。
- Framework：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。

Addax Framework 提供了简单的接口与插件交互，提供简单的插件接入机制，只需要任意加上一种插件，就能无缝对接其他数据源。

1.3 四、Addax 核心架构

Addax 3.0 开源版本支持单机多线程模式完成同步作业运行，本小节按一个 Addax 作业生命周期的时序图，从整体架构设计非常简要说明 Addax 各个模块相互关系。



addax_arch

1.3.1 核心模块介绍：

1. Addax 完成单个数据同步的作业，我们称之为 Job，Addax 接受到一个 Job 之后，将启动一个进程来完成整个作业同步过程。Addax Job 模块是单个作业的中枢管理节点，承担了数据清理、子任务切分 (将单一作业计算转化为多个子 Task)、TaskGroup 管理等功能。
2. AddaxJob 启动后，会根据不同的源端切分策略，将 Job 切分成多个小的 Task(子任务)，以便于并发执行。Task 便是 Addax 作业的最小单元，每一个 Task 都会负责一部分数据的同步工作。
3. 切分多个 Task 之后，Addax Job 会调用 Scheduler 模块，根据配置的并发数据量，将拆分成的 Task 重新组合，组装成 TaskGroup(任务组)。每一个 TaskGroup 负责以一定的并发运行完毕分配好的所有 Task，默认单个任务组的并发数量为 5。
4. 每一个 Task 都由 TaskGroup 负责启动，Task 启动后，会固定启动 `Reader—>Channel—>Writer` 的线程来完成任务同步工作。
5. Addax 作业运行起来之后，Job 监控并等待多个 TaskGroup 模块任务完成，等待所有 TaskGroup 任务完成后 Job 成功退出。否则，异常退出，进程退出值非 0

1.3.2 Addax 调度流程：

举例来说，用户提交了一个 Addax 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。Addax 的调度决策思路是：

1. AddaxJob 根据分库分表切分成了 100 个 Task。
2. 根据 20 个并发，Addax 计算共需要分配 20/5 = 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。

1.4 五、Addax 核心优势

1.4.1 可靠的数据质量监控

- 完美解决数据传输个别类型失真问题

  Addax 旧版对于部分数据类型 (比如时间戳) 传输一直存在毫秒阶段等数据失真情况，新版本 Addax3.0 已经做到支持所有的强数据类型，每一种插件都有自己的数据类型转换策略，让数据可以完整无损的传输到目的端。

- 提供作业全链路的流量、数据量运行时监控

  Addax3.0 运行过程中可以将作业本身状态、数据流量、数据速度、执行进度等信息进行全面的展示，让用户可以实时了解作业状态。并可在作业执行过程中智能判断源端和目的端的速度对比情况，给予用户更多性能排查信息。

- 提供脏数据探测

  在大量数据的传输过程中，必定会由于各种原因导致很多数据传输报错 (比如类型转换错误)，这种数据 Addax 认为就是脏数据。Addax 目前可以实现脏数据精确过滤、识别、采集、展示，为用户提供多种的脏数据处理模式，让用户准确把控数据质量大关！

1.4.2 丰富的数据转换功能

Addax 作为一个服务于大数据的 ETL 工具，除了提供数据快照搬迁功能之外，还提供了丰富数据转换的功能，让数据在传输过程中可以轻松完成数据脱敏，补全，过滤等数据转换功能，另外还提供了自动 groovy 函数，让用户自定义转换函数。详情请看 Addax3 的 transformer 详细介绍。

1.4.3 精准的速度控制

新版本 Addax3.0 提供了包括通道 (并发)、记录流、字节流三种流控模式，可以随意控制你的作业速度，让你的作业在库可以承受的范围内达到最佳的同步速度。

```
{
  "speed": {
    "channel": 5,
    "byte": 1048576,
    "record": 10000
  }
}
```

强劲地同步性能

Addax 每一种读插件都有一种或多种切分策略，都能将作业合理切分成多个 Task 并行执行，单机多线程执行模型可以让 Addax 速度随并发成线性增长。在源端和目的端性能都足够的情况下，单个作业一定可以打满网卡。另外，Addax 团队对所有的已经接入的插件都做了极致的性能优化，并且做了完整的性能测试。

健壮的容错机制

Addax 作业是极易受外部因素的干扰，网络闪断、数据源不稳定等因素很容易让同步到一半的作业报错停止。因此稳定性是 Addax 的基本要求，在 Addax 3.0 的设计中，重点完善了框架和插件的稳定性。目前 Addax3.0 可以做到线程级别、进程级别 (暂时未开放)、作业级别多层次局部/全局的重试，保证用户的作业稳定运行。

- 线程内部重试

  Addax 的核心插件都经过团队的全盘 review，不同的网络交互方式都有不同的重试策略。

- 线程级别重试

  目前 Addax 已经可以实现 TaskFailover，针对于中间失败的 Task，Addax 框架可以做到整个 Task 级别的重新调度。

CHAPTER 2

读取插件

本章描述 Addax 目前支持的数据读取插件

2.1 Cassandra Reader

`CassandraReader` 插件实现了从Cassandra 读取数据。

2.1.1 配置

下面是配置一个从 Cassandra 读取数据到终端的例子

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "cassandrareader",
          "parameter": {
            "host": "localhost",
            "port": 9042,
            "useSSL": false,
            "keyspace": "test",
            "table": "addax_src",
            "column": [
              "textCol",
```

(下页继续)

(续上页)

```
                "blobCol",
                "writetime(blobCol)",
                "boolCol",
                "smallintCol",
                "tinyintCol",
                "intCol",
                "bigintCol",
                "varintCol",
                "floatCol",
                "doubleCol",
                "decimalCol",
                "dateCol",
                "timeCol",
                "timeStampCol",
                "uuidCol",
                "inetCol",
                "durationCol",
                "listCol",
                "mapCol",
                "setCol",
                "tupleCol",
                "udtCol"
              ]
            }
          },
          "writer": {
            "name": "streamwriter",
            "parameter": {
              "print": true
            }
          }
        }
      ]
    }
}
```

2.1.2 参数说明

parameter 配置项支持以下配置

支持的数据类型

目前支持除 `counter` 和 `Custom` 类型之外的所有类型。

下面列出 `CassandraReader` 针对 `Cassandra` 类型转换列表:

2.2 ClickHouse Reader

`ClickHouseReader` 插件支持从 ClickHouse数据库读取数据。

2.2.1 示例

表结构及数据信息

假定需要的读取的表的结构以及数据如下:

```
CREATE TABLE ck_addax (
    c_int8 Int8,
    c_int16 Int16,
    c_int32 Int32,
    c_int64 Int64,
    c_uint8 UInt8,
    c_uint16 UInt16,
    c_uint32 UInt32,
    c_uint64 UInt64,
    c_float32 Float32,
    c_float64 Float64,
    c_decimal Decimal(38,10),
    c_string String,
    c_fixstr FixedString(36),
    c_uuid UUID,
    c_date Date,
    c_datetime DateTime('Asia/Chongqing'),
    c_datetime64 DateTime64(3, 'Asia/Chongqing'),
    c_enum Enum('hello' = 1, 'world'=2)
) ENGINE = MergeTree() ORDER BY (c_int8, c_int16) SETTINGS index_granularity = 8192;

insert into ck_addax values(
    127,
    -32768,
    2147483647,
    -9223372036854775808,
    255,
    65535,
    4294967295,
    18446744073709551615,
    0.999999999999,
    0.99999999999999999,
    123456789123456789123456789.1234567891,
    'Hello String',
    '2c:16:db:a3:3a:4f',
    '5F042A36-5B0C-4F71-ADFD-4DF4FCA1B863',
    '2021-01-01',
    '2021-01-01 00:00:00',
```

(下页继续)

(续上页)

```
    '2021-01-01 00:00:00',
    'hello'
);
```

2.2.2 配置 json 文件

下面的配置文件表示从 ClickHouse 数据库读取指定的表数据并打印到终端

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [
      {
        "reader": {
          "name": "clickhousereader",
          "parameter": {
            "username": "root",
            "password": "root",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "ck_addax"
                ],
                "jdbcUrl": [
                  "jdbc:clickhouse://127.0.0.1:8123/default"
                ]
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/clickhouse2stream.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/clickhouse2stream.json
```

其输出信息如下（删除了非关键信息）

```
2021-01-06 14:39:35.742 [main] INFO  VMInfo - VMInfo# operatingSystem class => com.
↪sun.management.internal.OperatingSystemImpl

2021-01-06 14:39:35.767 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "column":[
                                                "*"
                                        ],
                                        "connection":[
                                                {
                                                        "jdbcUrl":[
                                                                "jdbc:clickhouse://
↪127.0.0.1:8123/"
                                                        ],
                                                        "table":[
                                                                "ck_addax"
                                                        ]
                                                }
                                        ],
                                        "username":"default"
                                },
                                "name":"clickhousereader"
                        },
                        "writer":{
                                "parameter":{
                                        "print":true
                                },
                                "name":"streamwriter"
                        }
                }
        ],
        "setting":{
                "errorLimit":{
                        "record":0,
                        "percentage":0.02
                },
                "speed":{
                        "channel":3
                }
        }
}

127        -32768        2147483647        -
↪9223372036854775808        255        65535        4294967295        18446744073709551615        1
↪1234567891Hello String        2c:16:db:a3:3a:4f
5f042a36-5b0c-4f71-adfd-4df4fca1b863        2021-01-01        2021-01-01␣
↪00:00:00        2021-01-01 00:00:00        hello
```

(下页继续)

(续上页)

```
任务启动时刻                    : 2021-01-06 14:39:35
任务结束时刻                    : 2021-01-06 14:39:39
任务总计耗时                    :                  3s
任务平均流量                    :               77B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   1
读写失败总数                    :                   0
```

2.2.3 参数说明

`parameter` 配置项支持以下配置

2.2.4 支持的数据类型

目前 ClickHouseReader 支持大部分 ClickHouse 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 ClickHouseReader 针对 ClickHouse 类型转换列表:

2.2.5 限制

除上述罗列字段类型外，其他类型均不支持，如 Array、Nested 等

2.3 DbfFile Reader

`DbfFileReader` 插件支持读取 DBF 格式文件

2.3.1 配置说明

以下是读取 DBF 文件后打印到终端的配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "dbffilereader",
          "parameter": {
            "column": [
              {
                "index": 0,
```

(下页继续)

(续上页)

```
                "type": "string"
              },
              {
                "index": 1,
                "type": "long"
              },
              {
                "index": 2,
                "type": "string"
              },
              {
                "index": 3,
                "type": "boolean"
              },
              {
                "index": 4,
                "type": "string"
              },
              {
                "value": "dbf",
                "type": "string"
              }
            ],
            "path": [
              "/tmp/out"
            ],
            "encoding": "GBK"
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": "true"
          }
        }
      }
    ]
  }
}
```

2.3.2 参数说明

parameter 配置项支持以下配置

path

描述：本地文件系统的路径信息，注意这里可以支持填写多个路径。

- 当指定单个本地文件，DbfFileReader 暂时只能使用单线程进行数据抽取。二期考虑在非压缩文件情况下针对单个 File 可以进行多线程并发读取。
- 当指定多个本地文件，DbfFileReader 支持使用多线程进行数据抽取。线程并发数通过通道数指定。
- 当指定通配符，DbfFileReader 尝试遍历出多个文件信息。例如: 指定 `/*` 代表读取/目录下所有的文件，指定 `/bazhen/*` 代表读取 bazhen 目录下游所有的文件。dbfFileReader 目前只支持 `*` 作为文件通配符。

特别需要注意的是，Addax 会将一个作业下同步的所有 dbf File 视作同一张数据表。用户必须自己保证所有的 File 能够适配同一套 schema 信息。读取文件用户必须保证为类 dbf 格式，并且提供给 Addax 权限可读。

特别需要注意的是，如果 Path 指定的路径下没有符合匹配的文件抽取，Addax 将报错。

column

读取字段列表，type 指定源数据的类型，name 为字段名, 长度最大 8，value 指定当前类型为常量，不从源头文件读取数据，而是根据 value 值自动生成对应的列。

默认情况下，用户可以全部按照 String 类型读取数据，配置如下：

```
"column": ["*"]
```

用户可以指定 Column 字段信息，配置如下：

```
{
  "type": "long",
  "index": 0
},
{
"type": "string",
"value": "alibaba"
}
```

`index: 0` 表示从本地 DBF 文件第一列获取 int 字段 `value: alibaba` 表示从 dbfFileReader 内部生成 alibaba 的字符串字段作为当前字段对于用户指定 Column 信息，type 必须填写，index/value 必须选择其一。

支持的数据类型

本地文件本身提供数据类型，该类型是 Addax dbfFileReader 定义：

其中：

- Long 是指本地文件文本中使用整形的字符串表示形式，例如”19901219”。
- Double 是指本地文件文本中使用 Double 的字符串表示形式，例如”3.1415”。
- Boolean 是指本地文件文本中使用 Boolean 的字符串表示形式，例如”true”、”false”。不区分大小写。
- Date 是指本地文件文本中使用 Date 的字符串表示形式，例如”2014-12-31”，Date 可以指定 format 格式。

2.4 ElasticSearchReader

ElasticSearchReader 插件实现了从 Elasticsearch 读取索引的功能，它通过 Elasticsearch 提供的 Rest API （默认端口 9200），执行指定的查询语句批量获取数据

2.4.1 示例

假定要获取的索引内容如下

```
{
"took": 14,
"timed_out": false,
"_shards": {
"total": 1,
"successful": 1,
"skipped": 0,
"failed": 0
},
"hits": {
"total": 2,
"max_score": 1,
"hits": [
{
"_index": "test-1",
"_type": "default",
"_id": "38",
"_score": 1,
"_source": {
"col_date": "2017-05-25T11:22:33.000+08:00",
"col_integer": 19890604,
"col_keyword": "hello world",
"col_ip": "1.1.1.1",
"col_text": "long text",
"col_double": 19890604,
"col_long": 19890604,
"col_geo_point": "41.12,-71.34"
}
},
{
"_index": "test-1",
"_type": "default",
"_id": "103",
"_score": 1,
"_source": {
"col_date": "2017-05-25T11:22:33.000+08:00",
"col_integer": 19890604,
"col_keyword": "hello world",
"col_ip": "1.1.1.1",
"col_text": "long text",
"col_double": 19890604,
"col_long": 19890604,
"col_geo_point": "41.12,-71.34"
}
}
]
```

(下页继续)

(续上页)

```
}
}
```

配置一个从 Elasticsearch 读取数据并打印到终端的任务

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": -1,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "elasticsearchreader",
          "parameter": {
            "endpoint": "http://127.0.0.1:9200",
            "accessId":"",
            "accesskey":"",
            "index": "test-1",
            "type": "default",
            "searchType": "dfs_query_then_fetch",
            "headers": {},
            "scroll": "3m",
            "search": [
              {
                "query": {
                  "match": {
                    "col_ip": "1.1.1.1"
                  }
                },
                "aggregations": {
                  "top_10_states": {
                    "terms": {
                      "field": "col_date",
                      "size": 10
                    }
                  }
                }
              }
            ],
            "column": ["col_ip", "col_double", "col_long","col_integer",
              "col_keyword", "col_text","col_geo_point","col_date"
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true,
            "encoding": "UTF-8"
          }
        }
      }
```

(下页继续)

(续上页)

```
    ]
  }
}
```

将上述内容保存为 job/es2stream.json

执行下面的命令进行采集

```
bin/addax.sh job/es2stream.json
```

其输出结果类似如下（输出记录数有删减）

```
2021-02-19 13:38:15.860 [main] INFO  VMInfo - VMInfo# operatingSystem class => com.
↪sun.management.internal.OperatingSystemImpl
2021-02-19 13:38:15.895 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "accessId":"",
                                        "headers":{},
                                        "endpoint":"http://127.0.0.1:9200",
                                        "search":[
                    {
                      "query": {
                        "match": {
                          "col_ip": "1.1.1.1"
                        }
                      },
                      "aggregations": {
                        "top_10_states": {
                          "terms": {
                            "field": "col_date",
                            "size": 10
                          }
                        }
                      }
                    }
                                        ],
                                        "accesskey":"*****",
                                        "searchType":"dfs_query_then_fetch",
                                        "scroll":"3m",
                                        "column":[
                                                "col_ip",
                                                "col_double",
                                                "col_long",
                                                "col_integer",
                                                "col_keyword",
                                                "col_text",
                                                "col_geo_point",
                                                "col_date"
                                        ],
                                        "index":"test-1",
                                        "type":"default"
                                },
                                "name":"elasticsearchreader"
```

(下页继续)

(续上页)

```
                        },
                        "writer":{
                                "parameter":{
                                        "print":true,
                                        "encoding":"UTF-8"
                                },
                                "name":"streamwriter"
                        }
                }
        ],
        "setting":{
                "errorLimit":{
                        "record":0,
                        "percentage":0.02
                },
                "speed":{
                        "byte":-1,
                        "channel":1
                }
        }
}

2021-02-19 13:38:15.934 [main] INFO  PerfTrace - PerfTrace traceId=job_-1,␣
↪isEnable=false, priority=0
2021-02-19 13:38:15.934 [main] INFO  JobContainer - Addax jobContainer starts job.
2021-02-19 13:38:15.937 [main] INFO  JobContainer - Set jobId = 0

2017-05-25T11:22:33.000+08:00        19890604        hello world        1.1.1.
↪1        long text        19890604        19890604        41.12,-71.34
2017-05-25T11:22:33.000+08:00        19890604        hello world        1.1.1.
↪1        long text        19890604        19890604        41.12,-71.34

2021-02-19 13:38:19.845 [job-0] INFO  AbstractScheduler - Scheduler accomplished all␣
↪tasks.
2021-02-19 13:38:19.848 [job-0] INFO  JobContainer - Addax Writer.Job [streamwriter]␣
↪do post work.
2021-02-19 13:38:19.849 [job-0] INFO  JobContainer - Addax Reader.Job␣
↪[elasticsearchreader] do post work.
2021-02-19 13:38:19.855 [job-0] INFO  JobContainer - PerfTrace not enable!
2021-02-19 13:38:19.858 [job-0] INFO  StandAloneJobContainerCommunicator - Total 95␣
↪records, 8740 bytes | Speed 2.84KB/s, 31 records/s | Error 0 records, 0 bytes | ␣
↪All Task WaitWriterTime 0.000s |  All Task WaitReaderTime 0.103s | Percentage 100.00
↪%
2021-02-19 13:38:19.861 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-02-19 13:38:15
任务结束时刻                    : 2021-02-19 13:38:19
任务总计耗时                    :                  3s
任务平均流量                    :            2.84KB/s
记录写入速度                    :             31rec/s
读出记录总数                    :                   2
读写失败总数                    :                   0
```

2.4.2 参数说明

search

search 配置项允许配置为满足 Elasticsearch API 查询要求的内容，比如这样：

```
{
  "query": {
    "match": {
      "message": "myProduct"
    }
  },
  "aggregations": {
    "top_10_states": {
      "terms": {
        "field": "state",
        "size": 10
      }
    }
  }
}
```

searchType

searchType 目前支持以下几种：

- dfs_query_then_fetch
- query_then_fetch
- count
- scan

2.5 Addax FtpReader 说明

2.5.1 1 快速介绍

FtpReader 提供了读取远程 FTP 文件系统数据存储的能力。在底层实现上，FtpReader 获取远程 FTP 文件数据，并转换为 Addax 传输协议传递给 Writer。

本地文件内容存放的是一张逻辑意义上的二维表，例如 CSV 格式的文本信息。

2.5.2 2 功能与限制

FtpReader 实现了从远程 FTP 文件读取数据并转为 Addax 协议的功能，远程 FTP 文件本身是无结构化数据存储，对于 Addax 而言，FtpReader 实现上类比 TxtFileReader，有诸多相似之处。目前 FtpReader 支持功能如下：

1. 支持且仅支持读取 TXT 的文件，且要求 TXT 中 shema 为一张二维表。
2. 支持类 CSV 格式文件，自定义分隔符。
3. 支持多种类型数据读取 (使用 String 表示)，支持列裁剪，支持列常量
4. 支持递归读取、支持文件名过滤。

5. 支持文本压缩，现有压缩格式为 zip、gzip、bzip2。
6. 多个 File 可以支持并发读取。

我们暂时不能做到：

1. 单个 File 支持多线程并发读取，这里涉及到单个 File 内部切分算法。二期考虑支持。
2. 单个 File 在压缩情况下，从技术上无法支持多线程并发读取。

2.5.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "ftpreader",
          "parameter": {
            "protocol": "sftp",
            "host": "127.0.0.1",
            "port": 22,
            "username": "xx",
            "password": "xxx",
            "path": [
              "/var/pub/ftpReaderTest/data"
            ],
            "column": [
              {
                "index": 0,
                "type": "long"
              },
              {
                "index": 1,
                "type": "boolean"
              },
              {
                "index": 2,
                "type": "double"
              },
              {
                "index": 3,
                "type": "string"
              },
              {
                "index": 4,
                "type": "date",
                "format": "yyyy.MM.dd"
              }
```

(下页继续)

(续上页)

```
            ],
            "encoding": "UTF-8",
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "ftpWriter",
          "parameter": {
            "path": "/var/ftp/FtpWriter/result",
            "fileName": "shihf",
            "writeMode": "truncate",
            "format": "yyyy-MM-dd"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

path

远程 FTP 文件系统的路径信息，注意这里可以支持填写多个路径。

- 当指定单个远程 FTP 文件，FtpReader 暂时只能使用单线程进行数据抽取。二期考虑在非压缩文件情况下针对单个 File 可以进行多线程并发读取 = 当指定多个远程 FTP 文件，FtpReader 支持使用多线程进行数据抽取。线程并发数通过通道数指定
- 当指定通配符，FtpReader 尝试遍历出多个文件信息。例如: 指定 `/*` 代表读取/目录下所有的文件，指定 `/bazhen/*` 代表读取 bazhen 目录下游所有的文件。目前只支持 `*` 作为文件通配符。

特别需要注意的是，Addax 会将一个作业下同步的所有 Text File 视作同一张数据表。用户必须自己保证所有的 File 能够适配同一套 schema 信息。读取文件用户必须保证为类 CSV 格式，并且提供给 Addax 权限可读。特别需要注意的是，如果 Path 指定的路径下没有符合匹配的文件抽取，Addax 将报错。

column

读取字段列表，type 指定源数据的类型，index 指定当前列来自于文本第几列 (以 0 开始)，value 指定当前类型为常量，不从源头文件读取数据，而是根据 value 值自动生成对应的列。

默认情况下，用户可以全部按照 String 类型读取数据，配置如下：

```
    "column": ["*"]
```

用户可以指定 Column 字段信息，配置如下：

```
{
  "type": "long",
  "index": 0
  //从远程 FTP 文件文本第一列获取 int 字段
},
{
"type": "string",
```

(下页继续)

(续上页)

```
"value": "alibaba"  //从 FtpReader 内部生成 alibaba 的字符串字段作为当前字段
}
```

对于用户指定 Column 信息，type 必须填写，index/value 必须选择其一。

csvReaderConfig

常见配置：

```
"csvReaderConfig":{
"safetySwitch": false,
"skipEmptyRecords": false,
"useTextQualifier": false
}
```

所有配置项及默认值, 配置时 csvReaderConfig 的 map 中请**严格按照以下字段名字进行配置**：

```
boolean caseSensitive = true;
char textQualifier = 34;
boolean trimWhitespace = true;
boolean useTextQualifier = true;//是否使用 csv 转义字符
char delimiter = 44;//分隔符
char recordDelimiter = 0;
char comment = 35;
boolean useComments = false;
int escapeMode = 1;
boolean safetySwitch = true;//单列长度是否限制 100000 字符
boolean skipEmptyRecords = true;//是否跳过空行
boolean captureRawRecord = true;
```

3.3 类型转换

远程 FTP 文件本身不提供数据类型，该类型是 Addax FtpReader 定义：

其中：

- Long 是指远程 FTP 文件文本中使用整形的字符串表示形式，例如”19901219”。
- Double 是指远程 FTP 文件文本中使用 Double 的字符串表示形式，例如”3.1415”。
- Boolean 是指远程 FTP 文件文本中使用 Boolean 的字符串表示形式，例如”true”、”false”。不区分大小写。
- Date 是指远程 FTP 文件文本中使用 Date 的字符串表示形式，例如”2014-12-31”，Date 可以指定 format 格式。

2.6 Hbase11XReader 插件文档

2.6.1 1 快速介绍

HbaseReader 插件实现了从 Hbase 中读取数据。在底层实现上，HbaseReader 通过 HBase 的 Java 客户端连接远程 HBase 服务，并通过 Scan 方式读取你指定 rowkey 范围内的数据，并将读取的数据使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

以下演示基于下面创建的表以及数据

```
create 'users', 'address','info'
put 'users', 'lisi', 'address:country', 'china'
put 'users', 'lisi', 'address:province',    'beijing'
put 'users', 'lisi', 'info:age',        27
put 'users', 'lisi', 'info:birthday',   '1987-06-17'
put 'users', 'lisi', 'info:company',    'baidu'
put 'users', 'xiaoming', 'address:city',    'hangzhou'
put 'users', 'xiaoming', 'address:country', 'china'
put 'users', 'xiaoming', 'address:province',    'zhejiang'
put 'users', 'xiaoming', 'info:age',        29
put 'users', 'xiaoming', 'info:birthday',   '1987-06-17'
put 'users', 'xiaoming', 'info:company',    'alibaba'
```

1.1 支持模式

目前 HbaseReader 支持两模式读取：normal 模式、multiVersionFixedColumn 模式；

normal 模式

把 HBase 中的表，当成普通二维表（横表）进行读取, 读取最新版本数据。如：

```
hbase(main):017:0> scan 'users'
ROW           COLUMN+CELL
 lisi         column=address:city, timestamp=1457101972764, value=beijing
 lisi         column=address:country, timestamp=1457102773908, value=china
 lisi         column=address:province, timestamp=1457101972736, value=beijing
 lisi         column=info:age, timestamp=1457101972548, value=27
 lisi         column=info:birthday, timestamp=1457101972604, value=1987-06-17
 lisi         column=info:company, timestamp=1457101972653, value=baidu
 xiaoming     column=address:city, timestamp=1457082196082, value=hangzhou
 xiaoming     column=address:country, timestamp=1457082195729, value=china
 xiaoming     column=address:province, timestamp=1457082195773, value=zhejiang
 xiaoming     column=info:age, timestamp=1457082218735, value=29
 xiaoming     column=info:birthday, timestamp=1457082186830, value=1987-06-17
 xiaoming     column=info:company, timestamp=1457082189826, value=alibaba
2 row(s) in 0.0580 seconds
```

读取后数据

multiVersionFixedColumn 模式

把 HBase 中的表，当成竖表进行读取。读出的每条记录一定是四列形式，依次为：rowKey，family:qualifier，timestamp，value。

读取时需要明确指定要读取的列，把每一个 cell 中的值，作为一条记录（record），若有多个版本就有多条记录（record）。如：

```
hbase(main):018:0> scan 'users',{VERSIONS=>5}
ROW                                   COLUMN+CELL
 lisi                                 column=address:city, timestamp=1457101972764,␣
↪value=beijing
 lisi                                 column=address:contry, timestamp=1457102773908,␣
↪value=china
 lisi                                 column=address:province,␣
↪timestamp=1457101972736, value=beijing
 lisi                                 column=info:age, timestamp=1457101972548,␣
↪value=27
 lisi                                 column=info:birthday, timestamp=1457101972604,␣
↪value=1987-06-17
 lisi                                 column=info:company, timestamp=1457101972653,␣
↪value=baidu
 xiaoming                             column=address:city, timestamp=1457082196082,␣
↪value=hangzhou
 xiaoming                             column=address:contry, timestamp=1457082195729,␣
↪value=china
 xiaoming                             column=address:province,␣
↪timestamp=1457082195773, value=zhejiang
 xiaoming                             column=info:age, timestamp=1457082218735,␣
↪value=29
 xiaoming                             column=info:age, timestamp=1457082178630,␣
↪value=24
 xiaoming                             column=info:birthday, timestamp=1457082186830,␣
↪value=1987-06-17
 xiaoming                             column=info:company, timestamp=1457082189826,␣
↪value=alibaba
2 row(s) in 0.0260 seconds
```

读取后数据 (4 列)

1.2 限制

1. 目前不支持动态列的读取。考虑网络传输流量（支持动态列，需要先将 hbase 所有列的数据读取出来，再按规则进行过滤），现支持的两种读取模式中需要用户明确指定要读取的列。
2. 关于同步作业的切分：目前的切分方式是根据用户 hbase 表数据的 region 分布进行切分。即：在用户填写的 [startrowkey，endrowkey] 范围内，一个 region 会切分成一个 task，单个 region 不进行切分。
3. multiVersionFixedColumn 模式下不支持增加常量列

2.6.2 2 实现原理

简而言之，HbaseReader 通过 HBase 的 Java 客户端，通过 HTable, Scan, ResultScanner 等 API，读取你指定 rowkey 范围内的数据，并将读取的数据使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。hbase11xreader 与 hbase094xreader 的主要不同在于 API 的调用不同，Hbase1.1.x 废弃了很多 Hbase0.94.x 的 api。

2.6.3 3 功能说明

3.1 配置样例

配置一个从 HBase 抽取数据到本地的作业:（normal 模式）

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "hbase11xreader",
                    "parameter": {
                        "hbaseConfig": {
                            "hbase.zookeeper.quorum": "xxxf"
                        },
                        "table": "users",
                        "encoding": "utf-8",
                        "mode": "normal",
                        "column": [
                            {
                                "name": "rowkey",
                                "type": "string"
                            },
                            {
                                "name": "info: age",
                                "type": "string"
                            },
                            {
                                "name": "info: birthday",
                                "type": "date",
                                "format":"yyyy-MM-dd"
                            },
                            {
                                "name": "info: company",
                                "type": "string"
                            },
                            {
                                "name": "address: country",
                                "type": "string"
                            },
                            {
                                "name": "address: province",
                                "type": "string"
```

(下页继续)

(续上页)

```
                            },
                            {
                                "name": "address: city",
                                "type": "string"
                            }
                        ],
                        "range": {
                            "startRowkey": "",
                            "endRowkey": "",
                            "isBinaryRowkey": true
                        }
                    }
                },
                "writer": {
                    "name": "txtfilewriter",
                    "parameter": {
                        "path": "/Users/shf/workplace/addax_test/hbase11xreader/result
↪",
                        "fileName": "qiran",
                        "writeMode": "truncate"
                    }
                }
            }
        ]
    }
}
```

配置一个从 HBase 抽取数据到本地的作业:（multiVersionFixedColumn 模式）

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
       "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "hbase11xreader",
          "parameter": {
            "hbaseConfig": {
              "hbase.zookeeper.quorum": "127.0.0.1:2181"
            },
            "table": "users",
            "encoding": "utf-8",
            "mode": "multiVersionFixedColumn",
            "maxVersion": "-1",
            "column": [
                {
                    "name": "rowkey",
                    "type": "string"
                },
                {
                    "name": "info: age",
```

(下页继续)

(续上页)

```
                    "type": "string"
                },
                {
                    "name": "info: birthday",
                    "type": "date",
                    "format":"yyyy-MM-dd"
                },
                {
                    "name": "info: company",
                    "type": "string"
                },
                {
                    "name": "address: contry",
                    "type": "string"
                },
                {
                    "name": "address: province",
                    "type": "string"
                },
                {
                    "name": "address: city",
                    "type": "string"
                }
            ],
            "range": {
              "startRowkey": "",
              "endRowkey": ""
            }
          }
        },
        "writer": {
          "name": "txtfilewriter",
          "parameter": {
            "path": "/Users/shf/workplace/addax_test/hbase11xreader/result",
            "fileName": "qiran",
            "writeMode": "truncate"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

描述：要读取的 hbase 字段，normal 模式与 multiVersionFixedColumn 模式下必填项。

normal 模式

name 指定读取的 hbase 列，除了 rowkey 外，必须为列族: 列名的格式，type 指定源数据的类型，format 指定日期类型的格式，value 指定当前类型为常量，不从 hbase 读取数据，而是根据 value 值自动生成对应的列。配置格式如下：

```
"column":
[
```

(下页继续)

(续上页)

```
    {
        "name": "rowkey",
        "type": "string"
    },
    {
        "value": "test",
        "type": "string"
    }
]
```

normal 模式下，对于用户指定 Column 信息，type 必须填写，name/value 必须选择其一。

multiVersionFixedColumn 模式

name 指定读取的 hbase 列，除了 rowkey 外，必须为列族: 列名的格式，type 指定源数据的类型，format 指定日期类型的格式。multiVersionFixedColumn 模式下不支持常量列。配置格式如下：

```
"column": [
    {
        "name": "rowkey",
        "type": "string"
    },
    {
        "name": "info: age",
        "type": "string"
    }
]
```

range

指定 hbasereader 读取的 rowkey 范围

- startRowkey：指定开始 rowkey
- endRowkey 指定结束 rowkey
- isBinaryRowkey：指定配置的 startRowkey 和 endRowkey 转换为 `byte[]` 时的方式，默认值为 false，若为 true，则调用 `Bytes.toBytesBinary(rowkey)` 方法进行转换; 若为 false：则调用 `Bytes.toBytes(rowkey)`

配置格式如下：

```
"range": {
  "startRowkey": "aaa",
  "endRowkey": "ccc",
  "isBinaryRowkey":false
}
```

3.3 类型转换

下面列出支持的读取 HBase 数据类型，HbaseReader 针对 HBase 类型转换列表:

请注意:

除上述罗列字段类型外，其他类型均不支持

2.7 Hbase11XReader 插件文档

HbaseReader 插件实现了从 Hbase 中读取数据。在底层实现上，HbaseReader 通过 HBase 的 Java 客户端连接远程 HBase 服务，并通过 Scan 方式读取你指定 rowkey 范围内的数据，并将读取的数据使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

2.7.1 配置

以下演示基于下面创建的表以及数据

```
create 'users', {NAME=>'address', VERSIONS=>100},{NAME=>'info',VERSIONS=>1000}
put 'users', 'lisi', 'address:country', 'china1', 20200101
put 'users', 'lisi', 'address:province',    'beijing1', 20200101
put 'users', 'lisi', 'info:age',        27, 20200101
put 'users', 'lisi', 'info:birthday',   '1987-06-17', 20200101
put 'users', 'lisi', 'info:company',    'baidu1', 20200101
put 'users', 'xiaoming', 'address:city',    'hangzhou1', 20200101
put 'users', 'xiaoming', 'address:country', 'china1', 20200101
put 'users', 'xiaoming', 'address:province',    'zhejiang1',20200101
put 'users', 'xiaoming', 'info:age',        29, 20200101
put 'users', 'xiaoming', 'info:birthday',   '1987-06-17',20200101
put 'users', 'xiaoming', 'info:company',    'alibaba1', 20200101
put 'users', 'lisi', 'address:country', 'china2', 20200102
put 'users', 'lisi', 'address:province',    'beijing2', 20200102
put 'users', 'lisi', 'info:age',        27, 20200102
put 'users', 'lisi', 'info:birthday',   '1987-06-17', 20200102
put 'users', 'lisi', 'info:company',    'baidu2', 20200102
put 'users', 'xiaoming', 'address:city',    'hangzhou2', 20200102
put 'users', 'xiaoming', 'address:country', 'china2', 20200102
put 'users', 'xiaoming', 'address:province',    'zhejiang2', 20200102
put 'users', 'xiaoming', 'info:age',        29, 20200102
put 'users', 'xiaoming', 'info:birthday',   '1987-06-17', 20200102
put 'users', 'xiaoming', 'info:company',    'alibaba2', 20200102
```

目前 HbaseReader 支持两模式读取：normal 模式、multiVersionFixedColumn 模式；

normal 模式

把 HBase 中的表，当成普通二维表（横表）进行读取, 读取最新版本数据。如：

```
hbase(main):017:0> scan 'users'
ROW           COLUMN+CELL
 lisi         column=address:city, timestamp=1457101972764, value=beijing
 lisi         column=address:country, timestamp=1457102773908, value=china
 lisi         column=address:province, timestamp=1457101972736, value=beijing
 lisi         column=info:age, timestamp=1457101972548, value=27
 lisi         column=info:birthday, timestamp=1457101972604, value=1987-06-17
 lisi         column=info:company, timestamp=1457101972653, value=baidu
 xiaoming     column=address:city, timestamp=1457082196082, value=hangzhou
 xiaoming     column=address:country, timestamp=1457082195729, value=china
 xiaoming     column=address:province, timestamp=1457082195773, value=zhejiang
 xiaoming     column=info:age, timestamp=1457082218735, value=29
 xiaoming     column=info:birthday, timestamp=1457082186830, value=1987-06-17
 xiaoming     column=info:company, timestamp=1457082189826, value=alibaba
2 row(s) in 0.0580 seconds
```

读取后数据

multiVersionFixedColumn 模式

把 HBase 中的表，当成竖表进行读取。读出的每条记录一定是四列形式，依次为：`rowKey`, `family:qualifier`, `timestamp`, `value`。

读取时需要明确指定要读取的列，把每一个 cell 中的值，作为一条记录（record），若有多个版本就有多条记录（record）。如：

```
hbase(main):018:0> scan 'users',{VERSIONS=>5}
ROW                                  COLUMN+CELL
 lisi                                column=address:city, timestamp=1457101972764,␣
↪value=beijing
 lisi                                column=address:contry, timestamp=1457102773908,␣
↪value=china
 lisi                                column=address:province,␣
↪timestamp=1457101972736, value=beijing
 lisi                                column=info:age, timestamp=1457101972548,␣
↪value=27
 lisi                                column=info:birthday, timestamp=1457101972604,␣
↪value=1987-06-17
 lisi                                column=info:company, timestamp=1457101972653,␣
↪value=baidu
 xiaoming                            column=address:city, timestamp=1457082196082,␣
↪value=hangzhou
 xiaoming                            column=address:contry, timestamp=1457082195729,␣
↪value=china
 xiaoming                            column=address:province,␣
↪timestamp=1457082195773, value=zhejiang
 xiaoming                            column=info:age, timestamp=1457082218735,␣
↪value=29
 xiaoming                            column=info:age, timestamp=1457082178630,␣
↪value=24
 xiaoming                            column=info:birthday, timestamp=1457082186830,␣
↪value=1987-06-17
 xiaoming                            column=info:company, timestamp=1457082189826,␣
↪value=alibaba
```

(下页继续)

(续上页)

```
2 row(s) in 0.0260 seconds
```

读取后数据 (4 列)

配置一个从 HBase 抽取数据到本地的作业:（normal 模式）

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "hbase11xreader",
          "parameter": {
            "hbaseConfig": {
              "hbase.zookeeper.quorum": "xxxf"
            },
            "table": "users",
            "encoding": "utf-8",
            "mode": "normal",
            "column": [
              {
                "name": "rowkey",
                "type": "string"
              },
              {
                "name": "info: age",
                "type": "string"
              },
              {
                "name": "info: birthday",
                "type": "date",
                "format": "yyyy-MM-dd"
              },
              {
                "name": "info: company",
                "type": "string"
              },
              {
                "name": "address: country",
                "type": "string"
              },
              {
                "name": "address: province",
                "type": "string"
              },
              {
                "name": "address: city",
                "type": "string"
              }
            ],
            "range": {
```

(下页继续)

(续上页)

```
                "startRowkey": "",
                "endRowkey": "",
                "isBinaryRowkey": true
              }
            }
          },
          "writer": {
            "name": "txtfilewriter",
            "parameter": {
              "path": "/Users/shf/workplace/addax_test/hbase11xreader/result",
              "fileName": "qiran",
              "writeMode": "truncate"
            }
          }
        }
      ]
    }
}
```

配置一个从 HBase 抽取数据到本地的作业:（multiVersionFixedColumn 模式）

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "hbase11xreader",
          "parameter": {
            "hbaseConfig": {
              "hbase.zookeeper.quorum": "xxx"
            },
            "table": "users",
            "encoding": "utf-8",
            "mode": "multiVersionFixedColumn",
            "maxVersion": "-1",
            "column": [
              {
                "name": "rowkey",
                "type": "string"
              },
              {
                "name": "info: age",
                "type": "string"
              },
              {
                "name": "info: birthday",
                "type": "date",
                "format": "yyyy-MM-dd"
              },
              {
```

(下页继续)

(续上页)

```
                "name": "info: company",
                "type": "string"
              },
              {
                "name": "address: contry",
                "type": "string"
              },
              {
                "name": "address: province",
                "type": "string"
              },
              {
                "name": "address: city",
                "type": "string"
              }
            ],
            "range": {
              "startRowkey": "",
              "endRowkey": ""
            }
          }
        },
        "writer": {
          "name": "txtfilewriter",
          "parameter": {
            "path": "/Users/shf/workplace/addax_test/hbase11xreader/result",
            "fileName": "qiran",
            "writeMode": "truncate"
          }
        }
      }
    ]
  }
}
```

2.7.2 参数说明

column

描述：要读取的 hbase 字段，normal 模式与 multiVersionFixedColumn 模式下必填项。

normal 模式

name 指定读取的 hbase 列，除了 rowkey 外，必须为列族: 列名的格式，type 指定源数据的类型，format 指定日期类型的格式，value 指定当前类型为常量，不从 hbase 读取数据，而是根据 value 值自动生成对应的列。配置格式如下：

```
{
  "column": [
    {
      "name": "rowkey",
      "type": "string"
    },
    {
      "value": "test",
```

(下页继续)

(续上页)

```
      "type": "string"
    }
  ]
}
```

normal 模式下，对于用户指定 Column 信息，type 必须填写，name/value 必须选择其一。

multiVersionFixedColumn 模式

name 指定读取的 hbase 列，除了 rowkey 外，必须为列族: 列名的格式，type 指定源数据的类型，format 指定日期类型的格式。multiVersionFixedColumn 模式下不支持常量列。配置格式如下：

```
{
  "mode": "multiVersionFixedColumn",
  "maxVersion": 3,
  "column": [
    {
      "name": "rowkey",
      "type": "string"
    },
    {
      "name": "info: age",
      "type": "string"
    }
  ]
}
```

range

指定 hbasereader 读取的 rowkey 范围

- startRowkey：指定开始 rowkey
- endRowkey 指定结束 rowkey
- isBinaryRowkey：指定配置的 startRowkey 和 endRowkey 转换为 `byte[]` 时的方式，默认值为 false，若为 true，则调用 `Bytes.toBytesBinary(rowkey)` 方法进行转换; 若为 false：则调用 `Bytes.toBytes(rowkey)`

配置格式如下：

```
{
  "range": {
    "startRowkey": "aaa",
    "endRowkey": "ccc",
    "isBinaryRowkey": false
  }
}
```

2.7.3 类型转换

下面列出支持的读取 HBase 数据类型，HbaseReader 针对 HBase 类型转换列表:

请注意:

除上述罗列字段类型外，其他类型均不支持

2.7.4 限制

1. 目前不支持动态列的读取。考虑网络传输流量（支持动态列，需要先将 hbase 所有列的数据读取出来，再按规则进行过滤），现支持的两种读取模式中需要用户明确指定要读取的列。
2. 关于同步作业的切分：目前的切分方式是根据用户 hbase 表数据的 region 分布进行切分。即：在用户填写的 [startrowkey，endrowkey] 范围内，一个 region 会切分成一个 task，单个 region 不进行切分。
3. multiVersionFixedColumn 模式下不支持增加常量列

2.8 hbase11xsqlreader

hbase11xsqlreader 插件实现了从 Phoenix(HBase SQL) 读取数据。在底层实现上，hbase11xsqlreader 通过 Phoenix 客户端去连接远程的 HBase 集群，并执行相应的 sql 语句将数据从 Phoenix 库中 SELECT 出来。

2.8.1 功能说明

配置样例

配置一个从 Phoenix 同步抽取数据到本地的作业:

```
{
    "job": {
        "setting": {
            "speed": {
                "byte":-1,
              "channel": 1
            }
        },
        "content": [ {
                "reader": {
                    "name": "hbase11xsqlreader",
                    "parameter": {
                        "hbaseConfig": {
                            "hbase.zookeeper.quorum": "node1,node2,node3"
                        },
                        "table": "US_POPULATION",
                        "column": [],
                        "where": "1=1",
                        "querySql": ""
                    }
                },
                "writer": {
                    "name": "streamwriter",
                    "parameter": {
```

(下页继续)

(续上页)

```
                        "print":true,
                        "encoding": "UTF-8"
                    }
                }
            }
        ]
    }
}
```

参数说明

2.8.2 类型转换

目前 hbase11xsqlreader 支持大部分 Phoenix 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 MysqlReader 针对 Mysql 类型转换列表:

2.9 hbase20xsqlreader 插件文档

2.9.1 1 快速介绍

hbase20xsqlreader 插件实现了从 Phoenix(HBase SQL) 读取数据，对应版本为 HBase2.X 和 Phoenix5.X。

2.9.2 2 实现原理

简而言之，hbase20xsqlreader 通过 Phoenix 轻客户端去连接 Phoenix QueryServer，并根据用户配置信息生成查询 SELECT 语句，然后发送到 QueryServer 读取 HBase 数据，并将返回结果使用 Addax 自定义的数据类型拼装为抽象的数据集，最终传递给下游 Writer 处理。

2.9.3 3 功能说明

3.1 配置样例

配置一个从 Phoenix 同步抽取数据到本地的作业:

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "hbase20xsqlreader",
          "parameter": {
            "queryServerAddress": "http://127.0.0.1:8765",
            "serialization": "PROTOBUF",
            "table": "TEST",
            "column": [
              "ID",
```

(下页继续)

(续上页)

```
                "NAME"
              ],
              "splitKey": "ID"
            }
          },
          "writer": {
            "name": "streamwriter",
            "parameter": {
              "encoding": "UTF-8",
              "print": true
            }
          }
        }
      ],
      "setting": {
        "speed": {
          "channel": 3,
          "bytes": -1
        }
      }
    }
}
```

3.2 参数说明

3.3 类型转换

目前 hbase20xsqlreader 支持大部分 Phoenix 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 MysqlReader 针对 Mysql 类型转换列表:

2.9.4 4 约束限制

- 切分表时切分列仅支持单个列，且该列必须是表主键
- 不设置 splitPoint 默认使用自动切分，此时切分列仅支持整形和字符型
- 表名和 SCHEMA 名及列名大小写敏感，请与 Phoenix 表实际大小写保持一致
- 仅支持通过 Phoenix QeuryServer 读取数据，因此您的 Phoenix 必须启动 QueryServer 服务才能使用本插件

2.10 Addax HdfsReader 插件文档

2.10.1 1 快速介绍

HdfsReader 提供了读取分布式文件系统数据存储的能力。在底层实现上，HdfsReader 获取分布式文件系统上文件的数据，并转换为 Addax 传输协议传递给 Writer。

目前 HdfsReader 支持的文件格式如下:

- textfile（text）

- orcfile（orc）
- rcfile（rc）
- sequence file（seq）
- Csv(csv)
- parquet

2.10.2 2 功能与限制

HdfsReader 实现了从 Hadoop 分布式文件系统 Hdfs 中读取文件数据并转为 Addax 协议的功能。

`textfile` 是 Hive 建表时默认使用的存储格式，数据不做压缩，本质上 textfile 就是以文本的形式将数据存放在 hdfs 中，对于 Addax 而言，HdfsReader 实现上类比 TxtFileReader，有诸多相似之处。

orcfile，它的全名是 Optimized Row Columnar file，是对 RCFile 做了优化。

据官方文档介绍，这种文件格式可以提供一种高效的方法来存储 Hive 数据。HdfsReader 利用 Hive 提供的 OrcSerde 类，读取解析 orcfile 文件的数据。目前 HdfsReader 支持的功能如下：

1. 支持 textfile、orcfile、parquet、rcfile、sequence file 和 csv 格式的文件，且要求文件内容存放的是一张逻辑意义上的二维表。
2. 支持多种类型数据读取 (使用 String 表示)，支持列裁剪，支持列常量
3. 支持递归读取、支持正则表达式（`*` 和 `?`）。
4. 支持常见的压缩算法，包括 GZIP，SNAPPY，ZLIB 等。
5. 多个 File 可以支持并发读取。
6. 支持 sequence file 数据压缩，目前支持 lzo 压缩方式。
7. csv 类型支持压缩格式有：gzip、bz2、zip、lzo、lzo_deflate、snappy。
8. 目前插件中 Hive 版本为 `3.1.1`，Hadoop 版本为 `3.1.1`，在 Hadoop `2.7.x`，Hadoop `3.1.x` 和 Hive `2.x`，hive `3.1.x` 测试环境中写入正常；其它版本理论上都支持，但在生产环境使用前，请进一步测试；
9. 支持 `kerberos` 认证

2.10.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 3,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "hdfsreader",
          "parameter": {
            "path": "/user/hive/warehouse/mytable01/*",
```

(下页继续)

(续上页)

```
            "defaultFS": "hdfs://xxx:port",
            "column": [
              {
                "index": 0,
                "type": "long"
              },
              {
                "index": 1,
                "type": "boolean"
              },
              {
                "type": "string",
                "value": "hello"
              },
              {
                "index": 2,
                "type": "double"
              }
            ],
            "fileType": "orc",
            "encoding": "UTF-8",
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

3.2 配置项说明（各个配置项值前后不允许有空格）

path

要读取的文件路径，如果要读取多个文件，可以使用正则表达式 `*`，注意这里可以支持填写多个路径:

1. 当指定单个 Hdfs 文件，HdfsReader 暂时只能使用单线程进行数据抽取。二期考虑在非压缩文件情况下针对单个 File 可以进行多线程并发读取。
2. 当指定多个 Hdfs 文件，HdfsReader 支持使用多线程进行数据抽取。线程并发数通过通道数指定。
3. 当指定通配符，HdfsReader 尝试遍历出多个文件信息。例如: 指定 `/*` 代表读取 `/` 目录下所有的文件，指定 `/bazhen/*` 代表读取 bazhen 目录下游所有的文件。HdfsReader 目前只支持 `*` 和 `?` 作为文件通配符。

特别需要注意的是，Addax 会将一个作业下同步的所有的文件视作同一张数据表。用户必须自己保证所有的 File 能够适配同一套 schema 信息。并且提供给 Addax 权限可读。

defaultFS

Hadoop hdfs 文件系统 namenode 节点地址，如果 hdfs 配置了 HA 模式，则为 `defaultFS` 的值

目前 HdfsReader 已经支持 Kerberos 认证，如果需要权限认证，则需要用户配置 kerberos 参数，见下面

fileType

描述：文件的类型，目前只支持用户配置为

- text 表示 textfile 文件格式
- orc 表示 orcfile 文件格式
- rc 表示 rcfile 文件格式
- seq 表示 sequence file 文件格式
- csv 表示普通 hdfs 文件格式（逻辑二维表）
- parquet 表示 parquetfile 文件格式

特别需要注意的是，HdfsReader 能够自动识别文件是 orcfile、textfile 或者还是其它类型的文件，但该项是必填项，HdfsReader 则会只读取用户配置的类型的文件，忽略路径下其他格式的文件

另外需要注意的是，由于 textfile 和 orcfile 是两种完全不同的文件格式，所以 HdfsReader 对这两种文件的解析方式也存在差异，这种差异导致 hive 支持的复杂复合类型 (比如 map,array,struct,union) 在转换为 Addax 支持的 String 类型时，转换的结果格式略有差异，比如以 map 类型为例：

- orcfile map 类型经 hdfsreader 解析转换成 addax 支持的 string 类型后，结果为 `{job=80, team=60, person=70}`
- textfile map 类型经 hdfsreader 解析转换成 addax 支持的 string 类型后，结果为 `job:80,team:60, person:70`

从上面的转换结果可以看出，数据本身没有变化，但是表示的格式略有差异，所以如果用户配置的文件路径中要同步的字段在 Hive 中是复合类型的话，建议配置统一的文件格式。

如果需要统一复合类型解析出来的格式，我们建议用户在 hive 客户端将 textfile 格式的表导成 orcfile 格式的表

column

读取字段列表，type 指定源数据的类型，index 指定当前列来自于文本第几列 (以 0 开始)，value 指定当前类型为常量，不从源头文件读取数据，而是根据 value 值自动生成对应的列。

默认情况下，用户可以全部按照 String 类型读取数据，配置如下：

```
"column": ["*"]
```

用户可以指定 Column 字段信息，配置如下：

```
{
  "type": "long",
  "index": 0
  //从本地文件文本第一列获取 int 字段
},
{
"type": "string",
```

(下页继续)

(续上页)

```
"value": "alibaba"  //HdfsReader 内部生成 alibaba 的字符串字段作为当前字段
}
```

对于用户指定 Column 信息，type 必须填写，index/value 必须选择其一。

fieldDelimiter

读取的字段分隔符, HdfsReader 在读取 textfile 数据时, 需要指定字段分割符, 如果不指定默认为 `,` , HdfsReader 在读取 orcfile 时，用户无需指定字段分割符

encoding

读取文件的编码配置

nullFormat

文本文件中无法使用标准字符串定义 null(空指针)，Addax 提供 nullFormat 定义哪些字符串可以表示为 null。

例如如果用户配置: `"\\N"` ，那么如果源头数据是 `"\N"` ，Addax 视作 `null` 字段。

haveKerberos

是否有 Kerberos 认证，默认 `false`，如果用户配置 true，则配置项 `kerberosKeytabFilePath`，`kerberosPrincipal` 为必填。

kerberosKeytabFilePath

Kerberos 认证 keytab 文件路径，绝对路径

kerberosPrincipal

描述：Kerberos 认证 Principal 名，如 `xxxx/hadoopclient@xxx.xxx`

compress

当 fileType（文件类型）为 csv 下的文件压缩方式，目前仅支持 gzip、bz2、zip、lzo、lzo_deflate、hadoop-snappy、framing-snappy 压缩；值得注意的是，lzo 存在两种压缩格式：lzo 和 lzo_deflate，用户在配置的时候需要留心，不要配错了；另外，由于 snappy 目前没有统一的 stream format，addax 目前只支持最主流的两种：hadoop-snappy（hadoop 上的 snappy stream format）和 framing-snappy（google 建议的 snappy stream format）；

hadoopConfig

hadoopConfig 里可以配置与 Hadoop 相关的一些高级参数，比如 HA 的配置

```
{
  "hadoopConfig": {
    "dfs.nameservices": "cluster",
    "dfs.ha.namenodes.cluster": "nn1,nn2",
    "dfs.namenode.rpc-address.cluster.nn1": "node1.example.com:8020",
    "dfs.namenode.rpc-address.cluster.nn2": "node2.example.com:8020",
    "dfs.client.failover.proxy.provider.cluster": "org.apache.hadoop.hdfs.server.
↪namenode.ha.ConfiguredFailoverProxyProvider"
  }
}
```

这里的 cluster 表示 HDFS 配置成 HA 时的名字，也是 defaultFS 配置项中的名字如果实际环境中的名字不是 cluster ，则上述配置中所有写有 cluster 都需要替换

csvReaderConfig

读取 CSV 类型文件参数配置，Map 类型。读取 CSV 类型文件使用的 CsvReader 进行读取，会有很多配置，不配置则使用默认值。

常见配置：

```
"csvReaderConfig": {
"safetySwitch": false,
"skipEmptyRecords": false,
"useTextQualifier": false
}
```

所有配置项及默认值, 配置时 csvReaderConfig 的 map 中请 **严格按照以下字段名字进行配置**：

```
boolean caseSensitive = true;
char textQualifier = 34;
boolean trimWhitespace = true;
boolean useTextQualifier = true;//是否使用 csv 转义字符
char delimiter = 44;//分隔符
char recordDelimiter = 0;
char comment = 35;
boolean useComments = false;
int escapeMode = 1;
boolean safetySwitch = true;//单列长度是否限制 100000 字符
boolean skipEmptyRecords = true;//是否跳过空行
boolean captureRawRecord = true;
```

3.3 类型转换

由于 textfile 和 orcfile 文件表的元数据信息由 Hive 维护并存放在 Hive 自己维护的数据库（如 mysql）中，目前 HdfsReader 不支持对 Hive 元数

据数据库进行访问查询，因此用户在进行类型转换的时候，必须指定数据类型，如果用户配置的 column 为 *，则所有 column 默认转换为

string 类型。HdfsReader 提供了类型转换的建议表如下：

其中：

- Long 是指 Hdfs 文件文本中使用整形的字符串表示形式，例如 `123456789`
- Double 是指 Hdfs 文件文本中使用 Double 的字符串表示形式，例如 `3.1415`
- Boolean 是指 Hdfs 文件文本中使用 Boolean 的字符串表示形式，例如 `true、false`。不区分大小写。
- Date 是指 Hdfs 文件文本中使用 Date 的字符串表示形式，例如 `2014-12-31`
- Bytes 是指 HDFS 文件中使用二进制存储的内容，比如一张图片的数据

特别提醒：

- Hive 支持的数据类型 TIMESTAMP 可以精确到纳秒级别，所以 textfile、orcfile 中 TIMESTAMP 存放的数据类似于 `2015-08-21 22:40:47.397898389`，如果转换的类型配置为 Addax 的 Date，转换之后会导致纳秒部分丢失，所以如果需要保留纳秒部分的数据，请配置转换类型为 Addax 的 String 类型。

3.4 按分区读取

Hive 在建表的时候，可以指定分区 partition，例如创建分区 partition(day="20150820",hour="09")，对应的 hdfs 文件系统中，相应的表的目录下则会多出/20150820 和/09 两个目录，且/20150820 是/09 的父目录。了解了分区都会列成相应的目录结构，在按照某个分区读取某个表所有数据时，则只需配置好 json 中 path 的值即可。

比如需要读取表名叫 mytable01 下分区 day 为 20150820 这一天的所有数据，则配置如下：

```
"path": "/user/hive/warehouse/mytable01/20150820/*"
```

2.10.4 5 约束限制

略

2.10.5 6 FAQ

1. 如果报 java.io.IOException: Maximum column length of 100,000 exceeded in column... 异常信息，说明数据源 column 字段长度超过了 100000 字符。

需要在 json 的 reader 里增加如下配置

```
"csvReaderConfig": {
"safetySwitch": false,
"skipEmptyRecords": false,
"useTextQualifier": false
}
```

```
safetySwitch = false; //单列长度不限制 100000 字符
```

2.11 HttpReader

HttpReader 插件实现了读取 Restful API 数据的能力

2.11.1 示例

示例接口与数据

以下配置演示了如何从一个指定的 API 中获取数据，假定访问的接口为：

http://127.0.0.1:9090/mock/17/LDJSC/ASSET

走 GET 请求，请求的参数有

以下是访问的数据样例，（实际返回数据略有不同）


```
{
  "result": [
    {
      "CURR_DATE": "2019-12-09",
      "DEPT": "9700",
      "TOTAL_MANAGED_MARKET_VALUE": 1581.03,
      "TOTAL_MANAGED_MARKET_VALUE_GROWTH": 36.75,
      "TMMARKET_VALUE_DOD_GROWTH_RATE": -0.009448781026677719,
      "TMMARKET_VALUE_GROWTH_MON": -0.015153586011995693,
      "TMMARKET_VALUE_GROWTH_YEAR": 0.0652347643813081,
      "TMMARKET_VALUE_SHARECOM": 0.024853621341525287,
      "TMMARKET_VALUE_SHARE_GROWTH_RATE": -0.005242133578517903,
      "AVERAGE_NEW_ASSETS_DAYINMON": 1645.1193961136973,
      "YEAR_NEW_ASSET_SSHARECOM": 0.16690149257388515,
      "YN_ASSET_SSHARECOM_GROWTH_RATE": 0.017886267801303465,
      "POTENTIAL_LOST_ASSETS": 56.76,
      "TOTAL_LIABILITIES": 57.81,
      "TOTAL_ASSETS": 1306.33,
      "TOTAL_ASSETS_DOD_GROWTH": 4.79,
      "TOTAL_ASSETS_DOD_GROWTH_RATE": -0.006797058194980485,
      "NEW_ASSETS_DAY": 14.92,
      "NEW_ASSETS_MON": 90.29,
      "NEW_ASSETS_YEAR": 297.32,
      "NEW_ASSETS_DOD_GROWTH_RATE": -0.04015576541561927,
      "NEW_FUNDS_DAY": 18.16,
      "INFLOW_FUNDS_DAY": 2.12,
      "OUTFLOW_FUNDS_DAY": 9.73,
      "OVERALL_POSITION": 0.810298404938773,
      "OVERALL_POSITION_DOD_GROWTH_RATE": -0.03521615634095476,
      "NEW_CUST_FUNDS_MON": 69.44,
      "INFLOW_FUNDS_MONTH": 62.26,
      "OUTFLOW_FUNDS_MONTH": 32.59
    },
    {
      "CURR_DATE": "2019-08-30",
      "DEPT": "8700",
      "TOTAL_MANAGED_MARKET_VALUE": 1596.74,
      "TOTAL_MANAGED_MARKET_VALUE_GROWTH": 41.86,
      "TMMARKET_VALUE_DOD_GROWTH_RATE": 0.03470208565515685,
      "TMMARKET_VALUE_GROWTH_MON": 0.07818120801111743,
```


(下页继续)

(续上页)


```
    "TMMARKET_VALUE_GROWTH_YEAR": -0.05440250244736409,
    "TMMARKET_VALUE_SHARECOM": 0.09997733019626448,
    "TMMARKET_VALUE_SHARE_GROWTH_RATE": -0.019726478499825697,
    "AVERAGE_NEW_ASSETS_DAYINMON": 1007.9314679742108,
    "YEAR_NEW_ASSET_SSHARECOM": 0.15123738798885086,
    "YN_ASSET_SSHARECOM_GROWTH_RATE": 0.04694052069678048,
    "POTENTIAL_LOST_ASSETS": 52.48,
    "TOTAL_LIABILITIES": 55.28,
    "TOTAL_ASSETS": 1366.72,
    "TOTAL_ASSETS_DOD_GROWTH": 10.12,
    "TOTAL_ASSETS_DOD_GROWTH_RATE": 0.009708491982487952,
    "NEW_ASSETS_DAY": 12.42,
    "NEW_ASSETS_MON": 41.14,
    "NEW_ASSETS_YEAR": 279.32,
    "NEW_ASSETS_DOD_GROWTH_RATE": -0.025878627161898062,
    "NEW_FUNDS_DAY": 3.65,
    "INFLOW_FUNDS_DAY": 14.15,
    "OUTFLOW_FUNDS_DAY": 17.08,
    "OVERALL_POSITION": 0.9098432997243932,
    "OVERALL_POSITION_DOD_GROWTH_RATE": 0.02111922282868306,
    "NEW_CUST_FUNDS_MON": 57.21,
    "INFLOW_FUNDS_MONTH": 61.16,
    "OUTFLOW_FUNDS_MONTH": 15.83
  },
  {
    "CURR_DATE": "2019-06-30",
    "DEPT": "6501",
    "TOTAL_MANAGED_MARKET_VALUE": 1506.72,
    "TOTAL_MANAGED_MARKET_VALUE_GROWTH": -13.23,
    "TMMARKET_VALUE_DOD_GROWTH_RATE": -0.0024973354204176554,
    "TMMARKET_VALUE_GROWTH_MON": -0.015530793150701896,
    "TMMARKET_VALUE_GROWTH_YEAR": -0.08556724628979398,
    "TMMARKET_VALUE_SHARECOM": 0.15000077963967678,
    "TMMARKET_VALUE_SHARE_GROWTH_RATE": -0.049629446804825755,
    "AVERAGE_NEW_ASSETS_DAYINMON": 1250.1040863177336,
    "YEAR_NEW_ASSET_SSHARECOM": 0.19098445630488178,
    "YN_ASSET_SSHARECOM_GROWTH_RATE": -0.007881179708853471,
    "POTENTIAL_LOST_ASSETS": 50.53,
    "TOTAL_LIABILITIES": 56.62,
    "TOTAL_ASSETS": 1499.53,
    "TOTAL_ASSETS_DOD_GROWTH": 29.56,
    "TOTAL_ASSETS_DOD_GROWTH_RATE": -0.02599813232345556,
    "NEW_ASSETS_DAY": 28.81,
    "NEW_ASSETS_MON": 123.24,
    "NEW_ASSETS_YEAR": 263.63,
    "NEW_ASSETS_DOD_GROWTH_RATE": 0.0073986669331394875,
    "NEW_FUNDS_DAY": 18.52,
    "INFLOW_FUNDS_DAY": 3.26,
    "OUTFLOW_FUNDS_DAY": 6.92,
    "OVERALL_POSITION": 0.8713692113306709,
    "OVERALL_POSITION_DOD_GROWTH_RATE": 0.02977644553289545,
    "NEW_CUST_FUNDS_MON": 85.14,
    "INFLOW_FUNDS_MONTH": 23.35,
    "OUTFLOW_FUNDS_MONTH": 92.95
  },
  {
```


(下页继续)

(续上页)


```
      "CURR_DATE": "2019-12-07",
      "DEPT": "8705",
      "TOTAL_MANAGED_MARKET_VALUE": 1575.85,
      "TOTAL_MANAGED_MARKET_VALUE_GROWTH": 8.94,
      "TMMARKET_VALUE_DOD_GROWTH_RATE": -0.04384846980627058,
      "TMMARKET_VALUE_GROWTH_MON": -0.022962456288549656,
      "TMMARKET_VALUE_GROWTH_YEAR": -0.005047009316021089,
      "TMMARKET_VALUE_SHARECOM": 0.07819484815809447,
      "TMMARKET_VALUE_SHARE_GROWTH_RATE": -0.008534369960890256,
      "AVERAGE_NEW_ASSETS_DAYINMON": 1340.0339240689955,
      "YEAR_NEW_ASSET_SSHARECOM": 0.19019952857677042,
      "YN_ASSET_SSHARECOM_GROWTH_RATE": 0.01272353909992914,
      "POTENTIAL_LOST_ASSETS": 54.63,
      "TOTAL_LIABILITIES": 53.17,
      "TOTAL_ASSETS": 1315.08,
      "TOTAL_ASSETS_DOD_GROWTH": 49.31,
      "TOTAL_ASSETS_DOD_GROWTH_RATE": 0.0016538407028265922,
      "NEW_ASSETS_DAY": 29.17,
      "NEW_ASSETS_MON": 44.75,
      "NEW_ASSETS_YEAR": 172.87,
      "NEW_ASSETS_DOD_GROWTH_RATE": 0.045388692595736746,
      "NEW_FUNDS_DAY": 18.46,
      "INFLOW_FUNDS_DAY": 12.93,
      "OUTFLOW_FUNDS_DAY": 10.38,
      "OVERALL_POSITION": 0.8083127036694828,
      "OVERALL_POSITION_DOD_GROWTH_RATE": -0.02847453515632541,
      "NEW_CUST_FUNDS_MON": 49.74,
      "INFLOW_FUNDS_MONTH": 81.93,
      "OUTFLOW_FUNDS_MONTH": 18.17
    }
  ]
}
```


我们需要把 `result` 结果中的部分 key 值数据获取

配置

以下配置实现从接口获取数据并打印到终端

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "httpreader",
          "parameter": {
            "connection": [
              {
                "url": "http://127.0.0.1:9090/mock/17/LDJSC/ASSET",
```

(下页继续)

(续上页)

```
                "proxy": {
                    "host": "http://127.0.0.1:3128",
                    "auth": "user:pass"
                }
              }
            ],
            "reqParams": {
              "CURR_DATE":"2021-01-18",
              "DEPT":"9700"
            },
            "resultKey":"result",
            "method": "GET",
            "column": ["CURR_DATE","DEPT","TOTAL_MANAGED_MARKET_VALUE","TOTAL_MANAGED_
↪MARKET_VALUE_GROWTH"],
            "username": "user",
            "password": "passw0rd",
            "headers": {
                "X-Powered-by": "Addax"
            }
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": "true"
          }
        }
      }
    ]
  }
}
```

将上述内容保存为 `job/httpreader2stream.json` 文件。

执行

执行以下命令，进行采集

```
bin/addax.sh job/httpreader2stream.json
```

上述命令的输出结果大致如下：

```
2021-01-20 09:07:41.864 [main] INFO  VMInfo - VMInfo# operatingSystem class => com.
↪sun.management.internal.OperatingSystemImpl
2021-01-20 09:07:41.877 [main] INFO  Engine - the machine info  =>

        osInfo:         Mac OS X x86_64 10.15.1
        jvmInfo:        AdoptOpenJDK 14 14.0.2+12
        cpu num:        8

        totalPhysicalMemory:        -0.00G
        freePhysicalMemory:        -0.00G
        maxFileDescriptorCount:        -1
        currentOpenFileDescriptorCount:        -1
```

(下页继续)

(续上页)

```
        GC Names        [G1 Young Generation, G1 Old Generation]

        MEMORY_NAME                    | allocation_size                | init_size
        CodeHeap 'profiled nmethods'   | 117.21MB                       | 2.44MB
        G1 Old Gen                     | 2,048.00MB                     | 39.00MB
        G1 Survivor Space              | -0.00MB                        | 0.00MB
        CodeHeap 'non-profiled nmethods' | 117.21MB                       | 2.44MB
        Compressed Class Space         | 1,024.00MB                     | 0.00MB
        Metaspace                      | -0.00MB                        | 0.00MB
        G1 Eden Space                  | -0.00MB                        | 25.00MB
        CodeHeap 'non-nmethods'        | 5.57MB                         | 2.44MB


2021-01-20 09:07:41.903 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "reqParams":{
                                                "CURR_DATE":"2021-01-18",
                                                "DEPT":"9700"
                                        },
                                        "method":"GET",
                                        "column":[
                                                "CURR_DATE",
                                                "DEPT",
                                                "TOTAL_MANAGED_MARKET_VALUE",
                                                "TOTAL_MANAGED_MARKET_VALUE_GROWTH"
                                        ],
                                        "resultKey":"result",
                                        "connection":[
                                                {
                                                        "url":"http://127.0.0.1:9090/
↪mock/17/LDJSC/ASSET"
                                                }
                                        ]
                                },
                                "name":"httpreader"
                        },
                        "writer":{
                                "parameter":{
                                        "print":"true"
                                },
                                "name":"streamwriter"
                        }
                }
        ],
        "setting":{
                "speed":{
                        "bytes":-1,
                        "channel":1
                }
        }
}
```

(下页继续)

(续上页)

```
2021-01-20 09:07:41.926 [main] INFO  PerfTrace - PerfTrace traceId=job_-1,␣
↪isEnable=false, priority=0
2021-01-20 09:07:41.927 [main] INFO  JobContainer - Addax jobContainer starts job.
2021-01-20 09:07:41.928 [main] INFO  JobContainer - Set jobId = 0
2021-01-20 09:07:42.002 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0]␣
↪taskId[0] attemptCount[1] is started

2019-08-30        9700        1539.85        -14.78
2019-10-01        9700        1531.71        47.66
2020-12-03        9700        1574.38        7.34
2020-11-31        9700        1528.13        41.62
2019-03-01        9700        1554.28        -9.29

2021-01-20 09:07:45.006 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-01-20 09:07:41
任务结束时刻                    : 2021-01-20 09:07:44
任务总计耗时                    :                  3s
任务平均流量                    :               42B/s
记录写入速度                    :              1rec/s
读出记录总数                    :                   5
读写失败总数                    :                   0
```

参数说明

proxy

如果访问的接口需要通过代理，则可以配置 proxy 配置项，该配置项是一个 json 字典，包含一个必选的 host 字段和一个可选的 auth 字段。

```
{
  "proxy": {
    "host": "http://127.0.0.1:8080",
    "auth": "user:pass"
  }
}
```

如果是 sock 代理 (V4,v5)，则可以写

```
{
  "proxy": {
    "host": "socks://127.0.0.1:8080",
    "auth": "user:pass"
  }
}
```

host 是代理地址，包含代理类型，目前仅支持 http 代理和 socks(V4, V5 均可) 代理。如果代理需要认证，则可以配置 auth , 它由用户名和密码组成，两者之间用冒号 (:) 隔开。

限制说明

1. 返回的结果必须是 JSON 类型
2. 当前所有 key 的值均当作字符串类型
3. 暂不支持接口 Token 鉴权模式
4. 暂不支持分页获取
5. 代理仅支持 `http` 模式

2.12 InfluxDBReader

InfluxDBReader 插件实现了从 InfluxDB 读取数据。底层实现上，是通过调用 InfluQL 语言查询表，然后获得返回数据。

2.12.1 示例

以下示例用来演示该插件如何从指定表 (即指标) 上读取数据并输出到终端

创建需要的库表和数据

通过以下命令来创建需要读取的表以及数据

```
# create database
influx --execute "CREATE DATABASE NOAA_water_database"
# download sample data
curl https://s3.amazonaws.com/noaa.water-database/NOAA_data.txt -o NOAA_data.txt
# import data via influx-cli
influx -import -path=NOAA_data.txt -precision=s -database=NOAA_water_database
```

创建 job 文件

创建 `job/influxdb2stream.json` 文件，内容如下：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "influxdbreader",
          "parameter": {
            "column": [
              "*"
            ],
            "connection": [
              {
                "endpoint": "http://localhost:8086",
                "database": "NOAA_water_database",
                "table": "h2o_feet",
                "where": "1=1"
```

(下页继续)

(续上页)

```
                }
              ],
              "connTimeout": 15,
              "readTimeout": 20,
              "writeTimeout": 20,
              "username": "influx",
              "password": "influx123"
            }
          },
          "writer": {
            "name": "streamwriter",
            "parameter": {
              "print": "true"
            }
          }
        }
      ],
      "setting": {
        "speed": {
          "bytes": -1,
          "channel": 1
        }
      }
    }
}
```

运行

执行下面的命令进行数据采集

```
bin/addax.sh job/influxdb2stream.json
```

2.12.2 参数说明

2.12.3 类型转换

当前实现是将所有字段当作字符串处理

2.12.4 限制

1. 当前插件仅支持 1.x 版本，2.0 及以上并不支持

2.13 JsonFileReader 插件文档

2.13.1 1 快速介绍

JsonFileReader 提供了读取本地文件系统数据存储的能力。在底层实现上，JsonFileReader 获取本地文件数据，使用 Jayway JsonPath 抽取 Json 字符串，并转换为 Addax 传输协议传递给 Writer。

2.13.2 2 功能与限制

JsonFileReader 实现了从本地文件读取数据并转为 Addax 协议的功能，本地文件是可以是 Json 数据格式的集合，对于 Addax 而言，JsonFileReader 实现上类比 TxtFileReader，有诸多相似之处。目前 JsonFileReader 支持功能如下：

1. 支持且仅支持读取 TXT 的文件，且要求 TXT 中 s 内容必须符合 json
2. 支持列常量和 Json 的 Key 为空值
3. 支持递归读取、支持文件名过滤
4. 多个 File 可以支持并发读取

我们暂时不能做到：

1. 单个 File 支持多线程并发读取，这里涉及到单个 File 内部切分算法。
2. 单个 File 在压缩情况下，从技术上无法支持多线程并发读取。
3. 暂不支持读取压缩文件和日期类型的自定义日期

2.13.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": "true"
          }
        },
        "reader": {
          "name": "jsonfilereader",
          "parameter": {
            "path": [
              "/tmp/test*.json"
            ],
```

(下页继续)

(续上页)

```
            "column": [
              {
                "index": "$.id",
                "type": "long"
              },
              {
                "index": "$.name",
                "type": "string"
              },
              {
                "index": "$.age",
                "type": "long"
              },
              {
                "index": "$.score.math",
                "type": "double"
              },
              {
                "index": "$.score.english",
                "type": "double"
              },
              {
                "index": "$.pubdate",
                "type": "date"
              },
              {
                "type": "string",
                "value": "constant string"
              }
            ]
          }
        }
      }
    ]
  }
}
```

其中 /tmp/test*.json 为同一个 json 文件的多个复制，内容如下：

```
{
  "name": "zhangshan",
  "id": 19890604,
  "age": 12,
  "score": {
    "math": 92.5,
    "english": 97.5,
    "chinese": 95
  },
  "pubdate": "2020-09-05"
}
```

3.2 参数说明

path

本地文件系统的路径信息，注意这里可以支持填写多个路径

- 当指定单个本地文件，JsonFileReader 暂时只能使用单线程进行数据抽取。
- 当指定多个本地文件，JsonFileReader 支持使用多线程进行数据抽取。线程并发数通过通道数指定。
- 当指定通配符，JsonFileReader 尝试遍历出多个文件信息。例如: 指定`/*` 代表读取/目录下所有的文件，指定`/bazhen/*` 代表读取 bazhen 目录下游所有的文件。JsonFileReader 目前只支持 `*` 作为文件通配符。

特别需要注意的是，如果 Path 指定的路径下没有符合匹配的文件抽取，Addax 将报错。

column

读取字段列表，type 指定源数据的类型，index 指定当前列来自于 json 的指定，语法为 Jayway JsonPath 的语法，value 指定当前类型为常量，不从源头文件读取数据，而是根据 value 值自动生成对应的列。用户必须指定 Column 字段信息

对于用户指定 Column 信息，type 必须填写，index/value 必须选择其一

3.3 类型转换

本地文件本身不提供数据类型，该类型是 Addax JsonFileReade 定义：

2.14 KuduReader

KuduReader 插件利用 Kudu 的 java 客户端 KuduClient 进行 Kudu 的读操作。

KuduReader 通过 Addax 框架从 Kudu 并行的读取数据，通过主控的 JOB 程序按照指定的规则对 Kudu 中的数据进行分片，并行读取，然后将 Kudu 支持的类型通过逐一判断转换成 Addax 支持的类型。

2.14.1 示例

我们通过 Trino 的 `kudu connector` 连接 kudu 服务，然后进行表创建以及数据插入

建表语句以及数据插入语句

```
CREATE TABLE kudu.default.users (
  user_id int WITH (primary_key = true),
  user_name varchar,
  age int,
  salary double,
  longtitue decimal(18,6),
  latitude decimal(18,6),
  p decimal(21,20),
  mtime timestamp
) WITH (
  partition_by_hash_columns = ARRAY['user_id'],
```

(下页继续)

(续上页)

```
  partition_by_hash_buckets = 2
);

insert into kudu.default.users
values
(1, cast('wgzhao' as varchar), 18, cast(18888.88 as double),
 cast(123.282424 as decimal(18,6)), cast(23.123456 as decimal(18,6)),
 cast(1.12345678912345678912 as decimal(21,20)),
 timestamp '2021-01-10 14:40:41'),
(2, cast('anglina' as varchar), 16, cast(23456.12 as double),
 cast(33.192123 as decimal(18,6)), cast(56.654321 as decimal(18,6)),
 cast(1.12345678912345678912 as decimal(21,20)),
 timestamp '2021-01-10 03:40:41');
-- ONLY insert primary key value
 insert into kudu.default.users(user_id) values  (3);
```

配置

以下是读取 kudu 表并输出到终端的配置

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 3
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [
      {
        "reader": {
          "name": "kudureader",
          "parameter": {
            "masterAddress": "localhost:7051,localhost:7151,localhost:7251",
            "table": "users",
            "splitPk": "user_id",
            "lowerBound": 1,
            "upperBound": 100,
            "readTimeout": 5,
            "scanTimeout": 10
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

把上述配置文件保存为 job/kudu2stream.json

执行

执行下面的命令进行采集

```
bin/addax.sh job/kudu2stream.json
```

输出结果类似如下（删除了不必需要的内容)

```
2021-01-10 15:46:59.303 [main] INFO  VMInfo - VMInfo# operatingSystem class => sun.
↪management.OperatingSystemImpl

2021-01-10 15:46:59.329 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "masterAddress":"localhost:7051,
↪localhost:7151,localhost:7251",
                                        "upperBound":100,
                                        "readTimeout":5,
                                        "lowerBound":1,
                                        "splitPk":"user_id",
                                        "table":"users",
                                        "scanTimeout":10
                                },
                                "name":"kudureader"
                        },
                        "writer":{
                                "parameter":{
                                        "print":true
                                },
                                "name":"streamwriter"
                        }
                }
        ],
        "setting":{
                "errorLimit":{
                        "record":0,
                        "percentage":0.02
                },
                "speed":{
                        "channel":3
                }
        }
}

3        null        null        null        null        null        null        null
1        wgzhao        18        18888.88        123.282424        23.123456        1.
↪12345678912345678912        2021-01-10 22:40:41
2        anglina        16        23456.12        33.192123        56.654321        1.
↪12345678912345678912        2021-01-10 11:40:41

任务启动时刻                    : 2021-01-10 15:46:59
任务结束时刻                    : 2021-01-10 15:47:02
```

(下页继续)

(续上页)

```
任务总计耗时                  :              3s
任务平均流量                  :            52B/s
记录写入速度                  :           0rec/s
读出记录总数                  :               2
读写失败总数                  :               0
```

2.14.2 参数说明

2.14.3 类型转换

2.15 MongoDBReader 插件文档

2.15.1 1 快速介绍

MongoDBReader 插件利用 MongoDB 的 java 客户端 MongoClient 进行 MongoDB 的读操作。最新版本的 Mongo 已经将 DB 锁的粒度从 DB 级别降低到 document 级别，配合上 MongoDB 强大的索引功能，基本可以达到高性能的读取 MongoDB 的需求。

2.15.2 2 功能说明

2.1 配置样例

该示例从 MongoDB 中读一张表并打印到终端

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "mongodbreader",
          "parameter": {
            "address": [
              "127.0.0.1:32768"
            ],
            "userName": "",
            "userPassword": "",
            "dbName": "tag_per_data",
            "collectionName": "tag_data",
            "column": [
              {
                "name": "unique_id",
                "type": "string"
              },
              {
```

(下页继续)

(续上页)

```
                "name": "sid",
                "type": "string"
            },
            {
                "name": "user_id",
                "type": "string"
            },
            {
                "name": "auction_id",
                "type": "string"
            },
            {
                "name": "content_type",
                "type": "string"
            },
            {
                "name": "pool_type",
                "type": "string"
            },
            {
                "name": "frontcat_id",
                "type": "Array",
                "spliter": ""
            },
            {
                "name": "categoryid",
                "type": "Array",
                "spliter": ""
            },
            {
                "name": "gmt_create",
                "type": "string"
            },
            {
                "name": "taglist",
                "type": "Array",
                "spliter": " "
            },
            {
                "name": "property",
                "type": "string"
            },
            {
                "name": "scorea",
                "type": "int"
            },
            {
                "name": "scoreb",
                "type": "int"
            },
            {
                "name": "scorec",
                "type": "int"
            }
        ]
    }
```

(下页继续)

(续上页)

```
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": "true"
          }
        }
      }
    ]
  }
}
```

2.15.3 3 参数说明

2.15.4 4 类型转换

2.16 MysqlReader

MysqlReader 插件实现了从 Mysql 读取数据。在底层实现上，MysqlReader 通过 JDBC 连接远程 Mysql 数据库，并执行相应的 sql 语句将数据从 mysql 库中 SELECT 出来。

不同于其他关系型数据库，MysqlReader 不支持 FetchSize

MysqlReader 通过 JDBC 连接器连接到远程的 Mysql 数据库，并根据用户配置的信息生成查询 SELECT SQL 语句，然后发送到远程 Mysql 数据库，并将该 SQL 执行返回结果使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

对于用户配置 Table、Column、Where 的信息，MysqlReader 将其拼接为 SQL 语句发送到 Mysql 数据库；对于用户配置 querySql 信息，MysqlReader 直接将其发送到 Mysql 数据库。

2.16.1 示例

我们在 MySQL 的 test 库上创建如下表：

```
CREATE TABLE addax_reader
(
    c_bigint     bigint,
    c_varchar    varchar(100),
    c_timestamp  timestamp,
    c_text       text,
    c_decimal    decimal(8, 3),
    c_mediumtext mediumtext,
    c_longtext   longtext,
    c_int        int,
    c_time       time,
    c_datetime   datetime,
    c_enum       enum('one', 'two', 'three'),
    c_float      float,
    c_smallint   smallint,
    c_bit        bit,
    c_double     double,
    c_blob       blob,
```

(下页继续)

(续上页)

```
    c_char       char(5),
    c_varbinary  varbinary(100),
    c_tinyint    tinyint,
    c_json       json,
    c_set SET ('a', 'b', 'c', 'd'),
    c_binary     binary,
    c_longblob   longblob,
    c_mediumblob mediumblob
);

然后插入下面一条记录

```sql
INSERT INTO addax_reader
VALUES (2E18,
        'a varchar data',
        '2021-12-12 12:12:12',
        'a long text',
        12345.122,
        'a medium text',
        'a long text',
        2 ^ 32 - 1,
        '12:13:14',
        '2021-12-12 12:13:14',
        'one',
        17.191,
        126,
        0,
        1114.1114,
        'blob',
        'a123b',
        'a var binary content',
        126,
        '{"k1":"val1","k2":"val2"}',
        'b',
        binary(1),
        x
↪'89504E470D0A1A0A0000000D49484452000000100000001008020000009091683600000001735247420
↪',
        x '89504E470D0A1A0A0000000D');
```

下面的配置是读取该表到终端的作业:

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 3,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "mysqlreader",
          "parameter": {
```

(下页继续)
```

(续上页)

```
            "username": "root",
            "password": "root",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "addax_reader"
                ],
                "jdbcUrl": [
                  "jdbc:mysql://127.0.0.1:3306/test"
                ],
                "driver": "com.mysql.jdbc.Driver"
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/mysql2stream.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/mysql2stream.json
```

2.16.2 参数说明

driver

当前 Addax 采用的 MySQL JDBC 驱动为 8.0 以上版本，驱动类名使用的 `com.mysql.cj.jdbc.Driver`，而不是 `com.mysql.jdbc.Driver`。如果你需要采集的 MySQL 服务低于 `5.6`，需要使用到 `Connector/J 5.1` 驱动，则可以采取下面的步骤：

替换插件内置的驱动

```
rm -f plugin/reader/mysqlreader/lib/mysql-connector-java-*.jar
```

拷贝老的驱动到插件目录

```
cp mysql-connector-java-5.1.48.jar plugin/reader/mysqlreader/lib/
```

指定驱动类名称

在你的 json 文件类，配置 `"driver": "com.mysql.jdbc.Driver"`

2.16.3 类型转换

目前 MysqlReader 支持大部分 Mysql 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 MysqlReader 针对 Mysql 类型转换列表:

请注意:

- 除上述罗列字段类型外，其他类型均不支持
- `tinyint(1)` Addax 视作为整形
- `year` Addax 视作为字符串类型
- `bit` Addax 属于未定义行为

3.4 数据库编码问题

Mysql 本身的编码设置非常灵活，包括指定编码到库、表、字段级别，甚至可以均不同编码。优先级从高到低为字段、表、库、实例。我们不推荐数据库用户设置如此混乱的编码，最好在库级别就统一到 UTF-8。

MysqlReader 底层使用 JDBC 进行数据抽取，JDBC 天然适配各类编码，并在底层进行了编码转换。因此 MysqlReader 不需用户指定编码，可以自动获取编码并转码。

对于 Mysql 底层写入编码和其设定的编码不一致的混乱情况，MysqlReader 对此无法识别，对此也无法提供解决方案，对于这类情况，导出有可能为乱码。

2.17 OracleReader 插件文档

2.17.1 1 快速介绍

OracleReader 插件实现了从 Oracle 读取数据。在底层实现上，OracleReader 通过 JDBC 连接远程 Oracle 数据库，并执行相应的 sql 语句将数据从 Oracle 库中 SELECT 出来。

2.17.2 2 实现原理

简而言之，OracleReader 通过 JDBC 连接器连接到远程的 Oracle 数据库，并根据用户配置的信息生成查询 SELECT SQL 语句并发送到远程 Oracle 数据库，并将该 SQL 执行返回结果使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

对于用户配置 Table、Column、Where 的信息，OracleReader 将其拼接为 SQL 语句发送到 Oracle 数据库；对于用户配置 querySql 信息，Oracle 直接将其发送到 Oracle 数据库。

2.17.3 3 功能说明

3.1 配置样例

配置一个从 Oracle 数据库同步抽取数据到本地的作业:

```
{
  "job": {
    "setting": {
      "speed": {
```

(下页继续)

(续上页)

```
        "byte": 1048576,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "oraclereader",
          "parameter": {
            "username": "root",
            "password": "root",
            "column": [
              "id",
              "name"
            ],
            "splitPk": "db_id",
            "connection": [
              {
                "table": [
                  "table"
                ],
                "jdbcUrl": [
                  "jdbc:oracle:thin:@<HOST_NAME>:PORT:<DATABASE_NAME>"
                ]
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

session

控制写入数据的时间格式，时区等的配置，如果表中有时间字段，配置该值以明确告知写入 oracle 的时间格式。通常配置的参数为：NLS_DATE_FORMAT,NLS_TIME_FORMAT。其配置的值为 json 格式，例如：

```
"session": [
"alter session set NLS_DATE_FORMAT='yyyy-mm-dd hh24:mi:ss'",
"alter session set NLS_TIMESTAMP_FORMAT='yyyy-mm-dd hh24:mi:ss'",
"alter session set NLS_TIMESTAMP_TZ_FORMAT='yyyy-mm-dd hh24:mi:ss'",
"alter session set TIME_ZONE='Asia/Chongqing'"
]
```

注意 " 是 " 的转义字符串

3.3 类型转换

目前 OracleReader 支持大部分 Oracle 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 OracleReader 针对 Oracle 类型转换列表:

请注意: 除上述罗列字段类型外，其他类型均不支持

数据库编码问题

OracleReader 底层使用 JDBC 进行数据抽取，JDBC 天然适配各类编码，并在底层进行了编码转换。因此 OracleReader 不需用户指定编码，可以自动获取编码并转码。

对于 Oracle 底层写入编码和其设定的编码不一致的混乱情况，OracleReader 对此无法识别，对此也无法提供解决方案，对于这类情况，**导出有可能为乱码**。

2.18 PostgresqlReader

PostgresqlReader 插件实现了从 PostgreSQL 读取数据。在底层实现上，PostgresqlReader 通过 JDBC 连接远程 PostgreSQL 数据库，并执行相应的 sql 语句将数据从 PostgreSQL 库中 SELECT 出来。

2.18.1 示例

假定建表语句以及输入插入语句如下:

```
create table if not exists addax_tbl
(
    c_bigint bigint,
    c_bit bit(3),
    c_bool boolean,
    c_byte bytea,
    c_char char(10),
    c_varchar varchar(20),
    c_date  date,
    c_double float8,
    c_int integer,
    c_json json,
    c_number decimal(8,3),
    c_real  real,
    c_small smallint,
    c_text  text,
    c_ts timestamp,
    c_uuid uuid,
    c_xml xml,
    c_money money,
    c_inet inet,
    c_cidr cidr,
    c_macaddr macaddr
);
insert into addax_tbl
values (999988887777,
        B '101',
        TRUE,
```

(下页继续)

(续上页)

```
        '\xDEADBEEF',
        'hello',
        'hello, world',
        '2021-01-04',
        999888.9972,
        9876542,
        '{"bar": "baz", "balance": 7.77, "active": false}'::json,
        12345.123,
        123.123,
        126,
        'this is a long text ',
        '2020-01-04 12:13:14',
        'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11'::uuid,
        '<foo>bar</foo>'::xml,
        '52093.89'::money,
        '192.168.1.1'::inet,
        '192.168.1/24'::cidr,
        '08002b:010203'::macaddr);
```

配置一个从 PostgreSQL 数据库同步抽取数据到本地的作业:

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": -1,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "postgresqlreader",
          "parameter": {
            "username": "pgtest",
            "password": "pgtest",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "addax_tbl"
                ],
                "jdbcUrl": [
                  "jdbc:postgresql://127.0.0.1:5432/pgtest"
                ]
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
```

(下页继续)

(续上页)

```
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/postgres2stream.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/postgres2stream.json
```

其输出信息如下（删除了非关键信息）

```
2021-01-07 10:15:12.295 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "password":"*****",
                                        "column":[
                                                "*"
                                        ],
                                        "connection":[
                                                {
                                                        "jdbcUrl":[
                                                                "jdbc:postgresql://
↪localhost:5432/pgtest"
                                                        ],
                                                        "table":[
                                                                "addax_tbl"
                                                        ]
                                                }
                                        ],
                                        "username":"pgtest"
                                },
                                "name":"postgresqlreader"
                        },
                        "writer":{
                                "parameter":{
                                        "print":true
                                },
                                "name":"streamwriter"
                        }
                }
        ],
        "setting":{
                "speed":{
                        "byte":-1,
                        "channel":1
                }
        }
```

(下页继续)

(续上页)

```
}

999988887777        101        true            ⍰��        hello              hello,␣
↪world        2021-01-04        999888.99719999998        9876542        {"bar": "baz
↪", "balance": 7.77, "active": false}        12345.123        123.
↪123        126        this is a long text        2020-01-04␣
↪12:13:14        a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11        <foo>bar</foo>
↪        52093.89        192.168.1.1        192.168.1.0/24        08:00:2b:01:02:03

任务启动时刻                    : 2021-01-07 10:15:12
任务结束时刻                    : 2021-01-07 10:15:15
任务总计耗时                    :                  3s
任务平均流量                    :               90B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   1
读写失败总数                    :                   0
```

2.18.2 参数说明

2.18.3 类型转换

目前 PostgresqlReader 支持大部分 PostgreSQL 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 PostgresqlReader 针对 PostgreSQL 类型转换列表:

2.18.4 已知限制

除上述罗列字段类型外，其他类型均不支持;

2.19 RDBMS Reader

RDBMSReader 插件支持从传统 RDBMS 读取数据。这是一个通用关系数据库读取插件，可以通过注册数据库驱动等方式支持更多关系数据库读取。

同时 RDBMS Reader 又是其他关系型数据库读取插件的的基础类。以下读取插件均依赖该插件

- Oracle Reader
- MySQL Reader
- PostgreSQL Reader
- ClickHouse Reader
- SQLServer Reader

注意，对于 Addax 已经提供了专门的数据库读取插件的，推荐使用专用插件，如果你需要读取的数据库没有专门插件，则考虑使用该通用插件。在使用之前，还需要执行以下操作才可以正常运行，否则运行会出现异常。

2.19.1 配置驱动

假定你需要读取 IBM DB2 的数据，因为没有提供专门的读取插件，所以我们可以使用该插件来实现，在使用之前，需要执行下面两个操作：

1. 下载对应的 JDBC 驱动，并拷贝到 `plugin/reader/rdbmsreader/libs` 目录
2. 修改 `plugin/reader/rdbmsreader/plugin.json` 文件，找到 `drivers` 一项，填写正确的 JDBC 驱动名，比如 DB2 的驱动名为 `com.ibm.db2.jcc.DB2Driver`，类似这样：

```
{
"name": "rdbmsreader",
"class": "com.wgzhao.addax.plugin.reader.rdbmsreader.RdbmsReader",
"description": "",
"developer": "alibaba",
"drivers": ["com.ibm.db2.jcc.DB2Driver"]
}
```

以下列出常见的数据库以及对应的驱动名称

- Apache Impala: `com.cloudera.impala.jdbc41.Driver`
- Enterprise DB: `com.edb.Driver`
- PrestoDB: `com.facebook.presto.jdbc.PrestoDriver`
- IBM DB2: `com.ibm.db2.jcc.DB2Driver`
- MySQL: `com.mysql.cj.jdbc.Driver`
- Sybase Server: `com.sybase.jdbc3.jdbc.SybDriver`
- TDengine: `com.taosdata.jdbc.TSDBDriver`
- 达梦数据库: `dm.jdbc.driver.DmDriver`
- 星环 Inceptor: `io.transwarp.jdbc.InceptorDriver`
- TrinoDB: `io.trino.jdbc.TrinoDriver`
- PrestoSQL: `io.prestosql.jdbc.PrestoDriver`
- Oracle DB: `oracle.jdbc.OracleDriver`
- PostgreSQL: `org.postgresql.Drive`

2.19.2 配置说明

以下配置展示了如何从 Presto 数据库读取数据到终端

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": 1048576,
        "channel": 1
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
```

(下页继续)

(续上页)

```
    },
    "content": [
      {
        "reader": {
          "name": "rdbmsreader",
          "parameter": {
            "username": "hive",
            "password": "",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "default.table"
                ],
                "jdbcUrl": [
                  "jdbc:presto://127.0.0.1:8080/hive"
                ],
                "driver": ""
              }
            ],
            "fetchSize": 1024,
            "where": "1 = 1"
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

2.19.3 参数说明

`parameter` 配置项支持以下配置

jdbcUrl

`jdbcUrl` 配置除了配置必要的信息外，我们还可以在增加每种特定驱动的特定配置属性，这里特别提到我们可以利用配置属性对代理的支持从而实现通过代理访问数据库的功能。比如对于 PrestoSQL 数据库的 JDBC 驱动而言，支持 `socksProxy` 参数，于是上述配置的 `jdbcUrl` 可以修改为

`jdbc:presto://127.0.0.1:8080/hive?socksProxy=192.168.1.101:1081`

大部分关系型数据库的 JDBC 驱动支持 `socksProxyHost,socksProxyPort` 参数来支持代理访问。也有一些特别的情况。

以下是各类数据库 JDBC 驱动所支持的代理类型以及配置方式

driver

大部分情况下，一个数据库的 JDBC 驱动是固定的，但有些因为版本的不同，所建议的驱动类名不同，比如 MySQL。新的 MySQL JDBC 驱动类型推荐使用 `com.mysql.cj.jdbc.Driver` 而不是以前的 `com.mysql.jdbc.Drver`。如果想要使用就的驱动名称，则可以配置 `driver` 配置项。

column

所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。用户使用 `*` 代表默认使用所有列配置，例如 `["*"]`。

支持列裁剪，即列可以挑选部分列进行导出。

支持列换序，即列可以不按照表 schema 信息进行导出。

支持常量配置，用户需要按照 JSON 格式:

```
["id", "`table`", "1", "'bazhen.csy'", "null", "to_char(a + 1)", "2.3" ,
"true"]
```

- `id` 为普通列名
- `` `table` `` 为包含保留在的列名，
- `1` 为整形数字常量，
- `'bazhen.csy'` 为字符串常量
- `null` 为空指针，注意，这里的 `null` 必须以字符串形式出现，即用双引号引用
- `to_char(a + 1)` 为表达式，
- `2.3` 为浮点数，
- `true` 为布尔值，同样的，这里的布尔值也必须用双引号引用

Column 必须显示填写，不允许为空！

splitPk

RdbmsReader 进行数据抽取时，如果指定 splitPk，表示用户希望使用 splitPk 代表的字段进行数据分片，Addax 因此会启动并发任务进行数据同步，这样可以大大提供数据同步的效能。

推荐 splitPk 用户使用表主键，因为表主键通常情况下比较均匀，因此切分出来的分片也不容易出现数据热点。

目前 splitPk 仅支持整形、字符串型数据 (ASCII 类型) 切分，不支持浮点、日期等其他类型。如果用户指定其他非支持类型，RDBMSReader 将报错！

splitPk 如果不填写，将视作用户不对单表进行切分，RDBMSReader 使用单通道同步全量数据。

autoPk

从 3.2.6 版本开始，支持自动获取表主键或唯一索引，如果设置为 true ，RdbmsReader 将尝试通过查询数据库的元数据信息获取指定表的主键字段或唯一索引字段，如果获取可用于分隔的字段不止一个，则默认取第一个。后续将会考虑优先取整数类型。

该特性目前支持的数据库有：

- ClickHouse
- MySQL
- Oracle
- PostgreSQL
- SQL Server

3.3 类型转换

目前 RDBMSReader 支持大部分通用得关系数据库类型如数字、字符等，但也存在部分个别类型没有支持的情况，请注意检查你的类型，根据具体的数据库做选择。

2.19.4 4. 当前支持的数据库

- PrestoSQL
- TDH Inceptor2
- IBM DB2
- Apache Hive

2.20 RedisReader 插件文档

2.20.1 1 快速介绍

RedisReader 提供了读取 Redis RDB 的能力。在底层实现上获取本地 RDB 文件/Redis Server 数据，并转换为 Addax 传输协议传递给 Writer。

2.20.2 2 功能与限制

1. 支持读取本地 RDB/http RDB/redis server RDB 的文件并转换成 redis dump 格式。
2. 支持过滤 DB/key 名称过滤

我们暂时不能做到：

1. 单个 RDB 支持多线程并发读取。
2. Redis Server 未开启 sync 命令。
3. 读取 rdb 文件转换成 json 数据。

2.20.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "redisreader",
          "parameter": {
            "connection": [
              {
                "uri": "file:///root/dump.rdb",
                "uri": "http://localhost/dump.rdb",
                "uri": "tcp://127.0.0.1:7001",
                "uri": "tcp://127.0.0.1:7002",
                "uri": "tcp://127.0.0.1:7003",
                "auth": "password"
              }
            ],
            "include": [
              "^user"
            ],
            "exclude": [
              "^password"
            ],
            "db": [
              0,
              1
            ]
          }
        },
        "writer": {
          "name": "rediswriter",
          "parameter": {
            "connection": [
              {
                "uri": "tcp://127.0.0.1:6379",
                "auth": "123456"
              }
            ],
            "timeout": 60000
          }
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 1
      }
    }
  }
}
```

3.2 参数说明

2.20.4 5 约束限制

1. 不支持直接读取任何不支持 sync 命令的 redis server，如果需要请备份的 rdb 文件进行读取。
2. 如果是原生 redis cluster 集群，请填写所有 master 节点的 tcp 地址，redisreader 插件会自动 dump 所有节点的 rdb 文件。
3. 仅解析 `String` 数据类型，其他复合类型 (`Sets`, `List` 等会忽略)

2.21 SqlServerReader 插件文档

2.21.1 1 快速介绍

SqlServerReader 插件实现了从 SqlServer 读取数据。在底层实现上，SqlServerReader 通过 JDBC 连接远程 SqlServer 数据库，并执行相应的 sql 语句将数据从 SqlServer 库中 SELECT 出来。

2.21.2 2 实现原理

简而言之，SqlServerReader 通过 JDBC 连接器连接到远程的 SqlServer 数据库，并根据用户配置的信息生成查询 SELECT SQL 语句并发送到远程 SqlServer 数据库，并将该 SQL 执行返回结果使用 Addax 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

对于用户配置 Table、Column、Where 的信息，SqlServerReader 将其拼接为 SQL 语句发送到 SqlServer 数据库；对于用户配置 querySql 信息，SqlServer 直接将其发送到 SqlServer 数据库。

2.21.3 3 功能说明

3.1 配置样例

配置一个从 SqlServer 数据库同步抽取数据到本地的作业:

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": -1,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "sqlserverreader",
          "parameter": {
            "username": "root",
            "password": "root",
            "column": [
              "*"
            ],
            "splitPk": "db_id",
```

(下页继续)

(续上页)

```
            "connection": [
              {
                "table": [
                  "table"
                ],
                "jdbcUrl": [
                  "jdbc:sqlserver://localhost:3433;DatabaseName=dbname"
                ]
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true,
            "encoding": "UTF-8"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

3.3 类型转换

目前 SqlServerReader 支持大部分 SqlServer 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 SqlServerReader 针对 SqlServer 类型转换列表:

请注意:

- 除上述罗列字段类型外，其他类型均不支持
- timestamp 类型作为二进制类型

2.22 StreamReader

StreamReader 是一个从内存读取数据的插件，他主要用来快速生成期望的数据并对写入插件进行测试

一个完整的 StreamReader 配置文件如下:

```
{
  "reader": {
    "name": "streamreader",
    "parameter": {
      "column": [
        {
          "value": "unique_id",
          "type": "string"
        },
```

(下页继续)

(续上页)

```
        {
          "value": "1989-06-04 08:12:13",
          "type": "date",
          "dateFormat": "yyyy-MM-dd HH:mm:ss"
        },
        {
          "value": 1984,
          "type": "long"
        },
        {
          "value": 1989.64,
          "type": "double"
        },
        {
          "value": true,
          "type": "bool"
        },
        {
          "value": "a long text",
          "type": "bytes"
        }
      ],
      "sliceRecordCount": 10
    }
  }
}
```

上述配置文件将会生成 10 条记录（假定 channel 为 1），每条记录的内容如下：

```
unique_id,'1989-06-04 08:12:13',1984,1989.64,true,'a long text'
```

目前 StreamReader 支持的输出数据类型全部列在上面，分别是：

- string 字符类型
- date 日期类型
- long 所有整型类型
- double 所有浮点数
- bool 布尔类型
- bytes 字节类型

其中 date 类型还支持 `dateFormat` 配置，用来指定输入的日期的格式，默认为 `yyyy-MM-dd HH:mm:ss`。比如你的输入可以这样：

```
{
  "value": "1989/06/04 12:13:14",
  "type": "date",
  "dateFormat": "yyyy/MM/dd HH:mm:ss"
}
```

注意，日期类型不管输入是何种格式，内部都转为 `yyyy-MM-dd HH:mm:ss` 格式。

StreamReader 还支持随机输入功能，比如我们要随机得到 0-10 之间的任意一个整数，我们可以这样配置列：

```
{
  "random": "0,10",
```

(下页继续)

(续上页)

```
  "type": "long"
}
```

这里使用 `random` 这个关键字来表示其值为随机值，其值的范围为左右闭区间。

其他类型的随机类型配置如下：

- `long`: random 0, 10 0 到 10 之间的随机数字
- `string`: random 0, 10 0 到 10 长度之间的随机字符串
- `bool`: random 0, 10 false 和 true 出现的比率
- `double`: random 0, 10 0 到 10 之间的随机浮点数
- `date`: random '2014-07-07 00:00:00', '2016-07-07 00:00:00' 开始时间-> 结束时间之间的随机时间，日期格式默认 (不支持逗号)yyyy-MM-dd HH:mm:ss
- `BYTES`: random 0, 10 0 到 10 长度之间的随机字符串获取其 UTF-8 编码的二进制串

StreamReader 还支持递增函数，比如我们要得到一个从 1 开始，每次加 5 的等差数列，可以这样配置：

```
{
  "incr": "1,5",
  "type": "long"
}
```

如果需要获得一个递减的数列，则把第二个参数的步长（上例中的 5）改为负数即可。步长默认值为 1。

递增还支持日期类型 (`4.0.1` 版本引入)，比如下面的配置：

```
{
  "incr": "1989-06-04 09:01:02,2,d",
  "type": "date"
}
```

`incr` 由三部分组成，分别是开始日期，步长以及步长单位，中间用英文逗号 (,) 分隔。

- 开始日期：正确的日期字符串，默认格式为 `yyyy-MM-dd hh:mm:ss`，如果时间格式不同，则需要配置 `dateFormat` 来指定日期格式，这是必填项
- 步长：每次需要增加的长度，默认为 1，如果希望是递减，则填写负数，这是可选项
- 步长单位：按什么时间单位进行递增/递减，默认为按天（day），这是可选项，可选的单位有
  - d/day
  - M/month
  - y/year
  - h/hour
  - m/minute
  - s/second
  - w/week

配置项 `sliceRecordCount` 用来指定要生成的数据条数，如果指定的 `channel`，则实际生成的记录数为 `sliceRecordCount * channel`

2.23 TDengineReader

TDengineReader 插件实现了从涛思公司的 TDengine 读取数据。在底层实现上，TDengineReader 通过 JDBC JNI 驱动连接远程 TDengine 数据库，并执行相应的 sql 语句将数据从 TDengine 库中批量获。

不同于其他关系型数据库，TDengine 不支持 FetchSize

2.23.1 前置条件

考虑到性能问题，该插件使用了 TDengine 的 JDBC-JNI 驱动，该驱动直接调用客户端 API（libtaos.so 或 taos.dll）将写入和查询请求发送到 taosd 实例。因此在使用之前需要配置好动态库链接文件。

首先将 `plugin/reader/tdenginereader/libs/libtaos.so.2.0.16.0` 拷贝到 `/usr/lib64` 目录，然后执行下面的命令创建软链接

```
ln -sf /usr/lib64/libtaos.so.2.0.16.0 /usr/lib64/libtaos.so.1
ln -sf /usr/lib64/libtaos.so.1 /usr/lib64/libtaos.so
```

2.23.2 示例

TDengine 数据自带了一个演示数据库 taosdemo，我们从演示数据库读取部分数据并打印到终端

以下是配置文件

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 3
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [
      {
        "reader": {
          "name": "tdenginereader",
          "parameter": {
            "username": "root",
            "password": "taosdata",
            "connection": [
              {
                "jdbcUrl": [
                  "jdbc:TAOS://127.0.0.1:6030/test"
                ],
                "querySql": [
                  "select * from test.meters where ts <'2017-07-14 10:40:02' and  loc=
↪'beijing' limit 10"
                ]
              }
            ]
          }
```

（下页继续）

(续上页)

```
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 job/tdengine2stream.json

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/tdengine2stream.json
```

命令输出类似如下：

```
2021-02-20 15:32:23.161 [main] INFO  VMInfo - VMInfo# operatingSystem class => sun.
↪management.OperatingSystemImpl
2021-02-20 15:32:23.229 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "password":"*****",
                                        "connection":[
                                                {
                                                        "querySql":[
                                                                "select * from test.
↪meters where ts <'2017-07-14 10:40:02' and  loc='beijing' limit 100"
                                                        ],
                                                        "jdbcUrl":[
                                                                "jdbc:TAOS://127.0.0.
↪1:6030/test"
                                                        ]
                                                }
                                        ],
                                        "username":"root"
                                },
                                "name":"tdenginereader"
                        },
                        "writer":{
                                "parameter":{
                                        "print":true
                                },
                                "name":"streamwriter"
                        }
                }
        ],
```

(下页继续)

(续上页)

```
        "setting":{
                "errorLimit":{
                        "record":0,
                        "percentage":0.02
                },
                "speed":{
                        "channel":3
                }
        }
}

2021-02-20 15:32:23.277 [main] INFO  PerfTrace - PerfTrace traceId=job_-1,␣
↪isEnable=false, priority=0
2021-02-20 15:32:23.278 [main] INFO  JobContainer - Addax jobContainer starts job.
2021-02-20 15:32:23.281 [main] INFO  JobContainer - Set jobId = 0
java.library.path:/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
....
2021-02-20 15:32:23.687 [0-0-0-reader] INFO  CommonRdbmsReader$Task - Begin to read␣
↪record by Sql: [select * from test.meters where ts <'2017-07-14 10:40:02' and  loc=
↪'beijing' limit 100
] jdbcUrl:[jdbc:TAOS://127.0.0.1:6030/test].
2021-02-20 15:32:23.692 [0-0-0-reader] WARN  DBUtil - current database does not␣
↪supoort TYPE_FORWARD_ONLY/CONCUR_READ_ONLY
2021-02-20 15:32:23.740 [0-0-0-reader] INFO  CommonRdbmsReader$Task - Finished read␣
↪record by Sql: [select * from test.meters where ts <'2017-07-14 10:40:02' and  loc=
↪'beijing' limit 100
] jdbcUrl:[jdbc:TAOS://127.0.0.1:6030/test].

1500000001000        5        5        0        1        beijing
1500000001000        0        6        2        1        beijing
1500000001000        7        0        0        1        beijing
1500000001000        8        9        6        1        beijing
1500000001000        9        9        1        1        beijing
1500000001000        8        2        0        1        beijing
1500000001000        4        5        5        3        beijing
1500000001000        3        3        3        3        beijing
1500000001000        5        4        8        3        beijing
1500000001000        9        4        6        3        beijing

2021-02-20 15:32:26.689 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-02-20 15:32:23
任务结束时刻                    : 2021-02-20 15:32:26
任务总计耗时                    :                  3s
任务平均流量                    :             800B/s
记录写入速度                    :            33rec/s
读出记录总数                    :                100
读写失败总数                    :                  0
```

2.23.3 参数说明

使用 JDBC-RESTful 接口

如果不想依赖本地库，或者没有权限，则可以使用 JDBC-RESTful 接口来写入表，相比 JDBC-JNI 而言，配置区别是：

- driverClass 指定为 com.taosdata.jdbc.rs.RestfulDriver
- jdbcUrl 以 jdbc:TAOS-RS:// 开头；
- 使用 6041 作为连接端口

所以上述配置中的 connection 应该修改为如下：

```
"connection": [{
  "querySql":["select * from test.meters where ts <'2017-07-14 10:40:02' and  loc=
↪'beijing' limit 100"],
  "jdbcUrl": ["jdbc:TAOS-RS://127.0.0.1:6041/test"],
  "driver":"com.taosdata.jdbc.rs.RestfulDriver"
}]
```

2.23.4 类型转换

目前 TDenginereader 支持 TDengine 所有类型，具体如下

2.23.5 当前支持版本

TDengine 2.0.16

2.23.6 注意事项

- TDengine JDBC-JNI 驱动和动态库版本要求一一匹配，因此如果你的数据版本并不是 2.0.16，则需要同时替换动态库和插件目录中的 JDBC 驱动

2.24 Addax TxtFileReader 说明

2.24.1 1 快速介绍

TxtFileReader 提供了读取本地文件系统数据存储的能力。在底层实现上，TxtFileReader 获取本地文件数据，并转换为 Addax 传输协议传递给 Writer。

本地文件内容存放的是一张逻辑意义上的二维表，例如 CSV 格式的文本信息。

2.24.2 2 功能与限制

TxtFileReader 实现了从本地文件读取数据并转为 Addax 协议的功能，本地文件本身是无结构化数据存储，对于 Addax 而言，TxtFileReader 实现上类比 OSSReader，有诸多相似之处。目前 TxtFileReader 支持功能如下：

1. 支持且仅支持读取 TXT 的文件，且要求 TXT 中 shema 为一张二维表。
2. 支持类 CSV 格式文件，自定义分隔符。
3. 支持多种类型数据读取 (使用 String 表示)，支持列裁剪，支持列常量
4. 支持递归读取、支持文件名过滤。
5. 支持文本压缩，且自动猜测压缩格式
6. 多个 File 可以支持并发读取。

我们暂时不能做到：

1. 单个 File 支持多线程并发读取，这里涉及到单个 File 内部切分算法。二期考虑支持。
2. 单个 File 在压缩情况下，从技术上无法支持多线程并发读取。

2.24.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "txtfilereader",
          "parameter": {
            "path": [
              "/tmp/data"
            ],
            "encoding": "UTF-8",
            "column": [
              {
                "index": 0,
                "type": "long"
              },
              {
                "index": 1,
                "type": "boolean"
              },
              {
                "index": 2,
                "type": "double"
              },
              {
                "index": 3,
```

(下页继续)

(续上页)

```
                "type": "string"
              },
              {
                "index": 4,
                "type": "date",
                "format": "yyyy.MM.dd"
              }
            ],
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "txtfilewriter",
          "parameter": {
            "path": "/tmp/result",
            "fileName": "txt_",
            "writeMode": "truncate",
            "format": "yyyy-MM-dd"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

path

本地文件系统的路径信息，注意这里可以支持填写多个路径。

- 当指定单个本地文件，TxtFileReader 暂时只能使用单线程进行数据抽取。二期考虑在非压缩文件情况下针对单个 File 可以进行多线程并发读取
- 当指定多个本地文件，TxtFileReader 支持使用多线程进行数据抽取。线程并发数通过通道数指定
- 当指定通配符，TxtFileReader 尝试遍历出多个文件信息。例如: 指定 `/*` 代表读取 / 目录下所有的文件，指定 `/bazhen/*` 代表读取 bazhen 目录下游所有的文件。目前只支持 `*` 作为文件通配符。

特别需要注意的是，Addax 会将一个作业下同步的所有 Text File 视作同一张数据表。用户必须自己保证所有的 File 能够适配同一套 schema 信息。读取文件用户必须保证为类 CSV 格式，并且提供给 Addax 权限可读。

特别需要注意的是，如果 Path 指定的路径下没有符合匹配的文件抽取，Addax 将报错。

从 3.2.3 版本起，`path` 下允许混合不同压缩格式的文件，插件会尝试自动猜测压缩格式并自动解压，目前支持的压缩格式有:

- zip
- bzip2
- gzip
- LZ4
- PACK200
- XZ
- Compress

column

读取字段列表，type 指定源数据的类型，index 指定当前列来自于文本第几列 (以 0 开始)，value 指定当前类型为常量，不从源头文件读取数据，而是根据 value 值自动生成对应的列。

默认情况下，用户可以全部按照 String 类型读取数据，配置如下：

```
{
  "column": [
    "*"
  ]
}
```

用户可以指定 Column 字段信息，配置如下：

```
[
  {
    "type": "long",
    "index": 0
  },
  {
    "type": "string",
    "value": "alibaba"
  }
]
```

对于用户指定 Column 信息，type 必须填写，index/value 必须选择其一。

从 `4.0.1` 开始，表示字段除了使用 `index` 来指定字段的顺序外，还支持 `name` 方式，这需要所读取的文件的都包含了文件头，插件会尝试将指定的 `name` 去匹配从文件读取的文件头，然后得到对应的 `index` 值，并回写到配置文件中。同时，`index` 和 `name` 可以在不同的列上进行混合使用，比如下面这样：

```
[
  {
    "type": "long",
    "index": 0
  },
  {
    "name": "region",
    "type": "string"
  },
  {
    "type": "string",
    "value": "alibaba"
  }
]
```

注：这种方式以为这在准备阶段就要尝试读取文件，因为会有一定的性能损失，如非必要，不建议配置 `name` 方式。

csvReaderConfig

读取 CSV 类型文件参数配置，Map 类型。读取 CSV 类型文件使用的 CsvReader 进行读取，会有很多配置，不配置则使用默认值。

常见配置：

```
{
  "csvReaderConfig": {
    "safetySwitch": false,
    "skipEmptyRecords": false,
    "useTextQualifier": false
  }
}
```

所有配置项及默认值，配置时 csvReaderConfig 的 map 中请**严格按照以下字段名字进行配置**：

```
boolean caseSensitive = true;
char textQualifier = 34;
boolean trimWhitespace = true;
boolean useTextQualifier = true;//是否使用 csv 转义字符
char delimiter = 44;//分隔符
char recordDelimiter = 0;
char comment = 35;
boolean useComments = false;
int escapeMode = 1;
boolean safetySwitch = true;//单列长度是否限制 100000 字符
boolean skipEmptyRecords = true;//是否跳过空行
boolean captureRawRecord = true;
```

3.3 类型转换

本地文件本身不提供数据类型，该类型是 Addax TxtFileReader 定义：

其中：

- Long 是指本地文件文本中使用整形的字符串表示形式，例如”19901219”。
- Double 是指本地文件文本中使用 Double 的字符串表示形式，例如”3.1415”。
- Boolean 是指本地文件文本中使用 Boolean 的字符串表示形式，例如”true”、”false”。不区分大小写。
- Date 是指本地文件文本中使用 Date 的字符串表示形式，例如”2014-12-31”，Date 可以指定 format 格式。

CHAPTER 3

写入插件

本章描述 Addax 目前支持的数据写入插件

3.1 CassandraWriter 插件文档

3.1.1 1 快速介绍

CassandraWriter 插件实现了向 Cassandra 写入数据。在底层实现上，CassandraWriter 通过 datastax 的 java driver 连接 Cassandra 实例，并执行相应的 cql 语句将数据写入 cassandra 中。

3.1.2 2 实现原理

简而言之，CassandraWriter 通过 java driver 连接到 Cassandra 实例，并根据用户配置的信息生成 INSERT CQL 语句，然后发送到 Cassandra。

对于用户配置 Table、Column 的信息，CassandraReader 将其拼接为 CQL 语句发送到 Cassandra。

3.1.3 3 功能说明

3.1 配置样例

配置一个从内存产生到 Cassandra 导入的作业:

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 5,
        "bytes": -1
```

(下页继续)

(续上页)

```
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "name",
                "type": "string"
              },
              {
                "value": "false",
                "type": "bool"
              },
              {
                "value": "1988-08-08 08:08:08",
                "type": "date"
              },
              {
                "value": "addr",
                "type": "bytes"
              },
              {
                "value": 1.234,
                "type": "double"
              },
              {
                "value": 12345678,
                "type": "long"
              },
              {
                "value": 2.345,
                "type": "double"
              },
              {
                "value": 3456789,
                "type": "long"
              },
              {
                "value": "4a0ef8c0-4d97-11d0-db82-ebecdb03ffa5",
                "type": "string"
              },
              {
                "value": "value",
                "type": "bytes"
              },
              {
                "value": "-838383838,37377373,-383883838,27272772,393993939,-38383883,
↪83883838,-1350403181,817650816,1630642337,251398784,-622020148",
                "type": "string"
              }
            ],
            "sliceRecordCount": 10000000
          }
```

(下页继续)

(续上页)

```
        },
        "writer": {
          "name": "cassandrawriter",
          "parameter": {
            "host": "localhost",
            "port": 9042,
            "useSSL": false,
            "keyspace": "stresscql",
            "table": "dst",
            "batchSize": 10,
            "column": [
              "name",
              "choice",
              "date",
              "address",
              "dbl",
              "lval",
              "fval",
              "ival",
              "uid",
              "value",
              "listval"
            ]
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

3.3 类型转换

目前 CassandraReader 支持除 counter 和 Custom 类型之外的所有类型。

下面列出 CassandraReader 针对 Cassandra 类型转换列表:

请注意:

目前不支持 counter 类型和 custom 类型。

3.1.4 4 约束限制

batchSize

1. 不能超过 65535
2. batch 中的内容大小受到服务器端 `batch_size_fail_threshold_in_kb` 的限制。
3. 如果 batch 中的内容超过了 `batch_size_warn_threshold_in_kb` 的限制，会打出 warn 日志，但并不影响写入，忽略即可。
4. 如果批量提交失败，会把这个批量的所有内容重新逐条写入一遍。

3.2 ClickHouseWriter

ClickHouseWriter 插件实现了写入数据 ClickHouse。在底层实现上，ClickHouseWriter 通过 JDBC 连接远程 ClickHouse 数据库，并执行相应的 `insert into ....` 语句将数据插入到 ClickHouse 库中。

3.2.1 示例

以下示例我们演示从 clickhouse 中读取一张表的内容，并写入到相同表结构的另外一张表中，用来测试插件所支持的数据结构

表结构以数据

假定要读取的表结构及数据如下:

```
CREATE TABLE ck_addax (
    c_int8 Int8,
    c_int16 Int16,
    c_int32 Int32,
    c_int64 Int64,
    c_uint8 UInt8,
    c_uint16 UInt16,
    c_uint32 UInt32,
    c_uint64 UInt64,
    c_float32 Float32,
    c_float64 Float64,
    c_decimal Decimal(38,10),
    c_string String,
    c_fixstr FixedString(36),
    c_uuid UUID,
    c_date Date,
    c_datetime DateTime('Asia/Chongqing'),
    c_datetime64 DateTime64(3, 'Asia/Chongqing'),
    c_enum Enum('hello' = 1, 'world'=2)
) ENGINE = MergeTree() ORDER BY (c_int8, c_int16) SETTINGS index_granularity = 8192;

insert into ck_addax values(
    127,
    -32768,
    2147483647,
    -9223372036854775808,
    255,
    65535,
    4294967295,
    18446744073709551615,
    0.999999999999,
    0.99999999999999999,
    123456789123456789123456789.1234567891,
    'Hello String',
    '2c:16:db:a3:3a:4f',
    '5F042A36-5B0C-4F71-ADFD-4DF4FCA1B863',
    '2021-01-01',
    '2021-01-01 00:00:00',
    '2021-01-01 00:00:00',
    'hello'
);
```

要写入的表采取和读取表结构相同，其建表语句如下：

```
create table ck_addax_writer as ck_addax;
```

3.2.2 配置

以下为配置文件

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      }
    },
    "content": [
      {
        "writer": {
          "name": "clickhousewriter",
          "parameter": {
            "username": "default",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "ck_addax_writer"
                ],
                "jdbcUrl": "jdbc:clickhouse://127.0.0.1:8123/default"
              }
            ],
            "preSql": ["alter table @table delete where 1=1"]
          }
        },
        "reader": {
          "name": "clickhousereader",
          "parameter": {
            "username": "default",
            "column": [
              "*"
            ],
            "connection": [
              {
                "jdbcUrl": [
                  "jdbc:clickhouse://127.0.0.1:8123/"
                ],
                "table":["ck_addax"]
              }
            ]
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/clickhouse2clickhouse.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/clickhouse2clickhouse.json
```

3.2.3 参数说明

3.3 DbfFileWriter 插件文档

3.3.1 1 快速介绍

DbfFileWriter 提供了向本地文件写入类 dbf 格式的一个或者多个表文件。DbfFileWriter 服务的用户主要在于 Addax 开发、测试同学。

写入本地文件内容存放的是一张 dbf 表，例如 dbf 格式的文件信息。

3.3.2 2 功能说明

2.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "bytes": -1,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "value": 19880808,
                "type": "long"
              },
              {
                "value": "1989-06-04 00:00:00",
                "type": "date"
              },
              {
                "value": true,
```

(下页继续)

(续上页)

```
                "type": "bool"
              },
              {
                "value": " 中文测试",
                "type": "string"
              }
            ],
            "sliceRecordCount": 10
          }
        },
        "writer": {
          "name": "dbffilewriter",
          "parameter": {
            "column": [
              {
                "name": "col1",
                "type": "char",
                "length": 100
              },
              {
                "name": "col2",
                "type": "numeric",
                "length": 18,
                "scale": 0
              },
              {
                "name": "col3",
                "type": "date"
              },
              {
                "name": "col4",
                "type": "logical"
              },
              {
                "name": "col5",
                "type": "char",
                "length": 100
              }
            ],
            "fileName": "test.dbf",
            "path": "/tmp/out",
            "writeMode": "truncate",
            "encoding": "GBK"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

writeMode

DbfFileWriter 写入前数据清理处理模式：

- truncate，写入前清理目录下一 fileName 前缀的所有文件。
- append，写入前不做任何处理，Addax DbfFileWriter 直接使用 filename 写入，并保证文件名不冲突。
- nonConflict，如果目录下有 fileName 前缀的文件，直接报错。

3.3 类型转换

当前该插件支持写入的类型以及对应关系如下：

其中：

- numeric 是指本地文件中使用数字类型表示形式，例如”19901219”, 整形小数位数为 0。
- logical 是指本地文件文本中使用 Boolean 的表示形式，例如”true”、”false”。
- Date 是指本地文件文本中使用 Date 表示形式，例如”2014-12-31”，Date 是 JAVA 语言的 DATE 类型。

3.4 DorisWriter

DorisWriter 插件实现了以流式方式加载数据到 Doris，其实现上是通过访问 Doris http 连接 (8030)，然后通过 stream load 加载数据到数据中，相比 `insert into` 方式效率要高不少，也是官方推荐的生产环境下的数据加载方式。

Doris 是一个兼容 MySQL 协议的数据库后端，因此读取 Doris 可以使用 MySQLReader 进行访问。

3.4.1 示例

假定要写入的表的建表语句如下：

```
CREATE DATABASE example_db;
CREATE TABLE example_db.table1
(
    siteid INT DEFAULT '10',
    citycode SMALLINT,
    username VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, citycode, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

下面配置一个从内存读取数据，然后写入到 doris 表的配置文件

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2
```

(下页继续)

(续上页)

```
      }
    },
    "content": [
      {
        "writer": {
          "name": "doriswriter",
          "parameter": {
            "username": "test",
            "password": "123456",
            "batchSize": 1024,
            "connection": [
              {
                "table": "table1",
                "database": "example_db",
                "endpoint": "http://127.0.0.1:8030/"
              }
            ]
          }
        },
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "random": "1,500",
                "type": "long"
              },
              {
                "random": "1,127",
                "type": "long"
              },
              {
                "value": "this is a text",
                "type": "string"
              },
              {
                "random": "5,200",
                "type": "long"
              }
            ],
            "sliceRecordCount": 100
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/stream2doris.json`

执行下面的命令

```
bin/addax.sh job/stream2doris.json
```

输出类似如下：

```
2021-02-23 15:22:57.851 [main] INFO  VMInfo - VMInfo# operatingSystem class => sun.
↪management.OperatingSystemImpl
2021-02-23 15:22:57.871 [main] INFO  Engine -
{
"content":[
{
"reader":{
    "parameter":{
            "column":[
                    {
                            "random":"1,500",
                            "type":"long"
                    },
                    {
                            "random":"1,127",
                            "type":"long"
                    },
                    {
                            "type":"string",
                            "value":"username"
                    }
            ],
            "sliceRecordCount":100
    },
    "name":"streamreader"
},
"writer":{
    "parameter":{
            "password":"*****",
            "batchSize":1024,
            "connection":[
                    {
                            "database":"example_db",
                            "endpoint":"http://127.0.0.1:8030/",
                            "table":"table1"
                    }
            ],
            "username":"test"
    },
    "name":"doriswriter"
}
}
],
"setting":{
"speed":{
"channel":2
}
}
}

2021-02-23 15:22:57.886 [main] INFO  PerfTrace - PerfTrace traceId=job_-1,␣
↪isEnable=false, priority=0
2021-02-23 15:22:57.886 [main] INFO  JobContainer - Addax jobContainer starts job.
2021-02-23 15:22:57.920 [job-0] INFO  JobContainer - Scheduler starts [1] taskGroups.
2021-02-23 15:22:57.928 [taskGroup-0] INFO  TaskGroupContainer - taskGroupId=[0]␣
↪start [2] channels for [2] tasks.
2021-02-23 15:22:57.935 [taskGroup-0] INFO  Channel - Channel set byte_speed_limit to␣
↪-1, No bps activated.
```

(下页继续)

(续上页)

```
2021-02-23 15:22:57.936 [taskGroup-0] INFO  Channel - Channel set record_speed_limit␣
↪to -1, No tps activated.
2021-02-23 15:22:57.970 [0-0-1-writer] INFO  DorisWriterTask - connect DorisDB with␣
↪http://127.0.0.1:8030//api/example_db/table1/_stream_load
2021-02-23 15:22:57.970 [0-0-0-writer] INFO  DorisWriterTask - connect DorisDB with␣
↪http://127.0.0.1:8030//api/example_db/table1/_stream_load

2021-02-23 15:23:00.941 [job-0] INFO  JobContainer - PerfTrace not enable!
2021-02-23 15:23:00.946 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-02-23 15:22:57
任务结束时刻                    : 2021-02-23 15:23:00
任务总计耗时                    :                  3s
任务平均流量                    :            1.56KB/s
记录写入速度                    :             66rec/s
读出记录总数                    :                 200
读写失败总数                    :                   0
```

3.4.2 参数说明

column

该插件中的 column 不是必须项，如果没有配置该项，或者配置为 ["*"]，则按照 reader 插件获取的字段值进行顺序拼装。否则可以按照如下方式指定需要插入的字段

```
{
  "column": ["siteid","citycode","username"]
}
```

3.5 ElasticSearchWriter 插件文档

3.5.1 1 快速介绍

数据导入 elasticsearch 的插件

3.5.2 2 实现原理

使用 elasticsearch 的 rest api 接口，批量把从 reader 读入的数据写入 elasticsearch

3.5.3 3 功能说明

3.1 配置样例

job.json

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "random": "10,1000",
                "type": "long"
              },
              {
                "value": "1.1.1.1",
                "type": "string"
              },
              {
                "value": 19890604.0,
                "type": "double"
              },
              {
                "value": 19890604,
                "type": "long"
              },
              {
                "value": 19890604,
                "type": "long"
              },
              {
                "value": "hello world",
                "type": "string"
              },
              {
                "value": "long text",
                "type": "string"
              },
              {
                "value": "41.12,-71.34",
                "type": "string"
              },
              {
                "value": "2017-05-25 11:22:33",
                "type": "string"
              }
            ],
            "sliceRecordCount": 100
          }
        },
        "writer": {
          "name": "elasticsearchwriter",
          "parameter": {
```

(下页继续)

(续上页)

```
            "endpoint": "http://localhost:9200",
            "index": "test-1",
            "type": "default",
            "cleanup": true,
            "settings": {
              "index": {
                "number_of_shards": 1,
                "number_of_replicas": 0
              }
            },
            "discovery": false,
            "batchSize": 1000,
            "splitter": ",",
            "column": [
              {
                "name": "pk",
                "type": "id"
              },
              {
                "name": "col_ip",
                "type": "ip"
              },
              {
                "name": "col_double",
                "type": "double"
              },
              {
                "name": "col_long",
                "type": "long"
              },
              {
                "name": "col_integer",
                "type": "integer"
              },
              {
                "name": "col_keyword",
                "type": "keyword"
              },
              {
                "name": "col_text",
                "type": "text",
                "analyzer": "ik_max_word"
              },
              {
                "name": "col_geo_point",
                "type": "geo_point"
              },
              {
                "name": "col_date",
                "type": "date",
                "format": "yyyy-MM-dd HH:mm:ss"
              },
              {
                "name": "col_nested1",
                "type": "nested"
              },
```

(下页继续)

(续上页)

```
                {
                  "name": "col_nested2",
                  "type": "nested"
                },
                {
                  "name": "col_object1",
                  "type": "object"
                },
                {
                  "name": "col_object2",
                  "type": "object"
                },
                {
                  "name": "col_integer_array",
                  "type": "integer",
                  "array": true
                },
                {
                  "name": "col_geo_shape",
                  "type": "geo_shape",
                  "tree": "quadtree",
                  "precision": "10m"
                }
              ]
            }
          }
        }
      ]
    }
}
```

3.2 参数说明

3.5.4 4 约束限制

- 如果导入 id，这样数据导入失败也会重试，重新导入也仅仅是覆盖，保证数据一致性
- 如果不导入 id，就是 append_only 模式，elasticsearch 自动生成 id，速度会提升 20% 左右，但数据无法修复，适合日志型数据 (对数据精度要求不高的)

3.6 FtpWriter

FtpWriter 提供了向远程 FTP 或 SFTP 服务写入文件的能力，当前仅支持写入文本文件。

3.6.1 示例

配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {},
        "writer": {
          "name": "ftpwriter",
          "parameter": {
            "protocol": "sftp",
            "host": "***",
            "port": 22,
            "username": "xxx",
            "password": "xxx",
            "timeout": "60000",
            "connectPattern": "PASV",
            "path": "/tmp/data/",
            "fileName": "test",
            "writeMode": "truncate|append|nonConflict",
            "fieldDelimiter": ",",
            "encoding": "UTF-8",
            "nullFormat": "null",
            "dateFormat": "yyyy-MM-dd",
            "fileFormat": "csv",
            "useKey": false,
            "keyPath": "",
            "keyPass": "",
            "header": []
          }
        }
      }
    ]
  }
}
```

3.6.2 参数说明

writeMode

描述：FtpWriter 写入前数据清理处理模式：

1. `truncate`，写入前清理目录下一 fileName 前缀的所有文件。
2. `append`，写入前不做任何处理，Addax FtpWriter 直接使用 filename 写入，并保证文件名不冲突。
3. `nonConflict`，如果目录下有 fileName 前缀的文件，直接报错。

认证

从 `4.0.2` 版本开始，支持私钥认证方式登录 SFTP 服务器，如果密码和私有都填写了，则两者认证方式都会尝试。注意，如果填写了 `keyPath`, `keyPass` 项，但 `useKey` 设置为 `false` ，插件依然不会尝试用私钥进行登录。

3.6.3 类型转换

FTP 文件本身不提供数据类型，该类型是 Addax FtpWriter 定义：

其中：

- Long 是指 FTP 文件文本中使用整形的字符串表示形式，例如 `19901219`。
- Double 是指 FTP 文件文本中使用 Double 的字符串表示形式，例如 `3.1415`。
- Boolean 是指 FTP 文件文本中使用 Boolean 的字符串表示形式，例如 `true`、`false`。不区分大小写。
- Date 是指 FTP 文件文本中使用 Date 的字符串表示形式，例如 `2014-12-31 12:13:14`，Date 可以指定 format 格式。

3.7 Hbase11XWriter 插件文档

3.7.1 1 快速介绍

HbaseWriter 插件实现了从向 Hbase 中写取数据。在底层实现上，HbaseWriter 通过 HBase 的 Java 客户端连接远程 HBase 服务，并通过 put 方式写入 Hbase。

1.1 支持功能

- 目前 HbaseWriter 支持源端多个字段拼接作为 hbase 表的 rowkey，具体配置参考：rowkeyColumn 配置；
- 写入 hbase 的时间戳（版本）支持：用当前时间作为版本，指定源端列作为版本，指定一个时间三种方式作为版本；

1.2 限制

1. 目前只支持源端为横表写入，不支持竖表（源端读出的为四元组: `rowKey`, `family:qualifier`, `timestamp`, `value`）模式的数据写入；
2. 目前不支持写入 hbase 前清空表数据

3.7.2 2 实现原理

简而言之，HbaseWriter 通过 HBase 的 Java 客户端，通过 HTable, Put 等 API，将从上游 Reader 读取的数据写入 HBase 你 hbase11xwriter 与 hbase094xwriter 的主要不同在于 API 的调用不同，

3.7.3 3 功能说明

3.1 配置样例

配置一个从本地写入 hbase1.1.x 的作业：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 5,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "txtfilereader",
          "parameter": {
            "path": "/tmp/normal.txt",
            "charset": "UTF-8",
            "column": [
              {
                "index": 0,
                "type": "String"
              },
              {
                "index": 1,
                "type": "string"
              },
              {
                "index": 2,
                "type": "string"
              },
              {
                "index": 3,
                "type": "string"
              },
              {
                "index": 4,
                "type": "string"
              },
              {
```

(下页继续)

(续上页)

```
                "index": 5,
                "type": "string"
              },
              {
                "index": 6,
                "type": "string"
              }
            ],
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "hbase11xwriter",
          "parameter": {
            "hbaseConfig": {
              "hbase.zookeeper.quorum": "***"
            },
            "table": "writer",
            "mode": "normal",
            "rowkeyColumn": [
              {
                "index": 0,
                "type": "string"
              },
              {
                "index": -1,
                "type": "string",
                "value": "_"
              }
            ],
            "column": [
              {
                "index": 1,
                "name": "cf1:q1",
                "type": "string"
              },
              {
                "index": 2,
                "name": "cf1:q2",
                "type": "string"
              },
              {
                "index": 3,
                "name": "cf1:q3",
                "type": "string"
              },
              {
                "index": 4,
                "name": "cf2:q1",
                "type": "string"
              },
              {
                "index": 5,
                "name": "cf2:q2",
                "type": "string"
              },
```

(下页继续)

(续上页)

```
                {
                  "index": 6,
                  "name": "cf2:q3",
                  "type": "string"
                }
              ],
              "versionColumn": {
                "index": -1,
                "value": "123456789"
              },
              "encoding": "utf-8"
            }
          }
        }
      ]
    }
}
```

3.2 参数说明

column

要写入的 hbase 字段。index：指定该列对应 reader 端 column 的索引，从 0 开始；name：指定 hbase 表中的列，必须为列族: 列名的格式；type：指定写入数据类型，用于转换 HBase byte[]。配置格式如下：

```
"column": [
{
"index": 1,
"name": "cf1:q1",
"type": "string"
},
{
"index": 2,
"name": "cf1:q2",
"type": "string"
}
]
```

rowkeyColumn

要写入的 hbase 的 rowkey 列。index：指定该列对应 reader 端 column 的索引，从 0 开始，若为常量 index 为 -1；type：指定写入数据类型，用于转换 HBase byte[]；value：配置常量，常作为多个字段的拼接符。hbasewriter 会将 rowkeyColumn 中所有列按照配置顺序进行拼接作为写入 hbase 的 rowkey，不能全为常量。配置格式如下：

```
"rowkeyColumn": [
{
"index": 0,
"type": "string"
},
{
"index": -1,
```

(下页继续)

(续上页)

```
"type": "string",
"value": "_"
}
]
```

versionColumn

指定写入 hbase 的时间戳。支持：当前时间、指定时间列，指定时间，三者选一。若不配置表示用当前时间。

index：指定对应 reader 端 column 的索引，从 0 开始，需保证能转换为 long, 若是 Date 类型，会尝试用 `yyyy-MM-dd HH:mm:ss` 和 `yyyy-MM-dd HH:mm:ss SSS` 去解析；若为指定时间 index 为 `-1`；

value：指定时间的值,long 值。配置格式如下：

```
"versionColumn":{
"index": 1
}
```

或者

```
"versionColumn":{
"index": -1,
"value": 123456789
}
```

3.3 HBase 支持的列类型

- BOOLEAN
- SHORT
- INT
- LONG
- FLOAT
- DOUBLE
- STRING

请注意: 除上述罗列字段类型外，其他类型均不支持

3.8 HBase11xsqlwriter 插件文档

3.8.1 1. 快速介绍

HBase11xsqlwriter 实现了向 hbase 中的 SQL 表 (phoenix) 批量导入数据的功能。Phoenix 因为对 rowkey 做了数据编码，所以，直接使用 HBaseAPI 进行写入会面临手工数据转换的问题，麻烦且易错。本插件提供了单间的 SQL 表的数据导入方式。

在底层实现上，通过 Phoenix 的 JDBC 驱动，执行 UPSERT 语句向 hbase 写入数据。

1.1 支持的功能

支持带索引的表的数据导入，可以同步更新所有的索引表

1.2 限制

- 仅支持 1.x 系列的 hbase
- 仅支持通过 phoenix 创建的表，不支持原生 HBase 表
- 不支持带时间戳的数据导入

3.8.2 2. 实现原理

通过 Phoenix 的 JDBC 驱动，执行 UPSERT 语句向表中批量写入数据。因为使用上层接口，所以，可以同步更新索引表。

3.8.3 3. 配置说明

3.1 配置样例

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "txtfilereader",
          "parameter": {
            "path": "/tmp/normal.txt",
            "charset": "UTF-8",
            "column": [
              {
                "index": 0,
                "type": "String"
              },
              {
                "index": 1,
                "type": "string"
              },
              {
                "index": 2,
                "type": "string"
              },
              {
                "index": 3,
                "type": "string"
              }
            ],
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "hbase11xsqlwriter",
```

(下页继续)

(续上页)

```
          "parameter": {
            "batchSize": "256",
            "column": [
              "UID",
              "TS",
              "EVENTID",
              "CONTENT"
            ],
            "haveKerberos": "true",
            "kerberosPrincipal": "hive@EXAMPLE.COM",
            "kerberosKeytabFilePath": "/tmp/hive.headless.keytab",
            "hbaseConfig": {
              "hbase.zookeeper.quorum": "node1,node2,node3:2181",
              "zookeeper.znode.parent": "/hbase-secure"
            },
            "nullMode": "skip",
            "table": "TEST_TBL"
          }
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 5,
        "bytes": -1
      }
    }
  }
}
```

3.2 参数说明

注意：启用 kerberos 认证后，程序需要知道 `hbase-site.xml` 所在的路径，一种办法是运行执行在环境变量 `CLASSPATH` 中增加该文件的所在路径。

另外一个解决办法是修改 `addax.py` 中的 `CLASS_PATH` 变量，增加 `hbase-site.xml` 的路径

3.9 HBase20xsqlwriter 插件文档

3.9.1 1. 快速介绍

HBase20xsqlwriter 实现了向 hbase 中的 SQL 表 (phoenix) 批量导入数据的功能。Phoenix 因为对 rowkey 做了数据编码，所以，直接使用 HBaseAPI 进行写入会面临手工数据转换的问题，麻烦且易错。本插件提供了 SQL 方式直接向 Phoenix 表写入数据。

在底层实现上，通过 Phoenix QueryServer 的轻客户端驱动，执行 UPSERT 语句向 Phoenix 写入数据。

1.1 支持的功能

支持带索引的表的数据导入，可以同步更新所有的索引表

1.2 限制

1. 要求版本为 Phoenix5.x 及 HBase2.x
2. 仅支持通过 Phoenix QueryServer 导入数据，因此您 Phoenix 必须启动 QueryServer 服务才能使用本插件
3. 不支持清空已有表数据
4. 仅支持通过 phoenix 创建的表，不支持原生 HBase 表
5. 不支持带时间戳的数据导入

3.9.2 2. 实现原理

通过 Phoenix 轻客户端，连接 Phoenix QueryServer 服务，执行 UPSERT 语句向表中批量写入数据。因为使用上层接口，所以，可以同步更新索引表。

3.9.3 3. 配置说明

3.1 配置样例

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "txtfilereader",
          "parameter": {
            "path": "/tmp/normal.txt",
            "charset": "UTF-8",
            "column": [
              {
                "index": 0,
                "type": "String"
              },
              {
                "index": 1,
                "type": "string"
              },
              {
                "index": 2,
                "type": "string"
              },
              {
                "index": 3,
                "type": "string"
              }
            ],
            "fieldDelimiter": ","
          }
```

(下页继续)

(续上页)

```
        },
        "writer": {
          "name": "hbase20xsqlwriter",
          "parameter": {
            "batchSize": "100",
            "column": [
              "UID",
              "TS",
              "EVENTID",
              "CONTENT"
            ],
            "queryServerAddress": "http://127.0.0.1:8765",
            "nullMode": "skip",
            "table": "TEST_TBL"
          }
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 5,
        "bytes": -1
      }
    }
  }
}
```

3.2 参数说明

3.10 Addax HdfsWriter 插件文档

3.10.1 1 快速介绍

HdfsWriter 提供向 HDFS 文件系统指定路径中写入 `TEXTFile`，`ORCFile`，`PARQUET` 文件，文件内容可与 hive 中表关联。

3.10.2 2 功能与限制

1. 目前 HdfsWriter 仅支持 textfile，orcfile，parquet 三种格式的文件，且文件内容存放的必须是一张逻辑意义上的二维表;
2. 由于 HDFS 是文件系统，不存在 schema 的概念，因此不支持对部分列写入;
3. 目前仅支持与以下 Hive 数据类型:
   - 数值型: TINYINT(txt 或 ORC), SMALLINT(txt 或 ORC), INT(orc 或 parquet), INTEGER(txt 或 ORC), BIGINT(txt 或 ORC), LONG(parquet), FLOAT(orc 或 parquet), DOUBLE(orc 或 parquet), DECIMAL(orc 或 TXT), DECIMAL(18.9) (只有 PARQUET 必须带精度)
   - 字符串类型: STRING(TXT/orc 或 parquet),VARCHAR(TXT/orc),CHAR(TXT/orC)
   - 布尔类型: BOOLEAN(TXT/orc 或 parquet)
   - 时间类型: DATE(TXT/orC),TIMESTAMP(TXT/orC)

目前不支持：binary、arrays、maps、structs、union 类型

1. 对于 Hive 分区表目前仅支持一次写入单个分区;
2. 对于 textfile 需用户保证写入 hdfs 文件的分隔符**与在 Hive 上创建表时的分隔符一致**,从而实现写入 hdfs 数据与 Hive 表字段关联;
3. HdfsWriter 实现过程是：首先根据用户指定的 path，在 path 目录下，创建点开头 (.) 的临时目录，创建规则：`.path_<uuid>`；然后将读取的文件写入这个临时目录；全部写入后再将这个临时目录下的文件移动到用户指定目录（在创建文件时保证文件名不重复）;最后删除临时目录。如果在中间过程发生网络中断等情况造成无法与 hdfs 建立连接，需要用户手动删除已经写入的文件和临时目录。
4. 目前插件中 Hive 版本为 3.1.1，Hadoop 版本为 3.1.1，, 在 Hadoop 2.7.x, Hadoop 3.1.x 和 Hive 2.x, Hive 3.1.x 测试环境中写入正常；其它版本理论上都支持，但用于生产之前建议进一步测试；
5. 目前 HdfsWriter 支持 Kerberos 认证

3.10.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "value": 19890604,
                "type": "long"
              },
              {
                "value": "1989-06-04 00:00:00",
                "type": "date"
              },
              {
                "value": true,
                "type": "bool"
              },
              {
                "value": "test",
                "type": "bytes"
              }
            ],
            "sliceRecordCount": 1000
```

(下页继续)

(续上页)

```
            },
            "writer": {
              "name": "hdfswriter",
              "parameter": {
                "defaultFS": "hdfs://xxx:port",
                "fileType": "orc",
                "path": "/user/hive/warehouse/writerorc.db/orcfull",
                "fileName": "xxxx",
                "column": [
                  {
                    "name": "col1",
                    "type": "string"
                  },
                  {
                    "name": "col2",
                    "type": "int"
                  },
                  {
                    "name": "col3",
                    "type": "string"
                  },
                  {
                    "name": "col4",
                    "type": "boolean"
                  },
                  {
                    "name": "col5",
                    "type": "string"
                  }
                ],
                "writeMode": "overwrite",
                "fieldDelimiter": "\u0001",
                "compress": "SNAPPY"
              }
            }
          }
        }
    ]
  }
}
```

3.2 配置项说明

path

存储到 Hadoop hdfs 文件系统的路径信息，HdfsWriter 会根据并发配置在 `Path` 目录下写入多个文件。为与 hive 表关联，请填写 hive 表在 hdfs 上的存储路径。例：Hive 上设置的数据仓库的存储路径为：`/user/hive/warehouse/` ，已建立数据库：`test`，表：`hello`；则对应的存储路径为：`/user/hive/warehouse/test.db/hello` (如果建表时指定了 `location` 属性，则依据该属性的路径)

defaultFS

Hadoop hdfs 文件系统 namenode 节点地址。格式：`hdfs://ip:port`；例如：`hdfs://127.0.0.1:9000`，如果启用了 HA，则为 servicename 模式，比如 `hdfs://sandbox`

fileType

描述：文件的类型，目前只支持用户配置为

- text 表示 Text file 文件格式
- orc 表示 OrcFile 文件格式
- parquet 表示 Parquet 文件格式
- rc 表示 Rcfile 文件格式
- seq 表示 sequence file 文件格式
- csv 表示普通 hdfs 文件格式（逻辑二维表）

fileName

HdfsWriter 写入时的文件名，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。

column

写入数据的字段，不支持对部分列写入。为与 hive 中表关联，需要指定表中所有字段名和字段类型，其中：name 指定字段名，type 指定字段类型。

用户可以指定 column 字段信息，配置如下：

```
{
  "column": [
    {
      "name": "userName",
      "type": "string"
    },
    {
      "name": "age",
      "type": "long"
    },
    {
      "name": "salary",
      "type": "decimal(8,2)"
    }
  ]
}
```

对于数据类型是 `decimal` 类型的，需要注意：

1. 如果没有指定精度和小数位，则使用默认的 `decimal(38,10)` 表示
2. 如果仅指定了精度但未指定小数位，则小数位用 0 表示，即 `decimal(p,0)`
3. 如果都指定，则使用指定的规格，即 `decimal(p,s)`

writeMode

描述：hdfswriter 写入前数据清理处理模式：

- append，写入前不做任何处理，Addax hdfswriter 直接使用 filename 写入，并保证文件名不冲突。
- overwrite 如果写入目录存在数据，则先删除，后写入
- nonConflict，如果目录下有 fileName 前缀的文件，直接报错。

fieldDelimiter

hdfswriter 写入时的字段分隔符，需要用户保证与创建的 Hive 表的字段分隔符一致，否则无法在 Hive 表中查到数据，如果写入的文件格式为 orc，parquet ，rcfile 等二进制格式，则该参数并不起作用

encoding

写文件的编码配置，默认为 `utf-8` **慎重修改**

compress

描述：hdfs 文件压缩类型，默认不填写意味着没有压缩。其中：text 类型文件支持压缩类型有 gzip、bzip2;orc 类型文件支持的压缩类型有 NONE、SNAPPY（需要用户安装 SnappyCodec）

hadoopConfig

`hadoopConfig` 里可以配置与 Hadoop 相关的一些高级参数，比如 HA 的配置

```
"hadoopConfig":{
"dfs.nameservices": "testDfs",
"dfs.ha.namenodes.testDfs": "nn01,nn02",
"dfs.namenode.rpc-address.testDfs.namenode1": "192.168.1.1",
"dfs.namenode.rpc-address.testDfs.namenode2": "192.168.1.2",
"dfs.client.failover.proxy.provider.testDfs": "org.apache.hadoop.hdfs.server.namenode.
↪ha.ConfiguredFailoverProxyProvider"
}
```

haveKerberos

是否有 Kerberos 认证，默认 `false`，如果用户配置 true，则配置项 `kerberosKeytabFilePath`，`kerberosPrincipal` 为必填。

kerberosKeytabFilePath

Kerberos 认证 keytab 文件路径，绝对路径

kerberosPrincipal

描述：Kerberos 认证 Principal 名，如 `xxxx/hadoopclient@xxx.xxx`

3.3 类型转换

目前 HdfsWriter 支持大部分 Hive 类型，请注意检查你的类型。

下面列出 HdfsWriter 针对 Hive 数据类型转换列表:

3.11 InfluxDBWriter

InfluxDBWriter 插件实现了将数据写入 InfluxDB 读取数据的功能。底层实现上，是通过调用 InfluQL 语言接口，构建插入语句，然后进行数据插入。

3.11.1 示例

以下示例用来演示该插件从内存读取数据并写入到指定表

创建需要的库

通过以下命令来创建需要写入的库

```
# create database
influx --execute "CREATE DATABASE addax"
```

创建 job 文件

创建 `job/stream2kudu.json` 文件，内容如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
```

(下页继续)

(续上页)

```
                {
                  "random":"2001-01-01 00:00:00, 2016-07-07 23:59:59",
                  "type":"date"
                },
                {
                  "random": "1,1000",
                  "type": "long"
                },
                {
                  "random": "1,10",
                  "type": "string"
                },
                {
                  "random": "1000,50000",
                  "type": "double"
                }
              ],
              "sliceRecordCount": 10
            }
          },
          "writer": {
            "name": "influxdbwriter",
            "parameter": {
              "connection": [
                {
                  "endpoint": "http://localhost:8086",
                  "database": "addax",
                  "table": "addax_tbl"
                }
              ],
              "connTimeout": 15,
              "readTimeout": 20,
              "writeTimeout": 20,
              "username": "influx",
              "password": "influx123",
              "column": [
                {"name":"time", "type":"timestamp"},
                {"name":"user_id","type":"int"},
                {"name":"user_name", "type":"string"},
                {"name":"salary", "type":"double"}
              ],
              "preSql": ["delete from addax_tbl"],
              "batchSize": 1024,
              "retentionPolicy": {"name":"one_day_only", "duration": "1d", "replication
↪":1}
            }
          }
        }
      ]
    }
}
```

运行

执行下面的命令进行数据采集

```
bin/addax.sh job/stream2kudu.json
```

3.11.2 参数说明

column

InfluxDB 作为时序数据库，需要每条记录都有时间戳字段，因此这里会把 column 配置的第一个字段默认当作时间戳

retentionPolicy

设定数据库的 Retention Policy 策略，依据给定的配置，在指定数据库上创建一条 Retention Policy 信息。有关 Retention Policy 更详细的信息，可以参考官方文档

3.11.3 类型转换

当前支持 InfluxDB 的基本类型

3.11.4 限制

1. 当前插件仅支持 1.x 版本，2.0 及以上并不支持

3.12 KuduWriter

KuduWriter 插件实现了将数据写入到 kudu 的能力，当前是通过调用原生 RPC 接口来实现的。后期希望通过 impala 接口实现，从而增加更多的功能。

3.12.1 示例

以下示例演示了如何从内存读取样例数据并写入到 kudu 表中的。

表结构

我们用 trino 工具连接到 kudu 服务，然后通过下面的 SQL 语句创建表

```
CREATE TABLE kudu.default.users (
  user_id int WITH (primary_key = true),
  user_name varchar,
  salary double
) WITH (
  partition_by_hash_columns = ARRAY['user_id'],
  partition_by_hash_buckets = 2
);
```

job 配置文件

创建 job/stream2kudu.json 文件，内容如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "random": "1,1000",
                "type": "long"
              },
              {
                "random": "1,10",
                "type": "string"
              },
              {
                "random": "1000,50000",
                "type": "double"
              }
            ],
            "sliceRecordCount": 1000
          }
        },
        "writer": {
          "name": "kuduwriter",
          "parameter": {
            "masterAddress": "127.0.0.1:7051,127.0.0.1:7151,127.0.0.1:7251",
            "timeout": 60,
            "table": "users",
            "writeMode": "upsert",
            "column": [
              {"name":"user_id","type":"int8"},
              {"name":"user_name", "type":"string"},
              {"name":"salary", "type":"double"}
            ],
            "batchSize": 1024,
            "bufferSize": 2048,
            "skipFail": false,
            "encoding": "UTF-8"
          }
        }
      }
    ]
  }
}
```

运行

执行下下面的命令进行数据采集

```
bin/addax.sh job/stream2kudu.json
```

3.12.2 参数说明

3.12.3 已知限制

1. 暂时不支持 `truncate table`

3.13 MongoDBWriter 插件文档

3.13.1 1 快速介绍

MongoDBWriter 插件利用 MongoDB 的 java 客户端 MongoClient 进行 MongoDB 的写操作。最新版本的 Mongo 已经将 DB 锁的粒度从 DB 级别降低到 document 级别，配合上 MongoDB 强大的索引功能，基本可以满足数据源向 MongoDB 写入数据的需求，针对数据更新的需求，通过配置业务主键的方式也可以实现。

3.13.2 2 实现原理

MongoDBWriter 通过 addax 框架获取 Reader 生成的数据，然后将 Addax 支持的类型通过逐一判断转换成 MongoDB 支持的类型。其中一个值得指出的点就是 Addax 本身不支持数组类型，但是 MongoDB 支持数组类型，并且数组类型的索引还是蛮强大的。为了使用 MongoDB 的数组类型，则可以通过参数的特殊配置，将字符串可以转换成 MongoDB 中的数组。类型转换之后，就可以依托于 addax 框架并行的写入 MongoDB。

3.13.3 3 功能说明

3.1 配置样例

该示例将流式数据写入到 MongoDB 表中

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "unique_id",
```

(下页继续)

(续上页)

```
            "type": "string"
          },
          {
            "value": "sid",
            "type": "string"
          },
          {
            "value": "user_id",
            "type": "string"
          },
          {
            "value": "auction_id",
            "type": "string"
          },
          {
            "value": "content_type",
            "type": "string"
          },
          {
            "value": "pool_type",
            "type": "string"
          },
          {
            "value": "a1 a2 a3",
            "type": "string"
          },
          {
            "value": "c1 c2 c3",
            "type": "string"
          },
          {
            "value": "2020-09-06",
            "type": "string"
          },
          {
            "value": "tag1 tag2 tag3",
            "type": "string"
          },
          {
            "value": "property",
            "type": "string"
          },
          {
            "value": 1984,
            "type": "long"
          },
          {
            "value": 1900,
            "type": "long"
          },
          {
            "value": 75,
            "type": "long"
          }
        ],
        "sliceRecordCount": 10
```

(下页继续)

(续上页)

```
        }
      },
      "writer": {
        "name": "mongodbwriter",
        "parameter": {
          "address": [
            "127.0.0.1:32768"
          ],
          "userName": "",
          "userPassword": "",
          "dbName": "tag_per_data",
          "collectionName": "tag_data",
          "column": [
            {
              "name": "unique_id",
              "type": "string"
            },
            {
              "name": "sid",
              "type": "string"
            },
            {
              "name": "user_id",
              "type": "string"
            },
            {
              "name": "auction_id",
              "type": "string"
            },
            {
              "name": "content_type",
              "type": "string"
            },
            {
              "name": "pool_type",
              "type": "string"
            },
            {
              "name": "frontcat_id",
              "type": "Array",
              "splitter": " "
            },
            {
              "name": "categoryid",
              "type": "Array",
              "splitter": " "
            },
            {
              "name": "gmt_create",
              "type": "string"
            },
            {
              "name": "taglist",
              "type": "Array",
              "splitter": " "
            },
```

(下页继续)

(续上页)

```
                {
                  "name": "property",
                  "type": "string"
                },
                {
                  "name": "scorea",
                  "type": "int"
                },
                {
                  "name": "scoreb",
                  "type": "int"
                },
                {
                  "name": "scorec",
                  "type": "int"
                }
              ],
              "upsertInfo": {
                "isUpsert": "true",
                "upsertKey": "unique_id"
              }
            }
          }
        }
      ]
    }
}
```

3.2 参数说明

3.13.4 4 类型转换

3.14 MysqlWriter

MysqlWriter 插件实现了写入数据到 Mysql 主库的目的表的功能。在底层实现上，MysqlWriter 通过 JDBC 连接远程 Mysql 数据库，并执行相应的 insert into ... 或者 (replace into ...) 的 sql 语句将数据写入 Mysql，内部会分批次提交入库，需要数据库本身采用 innodb 引擎。

MysqlWriter 面向 ETL 开发工程师，他们使用 MysqlWriter 从数仓导入数据到 Mysql。同时 MysqlWriter 亦可以作为数据迁移工具为 DBA 等用户提供服务。

MysqlWriter 通过 Addax 框架获取 Reader 生成的协议数据，根据你配置的 `writeMode` 生成 `insert into...`(当主键/唯一性索引冲突时会写不进去冲突的行) 或者 `replace into...`(没有遇到主键/唯一性索引冲突时，与 insert into 行为一致，冲突时会用新行替换原有行所有字段) 的语句写入数据到 Mysql。出于性能考虑，采用了 `PreparedStatement + Batch`，并且设置了：`rewriteBatchedStatements=true`，将数据缓冲到线程上下文 Buffer 中，当 Buffer 累计到预定阈值时，才发起写入请求。

3.14.1 示例

假定要写入的 MySQL 表建表语句如下：

```
create table test.addax_tbl
(
col1 varchar(20) ,
col2 int(4),
col3 datetime,
col4 boolean,
col5 binary
) default charset utf8;
```

这里使用一份从内存产生到 Mysql 导入的数据。

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "value": 19880808,
                "type": "long"
              },
              {
                "value": "1988-08-08 08:08:08",
                "type": "date"
              },
              {
                "value": true,
                "type": "bool"
              },
              {
                "value": "test",
                "type": "bytes"
              }
            ],
            "sliceRecordCount": 1000
          }
        },
        "writer": {
          "name": "mysqlwriter",
          "parameter": {
            "writeMode": "insert",
            "username": "root",
```

(下页继续)

(续上页)

```
            "password": "",
            "column": [
              "*"
            ],
            "session": [
              "set session sql_mode='ANSI'"
            ],
            "preSql": [
              "delete from @table"
            ],
            "connection": [
              {
                "jdbcUrl": "jdbc:mysql://127.0.0.1:3306/test?useSSL=false",
                "table": [
                  "addax_tbl"
                ],
                "driver": "com.mysql.jdbc.Driver"
              }
            ]
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/stream2mysql.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/stream2mysql.json
```

3.14.2 参数说明

driver

当前 Addax 采用的 MySQL JDBC 驱动为 8.0 以上版本，驱动类名使用的 `com.mysql.cj.jdbc.Driver`，而不是 `com.mysql.jdbc.Driver`。如果你需要采集的 MySQL 服务低于 `5.6`，需要使用到 `Connector/J 5.1` 驱动，则可以采取下面的步骤：

替换插件内置的驱动

```
rm -f plugin/writer/mysqlwriter/lib/mysql-connector-java-*.jar
```

拷贝老的驱动到插件目录

```
cp mysql-connector-java-5.1.48.jar plugin/writer/mysqlwriter/lib/
```

指定驱动类名称

在你的 json 文件类，配置 `"driver": "com.mysql.jdbc.Driver"`

3.14.3 类型转换

目前 MysqlWriter 支持大部分 Mysql 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 MysqlWriter 针对 Mysql 类型转换列表:

bit 类型目前是未定义类型转换

3.15 OracleWriter 插件文档

3.15.1 1 快速介绍

OracleWriter 插件实现了写入数据到 Oracle 主库的目的表的功能。在底层实现上，OracleWriter 通过 JDBC 连接远程 Oracle 数据库，并执行相应的 `insert into ...` 语句将数据写入 Oracle，内部会分批次提交入库。

OracleWriter 面向 ETL 开发工程师，他们使用 OracleWriter 从数仓导入数据到 Oracle。同时 OracleWriter 亦可以作为数据迁移工具为 DBA 等用户提供服务。

3.15.2 2 实现原理

OracleWriter 通过 Addax 框架获取 Reader 生成的协议数据，根据你配置生成相应的 SQL 语句

注意:

1. 目的表所在数据库必须是主库才能写入数据；整个任务至少需具备 `insert into...` 的权限，是否需要其他权限，取决于你任务配置中在 `preSql` 和 `postSql` 中指定的语句。
2. OracleWriter 和 `MysqlWriter` 不同，不支持配置 `writeMode` 参数。

3.15.3 3 功能说明

3.1 配置样例

- 这里使用一份从内存产生到 Oracle 导入的数据。

```
{
"job": {
    "setting": {
        "speed": {
            "channel": 1,
          "bytes": -1
        }
    },
    "content": [{
        "reader": {
        "name": "streamreader",
        "parameter": {
            "column" : [
                {
                    "value": "Addax",
                    "type": "string"
                },
                {
```

(下页继续)

(续上页)

```
                    "value": 19880808,
                    "type": "long"
                },
                {
                    "value": "1988-08-08 08:08:08",
                    "type": "date"
                },
                {
                    "value": true,
                    "type": "bool"
                },
                {
                    "value": "test",
                    "type": "bytes"
                }
            ],
            "sliceRecordCount": 1000
        }
        },
        "writer": {
            "name": "oraclewriter",
            "parameter": {
                "username": "root",
                "password": "root",
                "column": [
                    "id",
                    "name"
                ],
                "preSql": [
                    "delete from test"
                ],
                "connection": [{
                    "jdbcUrl": "jdbc:oracle:thin:@[HOST_NAME]:PORT:[DATABASE_NAME]",
                    "table": ["test"]
                }]
            }
        }
    }]
}
}
```

3.2 参数说明

writeMode

默认情况下，采取 `insert into` 语法写入 Oracle 表，如果你希望采取主键存在时更新，不存在则写入的方式，也就是 Oracle 的 `merge into` 语法，可以使用 `update` 模式。假定表的主键为 `id`，则 `writeMode` 配置方法如下：

```
"writeMode": "update(id)"
```

如果是联合唯一索引，则配置方法如下：

```
"writeMode": "update(col1, col2)"
```

注：update 模式在 3.1.6 版本首次增加，之前版本并不支持。

session

描述：设置 oracle 连接时的 session 信息，格式示例如下：

```
"session":[
    "alter session set nls_date_format = 'dd.mm.yyyy hh24:mi:ss';"
    "alter session set NLS_LANG = 'AMERICAN';"
]
```

3.3 类型转换

类似*OracleReader*，目前 OracleWriter 支持大部分 Oracle 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

下面列出 OracleWriter 针对 Oracle 类型转换列表:

3.16 PostgresqlWriter

PostgresqlWriter 插件实现了写入数据到 PostgreSQL 主库目的表的功能。在底层实现上，PostgresqlWriter 通过 JDBC 连接远程 PostgreSQL 数据库，并执行相应的 insert into ... sql 语句将数据写入 PostgreSQL，内部会分批次提交入库。

PostgresqlWriter 面向 ETL 开发工程师，他们使用 PostgresqlWriter 从数仓导入数据到 PostgreSQL。同时 PostgresqlWriter 亦可以作为数据迁移工具为 DBA 等用户提供服务。

3.16.1 示例

以下配置演示从 postgresql 指定的表读取数据，并插入到具有相同表结构的另外一张表中，用来测试该插件所支持的数据类型。

表结构信息

假定建表语句以及输入插入语句如下：

```
create table if not exists addax_tbl
(
    c_bigint bigint,
    c_bit bit(3),
    c_bool boolean,
    c_byte bytea,
    c_char char(10),
    c_varchar varchar(20),
    c_date  date,
    c_double float8,
    c_int integer,
    c_json json,
    c_number decimal(8,3),
    c_real  real,
```

(下页继续)

(续上页)

```
    c_small smallint,
    c_text  text,
    c_ts timestamp,
    c_uuid uuid,
    c_xml xml,
    c_money money,
    c_inet inet,
    c_cidr cidr,
    c_macaddr macaddr
);
insert into addax_tbl values(
    999988887777,
    B'101',
    TRUE,
    '\xDEADBEEF',
    'hello',
    'hello, world',
    '2021-01-04',
    999888.9972,
    9876542,
    '{"bar": "baz", "balance": 7.77, "active": false}'::json,
    12345.123,
    123.123,
    126,
    'this is a long text ',
    '2020-01-04 12:13:14',
    'A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11'::uuid,
    '<foo>bar</foo>'::xml,
    '52093.89'::money,
    '192.168.1.1'::inet,
    '192.168.1/24'::cidr,
    '08002b:010203'::macaddr
);
```

创建需要插入的表的语句如下:

```
create table addax_tbl1 like addax_tbl;
```

任务配置

以下是配置文件

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": -1,
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "postgresqlreader",
          "parameter": {
```

(下页继续)

(续上页)

```
            "username": "pgtest",
            "password": "pgtest",
            "column": [
              "*"
            ],
            "connection": [
              {
                "table": [
                  "addax_tbl"
                ],
                "jdbcUrl": [
                  "jdbc:postgresql://localhost:5432/pgtest"
                ]
              }
            ]
          }
        },
        "writer": {
          "name": "postgresqlwriter",
          "parameter": {
            "column": [
              "*"
            ],
            "preSql": [
              "truncate table @table"
            ],
            "connection": [
              {
                "jdbcUrl": "jdbc:postgresql://127.0.0.1:5432/pgtest",
                "table": [
                  "addax_tbl1"
                ]
              }
            ],
            "username": "pgtest",
            "password": "pgtest",
            "writeMode": "insert"
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/pg2pg.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/pg2pg.json
```

3.16.2 参数说明

writeMode

默认情况下，采取 `insert into` 语法写入 postgresql 表，如果你希望采取主键存在时更新，不存在则写入的方式，可以使用 `update` 模式。假定表的主键为 `id`，则 `writeMode` 配置方法如下：

```
"writeMode": "update(id)"
```

如果是联合唯一索引，则配置方法如下：

```
"writeMode": "update(col1, col2)"
```

注：`update` 模式在 `3.1.6` 版本首次增加，之前版本并不支持。

3.16.3 类型转换

目前 PostgresqlWriter 支持大部分 PostgreSQL 类型，但也存在部分没有支持的情况，请注意检查你的类型。

下面列出 PostgresqlWriter 针对 PostgreSQL 类型转换列表:

3.16.4 已知限制

除以上列出的数据类型外，其他数据类型理论上均为转为字符串类型，但不确保准确性

3.17 RDBMS Writer

RDBMSWriter 插件支持从传统 RDBMS 读取数据。这是一个通用关系数据库读取插件，可以通过注册数据库驱动等方式支持更多关系数据库读取。

同时 RDBMS Writer 又是其他关系型数据库读取插件的的基础类。以下读取插件均依赖该插件

- Oracle Writer
- MySQL Writer
- PostgreSQL Writer
- ClickHouse Writer
- SQLServer Writer

注意，对于 Addax 已经提供了专门的数据库写入插件的，推荐使用专用插件，如果你需要写入的数据库没有专门插件，则考虑使用该通用插件。在使用之前，还需要执行以下操作才可以正常运行，否则运行会出现异常。

3.17.1 配置驱动

假定你需要写入 IBM DB2 的数据，因为没有提供专门的读取插件，所以我们可以使用该插件来实现，在使用之前，需要执行下面两个操作：

1. 下载对应的 JDBC 驱动，并拷贝到 `plugin/writer/rdbmswriter/libs` 目录
2. 修改 `plugin/writer/rdbmswriter/plugin.json` 文件，找到 `drivers` 一项，填写正确的 JDBC 驱动名，比如 DB2 的驱动名为 `com.ibm.db2.jcc.DB2Driver`，类似这样：

```
{
"name": "rdbmswriter",
"class": "com.wgzhao.addax.plugin.reader.rdbmswriter.RdbmsWriter",
"description": "",
"developer": "alibaba",
"drivers": ["com.ibm.db2.jcc.DB2Driver"]
}
```

以下列出常见的数据库以及对应的驱动名称

- Apache Impala: `com.cloudera.impala.jdbc41.Driver`
- Enterprise DB: `com.edb.Driver`
- PrestoDB: `com.facebook.presto.jdbc.PrestoDriver`
- IBM DB2: `com.ibm.db2.jcc.DB2Driver`
- MySQL: `com.mysql.cj.jdbc.Driver`
- Sybase Server: `com.sybase.jdbc3.jdbc.SybDriver`
- TDengine: `com.taosdata.jdbc.TSDBDriver`
- 达梦数据库: `dm.jdbc.driver.DmDriver`
- 星环 Inceptor: `io.transwarp.jdbc.InceptorDriver`
- TrinoDB: `io.trino.jdbc.TrinoDriver`
- PrestoSQL: `io.prestosql.jdbc.PrestoDriver`
- Oracle DB: `oracle.jdbc.OracleDriver`
- PostgreSQL: `org.postgresql.Drive`

3.17.2 配置说明

配置一个写入 RDBMS 的作业。

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
```

(下页继续)

(续上页)

```
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "value": 19880808,
                "type": "long"
              },
              {
                "value": "1988-08-08 08:08:08",
                "type": "date"
              },
              {
                "value": true,
                "type": "bool"
              },
              {
                "value": "test",
                "type": "bytes"
              }
            ],
            "sliceRecordCount": 1000
          }
        },
        "writer": {
          "name": "rdbmswriter",
          "parameter": {
            "connection": [
              {
                "jdbcUrl": "jdbc:dm://ip:port/database",
                "driver": "",
                "table": [
                  "table"
                ]
              }
            ],
            "username": "username",
            "password": "password",
            "column": [
              "*"
            ],
            "preSql": [
              "delete from XXX;"
            ]
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

column

所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。用户使用 * 代表默认使用所有列配置，例如 ["*"]。

支持列裁剪，即列可以挑选部分列进行导出。

支持列换序，即列可以不按照表 schema 信息进行导出。

支持常量配置，用户需要按照 JSON 格式:

```
["id", "`table`", "1", "'bazhen.csy'", "null", "to_char(a + 1)", "2.3" ,
"true"]
```

- id 为普通列名
- `table` 为包含保留在的列名，
- 1 为整形数字常量，
- 'bazhen.csy' 为字符串常量
- null 为空指针，注意，这里的 null 必须以字符串形式出现，即用双引号引用
- to_char(a + 1) 为表达式，
- 2.3 为浮点数，
- true 为布尔值，同样的，这里的布尔值也必须用双引号引用

Column 必须显示填写，不允许为空!

jdbcUrl

jdbcUrl 配置除了配置必要的信息外，我们还可以在增加每种特定驱动的特定配置属性，这里特别提到我们可以利用配置属性对代理的支持从而实现通过代理访问数据库的功能。比如对于 PrestoSQL 数据库的 JDBC 驱动而言，支持 socksProxy 参数，比如一个可能的 jdbcUrl 为

```
jdbc:presto://127.0.0.1:8080/hive?socksProxy=192.168.1.101:1081
```

大部分关系型数据库的 JDBC 驱动支持 socksProxyHost,socksProxyPort 参数来支持代理访问。也有一些特别的情况。

以下是各类数据库 JDBC 驱动所支持的代理类型以及配置方式

driver

大部分情况下，一个数据库的 JDBC 驱动是固定的，但有些因为版本的不同，所建议的驱动类名不同，比如 MySQL。新的 MySQL JDBC 驱动类型推荐使用 com.mysql.cj.jdbc.Driver 而不是以前的 com.mysql.jdbc.Drver。如果想要使用就的驱动名称，则可以配置 driver 配置项。

3.18 RedisWriter 插件文档

3.18.1 1 快速介绍

RedisWrite 提供了还原 Redis dump 命令的能力，并写入到目标 Redis。支持 redis cluster 集群、proxy、以及单机

3.18.2 2 功能与限制

1. 支持写入 redis cluster 集群、proxy、以及单机。

我们暂时不能做到：

1. 只支持写入 Redis 数据源。

3.18.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "redisreader",
          "parameter": {
            "connection": [
              {
                "uri": "file:///root/dump.rdb",
                "uri": "http://localhost/dump.rdb",
                "uri": "tcp://127.0.0.1:7001",
                "uri": "tcp://127.0.0.1:7002",
                "uri": "tcp://127.0.0.1:7003"
              }
            ]
          }
        },
        "writer": {
          "name": "rediswriter",
          "parameter": {
            "connection": [
              {
                "uri": "tcp://127.0.0.1:6379",
                "auth": "123456"
              }
            ],
            "redisCluster": false,
            "flushDB": false
          }
        }
      }
    ],
    "setting": {
      "speed": {
```

(下页继续)

(续上页)

```
        "channel": 1,
        "bytes": -1
      }
    }
  }
}
```

3.2 参数说明

3.19 SqlServerWriter 插件文档

3.19.1 1 快速介绍

SqlServerWriter 插件实现了写入数据到 SqlServer 库的目的表的功能。在底层实现上，SqlServerWriter 通过 JDBC 连接远程 SqlServer 数据库，并执行相应的 insert into ... sql 语句将数据写入 SqlServer，内部会分批次提交入库。

SqlServerWriter 面向 ETL 开发工程师，他们使用 SqlServerWriter 从数仓导入数据到 SqlServer。同时 SqlServerWriter 亦可以作为数据迁移工具为 DBA 等用户提供服务。

3.19.2 2 实现原理

SqlServerWriter 通过 Addax 框架获取 Reader 生成的协议数据，根据你配置生成相应的 SQL 语句 `insert into...`(当主键/唯一性索引冲突时会写不进去冲突的行)

注意：

1. 目的表所在数据库必须是主库才能写入数据；整个任务至少需具备 `insert into...` 的权限，是否需要其他权限，取决于你任务配置中在 `preSql` 和 `postSql` 中指定的语句。
2. SqlServerWriter 和 MysqlWriter 不同，不支持配置 writeMode 参数。

3.19.3 3 功能说明

3.1 配置样例

这里使用一份从内存产生到 SqlServer 导入的数据。

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {},
        "writer": {
          "name": "sqlserverwriter",
```

(下页继续)

(续上页)

```
            "parameter": {
              "username": "root",
              "password": "root",
              "column": [
                "db_id",
                "db_type",
                "db_ip",
                "db_port",
                "db_role",
                "db_name",
                "db_username",
                "db_password",
                "db_modify_time",
                "db_modify_user",
                "db_description",
                "db_tddl_info"
              ],
              "connection": [
                {
                  "table": [
                    "db_info_for_writer"
                  ],
                  "jdbcUrl": "jdbc:sqlserver://[HOST_NAME]:PORT;DatabaseName=[DATABASE_
↪NAME]"
                }
              ],
              "preSql": [
                "delete from @table where db_id = -1;"
              ],
              "postSql": [
                "update @table set db_modify_time = now() where db_id = 1;"
              ]
            }
          }
        }
      ]
    }
}
```

3.2 参数说明

3.3 类型转换

类似 SqlServerReader，目前 SqlServerWriter 支持大部分 SqlServer 类型，但也存在部分个别类型没有支持的情况，请注意检查你的类型。

3.20 StreamWriter

StreamWriter 是一个将数据写入内存的插件，一般用来将获取到的数据写到终端，用来调试读取插件的数据处理情况。

一个典型的 streamwriter 配置如下：

```
{
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "print": true
  }
}
```

上述配置会将获取的数据直接打印到终端。该插件也支持将数据写入到文件，配置如下：

```
{
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "path": "/tmp/out",
    "fileName": "out.txt",
    "fieldDelimiter": ",",
    "recordNumBeforeSleep": "100",
    "sleepTime": "5"
  }
}
```

上述配置中:

`fieldDelimiter` 表示字段分隔符，默认为制表符 (`\t`) `recordNumBeforeSleep` 表示获取多少条记录后，执行休眠，默认为 0，表示不启用该功能

`sleepTime` 则表示休眠多长时间，单位为秒，默认为 0，表示不启用该功能。

上述配置的含义是将数据写入到 `/tmp/out/out.txt` 文件，每获取 100 条记录后，休眠 5 秒。

3.21 TDengineWriter

TDengineWriter 插件实现了将数据写入到涛思公司的 TDengine 数据库系统。在底层实现上，TDengineWriter 通过 JDBC JNI 驱动连接远程 TDengine 数据库，并执行相应的 sql 语句将数据批量写入 TDengine 库中。

3.21.1 前置条件

考虑到性能问题，该插件使用了 TDengine 的 JDBC-JNI 驱动，该驱动直接调用客户端 API（`libtaos.so` 或 `taos.dll`）将写入和查询请求发送到 `taosd` 实例。因此在使用之前需要配置好动态库链接文件。

首先将 `plugin/writer/tdenginewriter/libs/libtaos.so.2.0.16.0` 拷贝到 `/usr/lib64` 目录，然后执行下面的命令创建软链接

```
ln -sf /usr/lib64/libtaos.so.2.0.16.0 /usr/lib64/libtaos.so.1
ln -sf /usr/lib64/libtaos.so.1 /usr/lib64/libtaos.so
```

3.21.2 示例

假定要写入的表如下：

```
create table test.addax_test (
    ts timestamp,
    name nchar(100),
    file_size int,
    file_date timestamp,
    flag_open bool,
    memo nchar(100)
);
```

以下是配置文件

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column" : [
              {
                "random":"2017-08-01 00:01:02,2020-01-01 12:13:14",
                "type": "date"
              },
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "value": 19880808,
                "type": "long"
              },
              {
                "value": "1988-08-08 08:08:08",
                "type": "date"
              },
              {
                "value": true,
                "type": "bool"
              },
              {
                "value": "test",
                "type": "bytes"
              }
            ],
            "sliceRecordCount": 1000
          }
        },
        "writer": {
```

(下页继续)

(续上页)

```
          "name": "tdenginewriter",
          "parameter": {
            "username": "root",
            "password": "taosdata",
            "column": ["ts", "name", "file_size", "file_date", "flag_open", "memo" ],
            "connection": [
              {
                "jdbcUrl": "jdbc:TAOS://127.0.0.1:6030/test",
                "table": [ "addax_test"]
              }
            ]
          }
        }
      }
    ]
  }
}
```

将上述配置文件保存为 `job/stream2tdengine.json`

执行采集命令

执行以下命令进行数据采集

```
bin/addax.sh job/tdengine2stream.json
```

命令输出类似如下：

```
2021-02-20 15:52:07.691 [main] INFO  VMInfo - VMInfo# operatingSystem class => sun.
↪management.OperatingSystemImpl
2021-02-20 15:52:07.748 [main] INFO  Engine -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "column":[
                                                {
                                                        "random":"2017-08-01 00:01:02,
↪2020-01-01 12:13:14",
                                                        "type":"date"
                                                },
                                                {
                                                        "type":"string",
                                                        "value":"Addax"
                                                },
                                                {
                                                        "type":"long",
                                                        "value":19880808
                                                },
                                                {
                                                        "type":"date",
                                                        "value":"1988-08-08 08:08:08"
                                                },
                                                {
```

(下页继续)

(续上页)

```
                                                        "type":"bool",
                                                        "value":true
                                                },
                                                {
                                                        "type":"bytes",
                                                        "value":"test"
                                                }
                                        ],
                                        "sliceRecordCount":1000
                                },
                                "name":"streamreader"
                        },
                        "writer":{
                                "parameter":{
                                        "password":"*****",
                                        "column":[
                                                "ts",
                                                "name",
                                                "file_size",
                                                "file_date",
                                                "flag_open",
                                                "memo"
                                        ],
                                        "connection":[
                                                {
                                                        "jdbcUrl":"jdbc:TAOS://127.0.
↪0.1:6030/test",
                                                        "table":[
                                                                "addax_test"
                                                        ]
                                                }
                                        ],
                                        "username":"root",
                                        "preSql":[]
                                },
                                "name":"tdenginewriter"
                        }
                }
        ],
        "setting":{
                "speed":{
                        "bytes":-1,
                        "channel":1
                }
        }
}

2021-02-20 15:52:07.786 [main] INFO  PerfTrace - PerfTrace traceId=job_-1,␣
↪isEnable=false, priority=0
2021-02-20 15:52:07.787 [main] INFO  JobContainer - Addax jobContainer starts job.
2021-02-20 15:52:07.789 [main] INFO  JobContainer - Set jobId = 0
java.library.path:/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
2021-02-20 15:52:08.048 [job-0] INFO  OriginalConfPretreatmentUtil - table:[addax_
↪test] all columns:[ts,name,file_size,file_date,flag_open,memo].
2021-02-20 15:52:08.056 [job-0] INFO  OriginalConfPretreatmentUtil - Write data [
INSERT INTO %s (ts,name,file_size,file_date,flag_open,memo) VALUES(?,?,?,?,?,?)
```

(下页继续)

(续上页)

```
], which jdbcUrl like:[jdbc:TAOS://127.0.0.1:6030/test]

2021-02-20 15:52:11.158 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-02-20 15:52:07
任务结束时刻                    : 2021-02-20 15:52:11
任务总计耗时                    :                  3s
任务平均流量                    :           11.07KB/s
记录写入速度                    :            333rec/s
读出记录总数                    :                1000
读写失败总数                    :                   0
```

3.21.3 参数说明

使用 JDBC-RESTful 接口

如果不想依赖本地库，或者没有权限，则可以使用 `JDBC-RESTful` 接口来写入表，相比 JDBC-JNI 而言，配置区别是：

- driverClass 指定为 `com.taosdata.jdbc.rs.RestfulDriver`
- jdbcUrl 以 `jdbc:TAOS-RS://` 开头；
- 使用 `6041` 作为连接端口

所以上述配置中的 `connection` 应该修改为如下：

```
"connection": [{
  "jdbcUrl":"jdbc:TAOS-RS://127.0.0.1:6041/test",
  "table":["addax_test"],
  "driver":"com.taosdata.jdbc.rs.RestfulDriver"
}]
```

3.21.4 类型转换

目前 TDenginereader 支持 TDengine 所有类型，具体如下

3.21.5 当前支持版本

TDengine 2.0.16

3.21.6 注意事项

- TDengine JDBC-JNI 驱动和动态库版本要求一一匹配，因此如果你的数据版本并不是 `2.0.16`，则需要同时替换动态库和插件目录中的 JDBC 驱动
- TDengine 的时序字段（timestamp）默认最小值为 `1500000000000`，即 `2017-07-14 10:40:00.0`，如果你写入的时许时间戳小于该值，则会报错

3.22 TxtFileWriter 插件文档

3.22.1 1 快速介绍

TxtFileWriter 提供了向本地文件写入类 CSV 格式的一个或者多个表文件。TxtFileWriter 服务的用户主要在于 Addax 开发、测试同学。

写入本地文件内容存放的是一张逻辑意义上的二维表，例如 CSV 格式的文本信息。

3.22.2 2 功能与限制

TxtFileWriter 实现了从 Addax 协议转为本地 TXT 文件功能，本地文件本身是无结构化数据存储，TxtFileWriter 如下几个方面约定:

1. 支持且仅支持写入 TXT 的文件，且要求 TXT 中 shema 为一张二维表。
2. 支持类 CSV 格式文件，自定义分隔符。
3. 支持文本压缩，现有压缩格式为 gzip、bzip2。
4. 支持多线程写入，每个线程写入不同子文件。
5. 文件支持滚动，当文件大于某个 size 值或者行数值，文件需要切换。[暂不支持]

我们不能做到:

1. 单个文件不能支持并发写入。

3.22.3 3 功能说明

3.1 配置样例

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 2,
        "bytes": -1
      }
    },
    "content": [
      {
        "reader": {
          "name": "txtfilereader",
          "parameter": {
            "path": [
              "/tmp/data"
            ],
            "encoding": "UTF-8",
            "column": [
              {
                "index": 0,
                "type": "long"
              },
              {
```

(下页继续)

(续上页)

```
                "index": 1,
                "type": "boolean"
              },
              {
                "index": 2,
                "type": "double"
              },
              {
                "index": 3,
                "type": "string"
              },
              {
                "index": 4,
                "type": "date",
                "format": "yyyy.MM.dd"
              }
            ],
            "fieldDelimiter": ","
          }
        },
        "writer": {
          "name": "txtfilewriter",
          "parameter": {
            "path": "/tmp/result",
            "fileName": "luohw",
            "writeMode": "truncate",
            "dateFormat": "yyyy-MM-dd"
          }
        }
      }
    ]
  }
}
```

3.2 参数说明

writeMode

写入前数据清理处理模式：

- truncate，写入前清理目录下一 fileName 前缀的所有文件。
- append，写入前不做任何处理，直接使用 filename 写入，并保证文件名不冲突。
- nonConflict，如果目录下有 fileName 前缀的文件，直接报错。

fileFormat

文件写出的格式，包括 csv 和 text 两种，csv 是严格的 csv 格式，如果待写数据包括列分隔符，则会按照 csv 的转义语法转义，转义符号为双引号 "；text 格式是用列分隔符简单分割待写数据，对于待写数据包括列分隔符情况下不做转义。

3.3 类型转换

本地文件本身不提供数据类型，该类型是 Addax TxtFileWriter 定义：

其中：

- Long 是指本地文件文本中使用整形的字符串表示形式，例如”19901219”。
- Double 是指本地文件文本中使用 Double 的字符串表示形式，例如”3.1415”。
- Boolean 是指本地文件文本中使用 Boolean 的字符串表示形式，例如”true”、”false”。不区分大小写。
- Date 是指本地文件文本中使用 Date 的字符串表示形式，例如”2014-12-31”，Date 可以指定 format 格式。

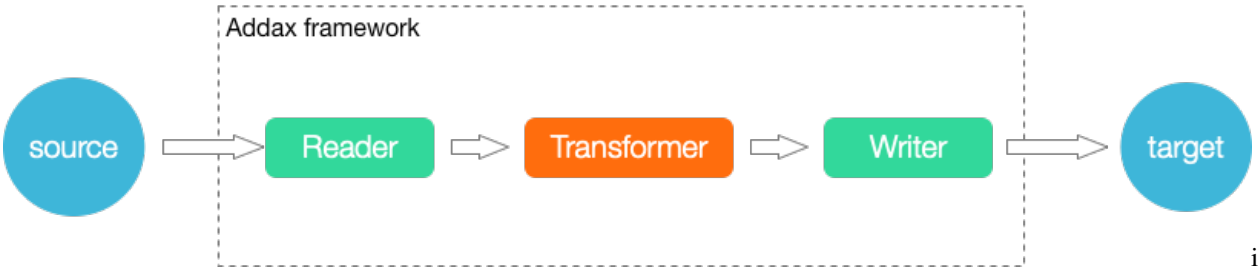
CHAPTER 4

Transformer 插件文档

4.1 Transformer 定义

在数据同步、传输过程中，存在用户对于数据传输进行特殊定制化的需求场景，包括裁剪列、转换列等工作，可以借助 ETL 的 T 过程实现 (Transformer)。Addax 包含了完成的 E(Extract)、T(Transformer)、L(Load) 支持。

4.2 运行模型



image

4.3 UDF 函数

4.3.1 dx_substr

```
dx_substr(idx, pos, length) -> str
```

参数

- `idx`: 字段编号，对应 record 中第几个字段
- `pos`: 字段值的开始位置
- `length`: 目标字段长度

返回：从字符串的指定位置（包含）截取指定长度的字符串。如果开始位置非法抛出异常。如果字段为空值，直接返回（即不参与本 transformer）

4.3.2 dx_pad

`dx_pad(idx, flag, length, chr)`

参数

- `idx`: 字段编号，对应 record 中第几个字段
- `flag`: "l","r", 指示是在头进行填充，还是尾进行填充
- `length`: 目标字段长度
- `chr`: 需要填充的字符

返回：如果源字符串长度小于目标字段长度，按照位置添加 pad 字符后返回。如果长于，直接截断（都截右边）。如果字段为空值，转换为空字符串进行 pad，即最后的字符串全是需要 pad 的字符

举例：

```
dx_pad(1,"l","4","A"), 如果 column 1 的值为 xyz=> Axyz,  值为 xyzzzzz => xyzz
dx_pad(1,"r","4","A"), 如果 column 1 的值为 xyz=> xyzA,  值为 xyzzzzz => xyzz
```

4.3.3 dx_replace

`dx_replace(idx, pos, length, str) -> str`

参数

- `idx`: 字段编号，对应 record 中第几个字段
- `pos`: 字段值的开始位置
- `length`: 需要替换的字段长度
- `str`: 要替换的字符串

返回：从字符串的指定位置（包含）替换指定长度的字符串。如果开始位置非法抛出异常。如果字段为空值，直接返回（即不参与本 transformer）

举例：

```
dx_replace(1,"2","4","****")  column 1 的 value 为“addaxTest” =>"da****est"
dx_replace(1,"5","10","****")  column 1 的 value 为“addaxTest” =>"data****"
```

4.3.4 dx_filter

`dx_filter(idx, operator, expr) -> str`

参数：

- `idx`: 字段编号，对应 record 中第几个字段
- `operator`: 运算符, 支持 `like`, `not like`, >, =, <, >=, !=, <=
- `expr`: 正则表达式（java 正则表达式）、值
- `str`: 要替换的字符串

返回：

- 如果匹配正则表达式，返回 Null，表示过滤该行。不匹配表达式时，表示保留该行。（注意是该行）。对于 >, =, < 都是对字段直接 compare 的结果.
- `like` , `not like` 是将字段转换成字符类型，然后和目标正则表达式进行全匹配。
- >, =, <, >=, !=, <= , 按照类型进行比较, 数值类型按大小比较, 字符及布尔类型按照字典序比较
- 如果目标字段为空（null），对于 `= null` 的过滤条件，将满足条件，被过滤。`! =null` 的过滤条件，null 不满足过滤条件，不被过滤。`like`，字段为 null 不满足条件，不被过滤，和 `not like`，字段为 null 满足条件，被过滤。

举例

```
dx_filter(1,"like","dataTest")
dx_filter(1,">=","10")
```

关联 filter 暂不支持，即多个字段的联合判断，函参太过复杂，用户难以使用。

4.3.5 dx_groovy

`dx_groovy(code, package) -> record`

参数

- `coee`: 符合 groovy 编码要求的代码
- `package`: extraPackage, 列表或者为空

返回

Record 数据类型

注意：

- `dx_groovy` 只能调用一次。不能多次调用。
- `groovy code` 中支持 `java.lang`, `java.util` 的包，可直接引用的对象有 `record`，以及 element 下的各种 column（BoolColumn.class,BytesColumn.class,DateColumn.class,DoubleColumn.class,LongColumn.class,StringColumn.cla 不支持其他包，如果用户有需要用到其他包，可设置 extraPackage，注意 extraPackage 不支持第三方 jar 包。
- `groovy code` 中，返回更新过的 `Record`（比如 record.setColumn(columnIndex, new StringColumn(newValue));），或者 null。返回 null 表示过滤此行。
- 用户可以直接调用静态的 Util 方式（GroovyTransformerStaticUtil)

举例:

groovy 实现的 subStr

```
String code="Column column = record.getColumn(1);\n"+
        " String oriValue = column.asString();\n"+
        " String newValue = oriValue.substring(0, 3);\n"+
        " record.setColumn(1, new StringColumn(newValue));\n"+
        " return record;";
        dx_groovy(record);
```

groovy 实现的 Replace

```
String code2="Column column = record.getColumn(1);\n"+
        " String oriValue = column.asString();\n"+
        " String newValue = \"****\" + oriValue.substring(3, oriValue.length());\n"+
        " record.setColumn(1, new StringColumn(newValue));\n"+
        " return record;";
```

groovy 实现的 Pad

```
String code3="Column column = record.getColumn(1);\n"+
        " String oriValue = column.asString();\n"+
        " String padString = \"12345\";\n"+
        " String finalPad = \"\";\n"+
        " int NeedLength = 8 - oriValue.length();\n"+
        "         while (NeedLength > 0) {\n"+
        "\n"+
        "             if (NeedLength >= padString.length()) {\n"+
        "                 finalPad += padString;\n"+
        "                 NeedLength -= padString.length();\n"+
        "             } else {\n"+
        "                 finalPad += padString.substring(0, NeedLength);\n"+
        "                 NeedLength = 0;\n"+
        "             }\n"+
        "         }\n"+
        " String newValue= finalPad + oriValue;\n"+
        " record.setColumn(1, new StringColumn(newValue));\n"+
        " return record;";
```

4.4 Job 定义

本例中，配置 4 个 UDF。

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "My name is xxxx",
                "type": "string"
              },
              {
                "value": "password is Passw0rd",
                "type": "string"
              },
              {
                "value": 19890604,
                "type": "long"
```

(下页继续)

(续上页)

```
            },
            {
              "value": "1989-06-04 00:00:00",
              "type": "date"
            },
            {
              "value": true,
              "type": "bool"
            },
            {
              "value": "test",
              "type": "bytes"
            },
            {
              "random": "0,10",
              "type": "long"
            }
          ],
          "sliceRecordCount": 10
        }
      },
      "writer": {
        "name": "streamwriter",
        "parameter": {
          "print": true,
          "encoding": "UTF-8"
        }
      },
      "transformer": [
        {
          "name": "dx_replace",
          "parameter": {
            "columnIndex": 0,
            "paras": [
              "11",
              "6",
              "wgzhao"
            ]
          }
        },
        {
          "name": "dx_substr",
          "parameter": {
            "columnIndex": 1,
            "paras": [
              "0",
              "12"
            ]
          }
        },
        {
          "name": "dx_map",
          "parameter": {
            "columnIndex": 2,
            "paras": [
              "^",
```

(下页继续)

(续上页)

```
                "2"
              ]
            }
          },
          {
            "name": "dx_filter",
            "parameter": {
              "columnIndex": 6,
              "paras": [
                "<",
                "5"
              ]
            }
          }
        ]
      }
    ]
  }
}
```

4.5 自定义函数

如果自带的函数不满足数据转换要求，我们可以在 transformer 编写满足 groovy 规范要求的代码，下面给出一个完整的例子

```
{
  "job": {
    "setting": {
      "speed": {
        "byte": -1,
        "channel": 1
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "column": [
              {
                "value": "Addax",
                "type": "string"
              },
              {
                "incr": "1",
                "type": "long"
              },
              {
                "incr": "1989/06/04 00:00:01,-1",
                "type": "date",
```

(下页继续)

(续上页)

```
                "dateFormat": "yyyy/MM/dd hh:mm:ss"
              },
              {
                "value": true,
                "type": "bool"
              },
              {
                "value": "test",
                "type": "bytes"
              }
            ],
            "sliceRecordCount": 10
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "print": true,
            "column": [
              "col1"
            ],
            "encoding": "UTF-8"
          }
        },
        "transformer": [
          {
            "name":"dx_groovy",
            "description":"Add string 'Header_' to the first column value;Double the␣
↪value of the second field",
            "parameter": {
              "code": "record.setColumn(0, new StringColumn('Header_' + record.
↪getColumn(0).asString()));record.setColumn(1, new LongColumn(record.getColumn(1).
↪asLong() * 2));return record;"
            }
          }
        ]
      }
    ]
  }
}
```

上述 `transformer` 代码针对每条记录的前面两个字段做了修改，对第一个字段的字符串，在字符串前面增加 `Header_` 字符；第二个整数字段值进行倍增处理。最后执行的结果如下：

```
$ bin/addax.sh job/transformer_demo.json


  ___      _     _
 / _ \    | |   | |
/ /_\ \ __| | __| | __ ___  __
|  _  |/ _` |/ _` |/ _` \ \/ /
| | | | (_| | (_| | (_| |>  <
\_| |_/\__,_|\__,_|\__,_/_/\_\

:: Addax version ::    (v4.0.2-SNAPSHOT)

2021-08-04 15:45:56.421 [        main] INFO  VMInfo               - VMInfo#␣
↪operatingSystem class => com.sun.management.internal.OperatingSystemImpl
```

(下页继续)

(续上页)

```
2021-08-04 15:45:56.443 [        main] INFO  Engine               -
{
        "content":[
                {
                        "reader":{
                                "parameter":{
                                        "column":[
                                                {
                                                        "type":"string",
                                                        "value":"Addax"
                                                },
                                                {
                                                        "incr":"1",
                                                        "type":"long"
                                                },
                                                {
                                                        "incr":"1989/06/04 00:00:01,-1
↪",
                                                        "dateFormat":"yyyy/MM/dd␣
↪hh:mm:ss",
                                                        "type":"date"
                                                },
                                                {
                                                        "type":"bool",
                                                        "value":true
                                                },
                                                {
                                                        "type":"bytes",
                                                        "value":"test"
                                                }
                                        ],
                                        "sliceRecordCount":10
                                },
                                "name":"streamreader"
                        },
                        "transformer":[
                                {
                                        "parameter":{
                                                "code":"record.setColumn(0, new␣
↪StringColumn('Header_' + record.getColumn(0).asString()));record.setColumn(1, new␣
↪LongColumn(record.getColumn(1).asLong() * 2));return record;"
                                        },
                                        "name":"dx_groovy",
                                        "description":"Add string 'Header_' to the␣
↪first column value;Double the value of the second field"
                                }
                        ],
                        "writer":{
                                "parameter":{
                                        "print":true,
                                        "column":[
                                                "col1"
                                        ],
                                        "encoding":"UTF-8"
                                },
                                "name":"streamwriter"
```

(下页继续)

(续上页)

```
                        }
                }
        ],
        "setting":{
                "errorLimit":{
                        "record":0,
                        "percentage":0.02
                },
                "speed":{
                        "byte":-1,
                        "channel":1
                }
        }
}

2021-08-04 15:45:56.458 [        main] INFO  PerfTrace            - PerfTrace␣
↪traceId=job_-1, isEnable=false, priority=0
2021-08-04 15:45:56.459 [        main] INFO  JobContainer         - Addax␣
↪jobContainer starts job.
2021-08-04 15:45:56.460 [        main] INFO  JobContainer         - Set jobId = 0
2021-08-04 15:45:56.470 [       job-0] INFO  JobContainer         - Addax Reader.Job␣
↪[streamreader] do prepare work .
2021-08-04 15:45:56.471 [       job-0] INFO  JobContainer         - Addax Writer.Job␣
↪[streamwriter] do prepare work .
2021-08-04 15:45:56.471 [       job-0] INFO  JobContainer         - Job set Channel-
↪Number to 1 channels.
2021-08-04 15:45:56.472 [       job-0] INFO  JobContainer         - Addax Reader.Job␣
↪[streamreader] splits to [1] tasks.
2021-08-04 15:45:56.472 [       job-0] INFO  JobContainer         - Addax Writer.Job␣
↪[streamwriter] splits to [1] tasks.
2021-08-04 15:45:56.498 [       job-0] INFO  JobContainer         - Scheduler starts␣
↪[1] taskGroups.
2021-08-04 15:45:56.505 [ taskGroup-0] INFO  TaskGroupContainer   - taskGroupId=[0]␣
↪start [1] channels for [1] tasks.
2021-08-04 15:45:56.517 [ taskGroup-0] INFO  Channel              - Channel set byte_
↪speed_limit to -1, No bps activated.
2021-08-04 15:45:56.517 [ taskGroup-0] INFO  Channel              - Channel set␣
↪record_speed_limit to -1, No tps activated.
2021-08-04 15:45:56.520 [ taskGroup-0] INFO  TransformerUtil      -  user config␣
↪transformers [[dx_groovy]], loading...
2021-08-04 15:45:56.531 [ taskGroup-0] INFO  TransformerUtil      -  1 of transformer␣
↪init success. name=dx_groovy, isNative=true parameter = {"code":"record.setColumn(0,
↪ new StringColumn('Header_' + record.getColumn(0).asString()));record.setColumn(1,␣
↪new LongColumn(record.getColumn(1).asLong() * 2));return record;"}

Header_Addax    2       1989-06-04 00:00:01     true    test
Header_Addax    4       1989-06-03 00:00:01     true    test
Header_Addax    6       1989-06-02 00:00:01     true    test
Header_Addax    8       1989-06-01 00:00:01     true    test
Header_Addax    10      1989-05-31 00:00:01     true    test
Header_Addax    12      1989-05-30 00:00:01     true    test
Header_Addax    14      1989-05-29 00:00:01     true    test
Header_Addax    16      1989-05-28 00:00:01     true    test
Header_Addax    18      1989-05-27 00:00:01     true    test
Header_Addax    20      1989-05-26 00:00:01     true    test
```

(下页继续)

(续上页)

```
2021-08-04 15:45:59.515 [       job-0] INFO  AbstractScheduler    - Scheduler␣
↪accomplished all tasks.
2021-08-04 15:45:59.517 [       job-0] INFO  JobContainer         - Addax Writer.Job␣
↪[streamwriter] do post work.
2021-08-04 15:45:59.518 [       job-0] INFO  JobContainer         - Addax Reader.Job␣
↪[streamreader] do post work.
2021-08-04 15:45:59.521 [       job-0] INFO  JobContainer         - PerfTrace not␣
↪enable!
2021-08-04 15:45:59.524 [       job-0] INFO  StandAloneJobContainerCommunicator -␣
↪Total 10 records, 330 bytes | Speed 110B/s, 3 records/s | Error 0 records, 0 bytes␣
↪|  All Task WaitWriterTime 0.000s |  All Task WaitReaderTime 0.000s | Transformer␣
↪Success 10 records | Transformer Error 0 records | Transformer Filter 0 records |␣
↪Transformer usedTime 0.383s | Percentage 100.00%
2021-08-04 15:45:59.527 [       job-0] INFO  JobContainer         -
任务启动时刻                    : 2021-08-04 15:45:56
任务结束时刻                    : 2021-08-04 15:45:59
任务总计耗时                    :                  3s
任务平均流量                    :              110B/s
记录写入速度                    :              3rec/s
读出记录总数                    :                  10
读写失败总数                    :                   0

2021-08-04 15:45:59.528 [       job-0] INFO  JobContainer         -
Transformer 成功记录总数          :                  10
Transformer 失败记录总数          :                   0
Transformer 过滤记录总数          :                   0
```

4.6 计量和脏数据

Transform 过程涉及到数据的转换，可能造成数据的增加或减少，因此更加需要精确度量，包括：

- Transform 的入参 Record 条数、字节数。
- Transform 的出参 Record 条数、字节数。
- Transform 的脏数据 Record 条数、字节数。
- 如果是多个 Transform，某一个发生脏数据，将不会再进行后面的 transform，直接统计为脏数据。
- 目前只提供了所有 Transform 的计量（成功，失败，过滤的 count，以及 transform 的消耗时间）。

涉及到运行过程的计量数据展现定义如下：

```
Total 1000000 records, 22000000 bytes | Transform 100000 records(in), 10000␣
↪records(out) | Speed 2.10MB/s, 100000 records/s | Error 0 records, 0 bytes |␣
↪Percentage 100.00%
```

注意，这里主要记录转换的输入输出，需要检测数据输入输出的记录数量变化。

涉及到最终作业的计量数据展现定义如下：

```
任务启动时刻                    : 2015-03-10 17:34:21
任务结束时刻                    : 2015-03-10 17:34:31
任务总计耗时                    :                 10s
任务平均流量                    :            2.10MB/s
记录写入速度                    :          100000rec/s
```

(下页继续)

(续上页)

```
转换输入总数                    :            1000000
转换输出总数                    :            1000000
读出记录总数                    :            1000000
同步失败总数                    :                  0
```

注意，这里主要记录转换的输入输出，需要检测数据输入输出的记录数量变化。

CHAPTER 5

任务结果上报服务器功能说明

5.1 快速介绍

主要用于将定时任务的结果上报给指定服务器

5.2 功能与限制

1. 支付 http 协议，JSON 格式。
2. 接口地址配置在 `core.json` 文件下的 `core.addaxServer.address` 下。
3. 异步发送。
4. 需要引入 `httpclient-4.5.2.jar`，`httpcore-4.4.5.jar`，`httpcore-nio-4.4.5.jar`，`httpasyncclient-4.1.2.jar` 相关 jar 包

5.3 功能说明

5.3.1 配置样例

```
{
  "jobName": "test",
  "startTimeStamp": 1587971621,
  "endTimeStamp": 1587971621,
  "totalCosts": 10,
  "byteSpeedPerSecond": 33,
  "recordSpeedPerSecond": 1,
  "totalReadRecords": 6,
  "totalErrorRecords": 0
}
```

5.3.2 参数说明

jobName 的设置规则如下:

1. 在命令行通过传递 -P-DjobName=xxxx 方式指定, 否则
2. 配置文件的 writer.parameters.path 值按 / 分割后取第 2，3 列用点 (.) 拼接而成，其含义是为库名及表名, 否则
3. 否则设置为 jobName

CHAPTER 6

Addax 插件开发宝典

本文面向 Adda x 插件开发人员，尝试尽可能全面地阐述开发一个 Addax 插件所经过的历程，力求消除开发者的困惑，让插件开发变得简单。

6.1 Addax 为什么要使用插件机制

从设计之初，Addax 就把异构数据源同步作为自身的使命，为了应对不同数据源的差异、同时提供一致地同步原语和扩展能力，Addax 自然而然地采用了 框架 + 插件的模式：

- 插件只需关心数据的读取或者写入本身。
- 而同步的共性问题，比如：类型转换、性能、统计，则交由框架来处理。

作为插件开发人员，则需要关注两个问题：

1. 数据源本身的读写数据正确性。
2. 如何与框架沟通、合理正确地使用框架。

6.2 插件视角看框架

6.2.1 逻辑执行模型

插件开发者不用关心太多，基本只需要关注特定系统读和写，以及自己的代码在逻辑上是怎样被执行的，哪一个方法是在什么时候被调用的。在此之前，需要明确以下概念：

- Job: Job 是 Addax 用以描述从一个源头到一个目的端的同步作业，是 Addax 数据同步的最小业务单元。比如：从一张 mysql 的表同步到 odps 的一个表的特定分区。
- Task: Task 是为最大化而把 Job 拆分得到的最小执行单元。比如：读一张有 1024 个分表的 mysql 分库分表的 Job，拆分成 1024 个读 Task，用若干个并发执行。

- TaskGroup: 描述的是一组 Task 集合。在同一个 TaskGroupContainer 执行下的 Task 集合称之为 TaskGroup
- JobContainer: Job 执行器，负责 Job 全局拆分、调度、前置语句和后置语句等工作的工作单元。类似 Yarn 中的 JobTracker
- TaskGroupContainer: TaskGroup 执行器，负责执行一组 Task 的工作单元，类似 Yarn 中的 TaskTracker。

简而言之，**Job 拆分成 Task，在分别在框架提供的容器中执行，插件只需要实现 Job 和 Task 两部分逻辑**。

6.2.2 物理执行模型

框架为插件提供物理上的执行能力（线程）。Addax 框架有三种运行模式：

- Standalone: 单进程运行，没有外部依赖。
- Local: 单进程运行，统计信息、错误信息汇报到集中存储。
- Distrubuted: 分布式多进程运行，依赖 Addax Service 服务。

当然，上述三种模式对插件的编写而言没有什么区别，你只需要避开一些小错误，插件就能够在单机/分布式之间无缝切换了。当 JobContainer 和 TaskGroupContainer 运行在同一个进程内时，就是单机模式（Standalone 和 Local）；当它们分布在不同的进程中执行时，就是分布式（Distributed）模式。

6.3 编程接口

那么，Job 和 Task 的逻辑应是怎么对应到具体的代码中的？

首先，插件的入口类必须扩展 Reader 或 Writer 抽象类，并且实现分别实现 Job 和 Task 两个内部抽象类，Job 和 Task 的实现必须是 **内部类**的形式，原因见 **加载原理**一节。以 Reader 为例：

```
public class SomeReader
        extends Reader
{
    public static class Job
            extends Reader.Job
    {
        @Override
        public void init()
        {
        }

        @Override
        public void prepare()
        {
        }

        @Override
        public List<Configuration> split(int adviceNumber)
        {
            return null;
        }

        @Override
        public void post()
```

(下页继续)

(续上页)

```
        {
        }

        @Override
        public void destroy()
        {
        }
    }

    public static class Task
            extends Reader.Task
    {

        @Override
        public void init()
        {
        }

        @Override
        public void prepare()
        {
        }

        @Override
        public void startRead(RecordSender recordSender)
        {
        }

        @Override
        public void post()
        {
        }

        @Override
        public void destroy()
        {
        }
    }
}
```

Job 接口功能如下：

- init: Job 对象初始化工作，测试可以通过 super.getPluginJobConf() 获取与本插件相关的配置。读插件获得配置中 reader 部分，写插件获得 writer 部分。
- prepare: 全局准备工作，比如 mysql 清空目标表。
- split: 拆分 Task。参数 adviceNumber 框架建议的拆分数，一般是运行时所配置的并发度。值返回的是 Task 的配置列表。
- post: 全局的后置工作，比如 mysql writer 同步完影子表后的 rename 操作。
- destroy: Job 对象自身的销毁工作。

Task 接口功能如下：

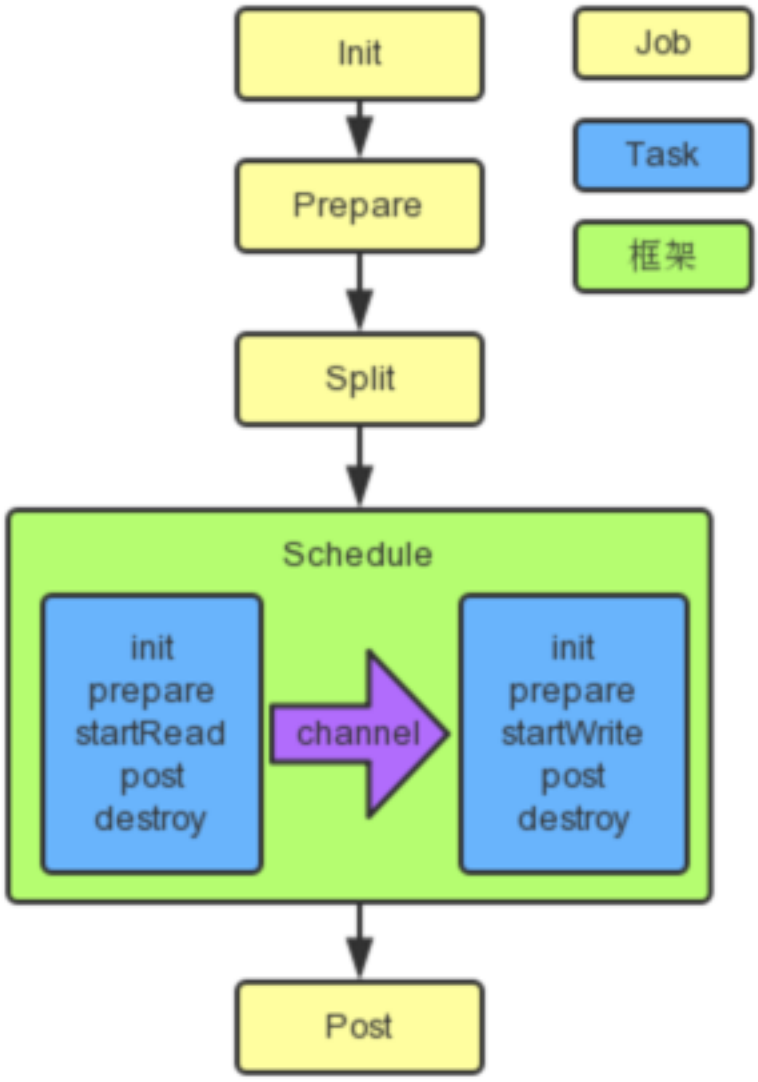
- init：Task 对象的初始化。此时可以通过 super.getPluginJobConf() 获取与本 Task 相关的配置。这里的配置是 Job 的 split 方法返回的配置列表中的其中一个。
- prepare：局部的准备工作。

- `startRead`: 从数据源读数据，写入到 `RecordSender` 中。`RecordSender` 会把数据写入连接 Reader 和 Writer 的缓存队列。
- `startWrite`: 从 `RecordReceiver` 中读取数据，写入目标数据源。`RecordReceiver` 中的数据来自 Reader 和 Writer 之间的缓存队列。
- `post`: 局部的后置工作。
- `destroy`: Task 象自身的销毁工作。

需要注意的是：

- `Job` 和 `Task` 之间一定不能有共享变量，因为分布式运行时不能保证共享变量会被正确初始化。两者之间只能通过配置文件进行依赖。
- `prepare` 和 `post` 在 `Job` 和 `Task` 中都存在，插件需要根据实际情况确定在什么地方执行操作。

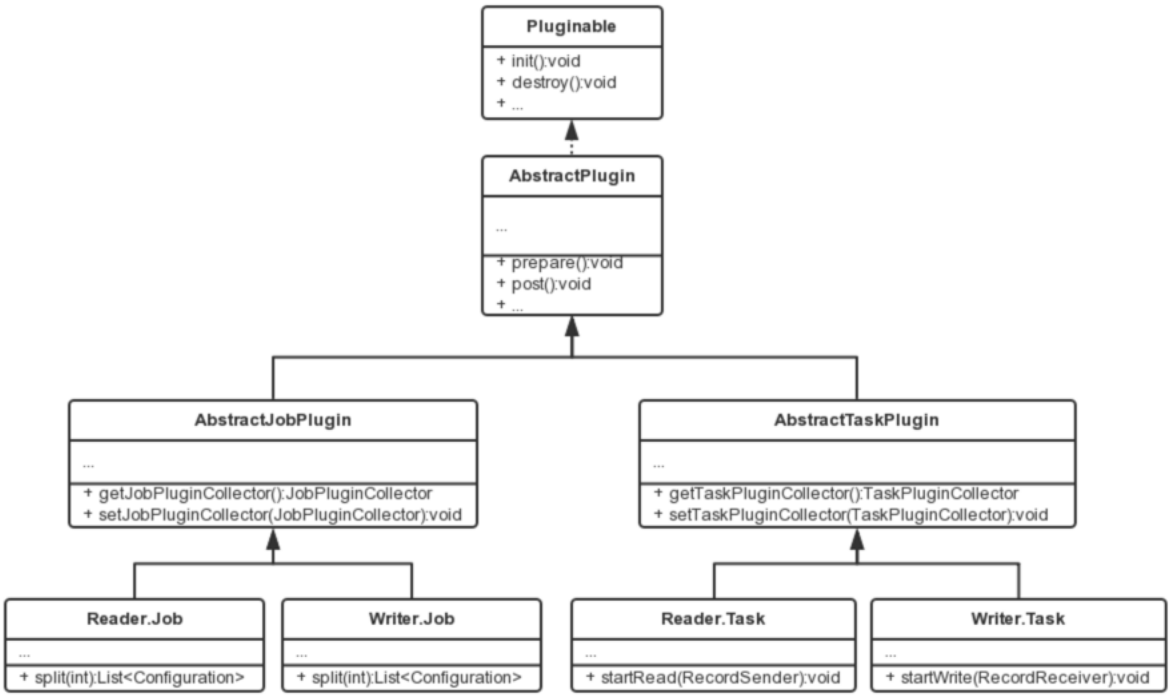
框架按照如下的顺序执行 `Job` 和 `Task` 的接口：



AddaxReaderWriter

上图中，黄色表示 `Job` 部分的执行阶段，蓝色表示 `Task` 部分的执行阶段，绿色表示框架执行阶段。

相关类关系如下：



Addax

6.3.1 插件定义

代码写好了，有没有想过框架是怎么找到插件的入口类的？框架是如何加载插件的呢？

在每个插件的项目中，都有一个 `plugin.json` 文件，这个文件定义了插件的相关信息，包括入口类。例如：

```
{
  "name": "mysqlwriter",
  "class": "com.wgzhao.addax.plugin.writer.mysqlwriter.MysqlWriter",
  "description": "Use Jdbc connect to database, execute insert sql.",
  "developer": "wgzhao"
}
```

- `name`: 插件名称，大小写敏感。框架根据用户在配置文件中指定的名称来搜寻插件。**十分重要**。
- `class`: 入口类的全限定名称，框架通过反射穿件入口类的实例。**十分重要**。
- `description`: 描述信息。
- `developer`: 开发人员。

6.3.2 打包发布

Addax 使用 assembly 打包，打包命令如下：

```
mvn clean package
mvn package assembly:single
```

Addax 插件需要遵循统一的目录结构：

```
${ADDAX_HOME}
|-- bin
|   `-- addax.py
|-- conf
|   |-- core.json
|   `-- logback.xml
|-- lib
|   `-- addax-core-dependencies.jar
`-- plugin
    |-- reader
    |   `-- mysqlreader
    |       |-- libs
    |       |   `-- mysql-reader-plugin-dependencies.jar
    |       |-- mysqlreader-0.0.1-SNAPSHOT.jar
    |       `-- plugin.json
    `-- writer
        |-- mysqlwriter
        |   |-- libs
        |   |   `-- mysql-writer-plugin-dependencies.jar
        |   |-- mysqlwriter-0.0.1-SNAPSHOT.jar
        |   `-- plugin.json
        |-- oceanbasewriter
        `-- odpswriter
```

- ${ADDAX_HOME}/bin: 可执行程序目录。
- ${ADDAX_HOME}/conf: 框架配置目录。
- ${ADDAX_HOME}/lib: 框架依赖库目录。
- ${ADDAX_HOME}/plugin: 插件目录。

插件目录分为 reader 和 writer 子目录，读写插件分别存放。插件目录规范如下：

- ${PLUGIN_HOME}/libs: 插件的依赖库。
- ${PLUGIN_HOME}/plugin-name-version.jar: 插件本身的 jar。
- ${PLUGIN_HOME}/plugin.json: 插件描述文件。

尽管框架加载插件时，会把 ${PLUGIN_HOME} 下所有的 jar 放到 classpath，但还是推荐依赖库的 jar 和插件本身的 jar 分开存放。

注意：

插件的目录名字必须和 plugin.json 中定义的插件名称一致。

6.4 配置文件

Addax 使用 json 作为配置文件的格式。一个典型的 Addax 任务配置如下：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "odpsreader",
          "parameter": {
            "accessKey": "",
            "accessId": "",
            "column": [
              ""
            ],
            "isCompress": "",
            "odpsServer": "",
            "partition": [
              ""
            ],
            "project": "",
            "table": "",
            "tunnelServer": ""
          }
        },
        "writer": {
          "name": "oraclewriter",
          "parameter": {
            "username": "",
            "password": "",
            "column": [
              "*"
            ],
            "connection": [
              {
                "jdbcUrl": "",
                "table": [
                  ""
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

Addax 框架有 core.json 配置文件，指定了框架的默认行为。任务的配置里头可以指定框架中已经存在的配置项，而且具有更高的优先级，会覆盖 core.json 中的默认值。

配置中 job.content.reader.parameter 的 value 部分会传给 Reader.Job；job.content.writer.parameter 的 value 部分会传给 Writer.Job，Reader.Job 和 Writer.Job 可以通过 super.getPluginJobConf() 来获取。

Addax 框架支持对特定的配置项进行 RSA 加密，例子中以 * 开头的项目便是加密后的值。**配置项加密解密过程对插件透明，插件仍然以不带 * 的 key 来查询配置和操作配置项**。

6.4.1 如何设计配置参数

配置文件的设计是插件开发的第一步！

任务配置中 `reader` 和 `writer` 下 `parameter` 部分是插件的配置参数，插件的配置参数应当遵循以下原则：

- 驼峰命名：所有配置项采用驼峰命名法，首字母小写，单词首字母大写。
- 正交原则：配置项必须正交，功能没有重复，没有潜规则。
- 富类型：合理使用 json 的类型，减少无谓的处理逻辑，减少出错的可能。
  - 使用正确的数据类型。比如，bool 类型的值使用 `true/false`，而非 `"yes"/"true"/0` 等。
  - 合理使用集合类型，比如，用数组替代有分隔符的字符串。
- 类似通用：遵守同一类型的插件的习惯，比如关系型数据库的 `connection` 参数都是如下结构：

```
{
  "connection": [
    {
      "table": [
        "table_1",
        "table_2"
      ],
      "jdbcUrl": [
        "jdbc:mysql://127.0.0.1:3306/database_1",
        "jdbc:mysql://127.0.0.2:3306/database_1_slave"
      ]
    },
    {
      "table": [
        "table_3",
        "table_4"
      ],
      "jdbcUrl": [
        "jdbc:mysql://127.0.0.3:3306/database_2",
        "jdbc:mysql://127.0.0.4:3306/database_2_slave"
      ]
    }
  ]
}
```

6.4.2 如何使用 `Configuration` 类

为了简化对 json 的操作，`Addax` 提供了简单的 DSL 配合 `Configuration` 类使用。

`Configuration` 提供了常见的 `get`, 带类型 `get`，带默认值 `get`，`set` 等读写配置项的操作，以及 `clone`, `toJSON` 等方法。配置项读写操作都需要传入一个 `path` 做为参数，这个 `path` 就是 `Addax` 定义的 DSL。语法有两条：

1. 子 map 用 `.key` 表示，`path` 的第一个点省略。
2. 数组元素用 `[index]` 表示。

比如操作如下 json：

```
{
  "a": {
    "b": {
      "c": 2
    },
    "f": [
      1,
      2,
      {
        "g": true,
        "h": false
      },
      4
    ]
  },
  "x": 4
}
```

比如调用 configuration.get(path) 方法，当 path 为如下值的时候得到的结果为：

- x: 4
- a.b.c: 2
- a.b.c.d: null
- a.b.f[0]: 1
- a.b.f[2].g: true

注意，因为插件看到的配置只是整个配置的一部分。使用 Configuration 对象时，需要注意当前的根路径是什么。

更多 Configuration 的操作请参考 ConfigurationTest.java。

6.5 插件数据传输

跟一般的 生产者-消费者模式一样，Reader 插件和 Writer 插件之间也是通过 channel 来实现数据的传输的。channel 可以是内存的，也可能是持久化的，插件不必关心。插件通过 RecordSender 往 channel 写入数据，通过 RecordReceiver 从 channel 读取数据。

channel 中的一条数据为一个 Record 的对象，Record 中可以放多个 Column 对象，这可以简单理解为数据库中的记录和列。

Record 有如下方法：

```
public interface Record
{
    // 加入一个列，放在最后的位置
    void addColumn(Column column);

    // 在指定下标处放置一个列
    void setColumn(int i, final Column column);

    // 获取一个列
    Column getColumn(int i);

    // 转换为 json String
```

(下页继续)

(续上页)

```
    String toString();

    // 获取总列数
    int getColumnNumber();

    // 计算整条记录在内存中占用的字节数
    int getByteSize();
}
```

因为 Record 是一个接口，Reader 插件首先调用 RecordSender.createRecord() 创建一个 Record 实例，然后把 Column 一个个添加到 Record 中。

Writer 插件调用 RecordReceiver.getFromReader() 方法获取 Record，然后把 Column 遍历出来，写入目标存储中。当 Reader 尚未退出，传输还在进行时，如果暂时没有数据 RecordReceiver.getFromReader() 方法会阻塞直到有数据。如果传输已经结束，会返回 null，Writer 插件可以据此判断是否结束 startWrite 方法。

Column 的构造和操作，我们在《类型转换》一节介绍。

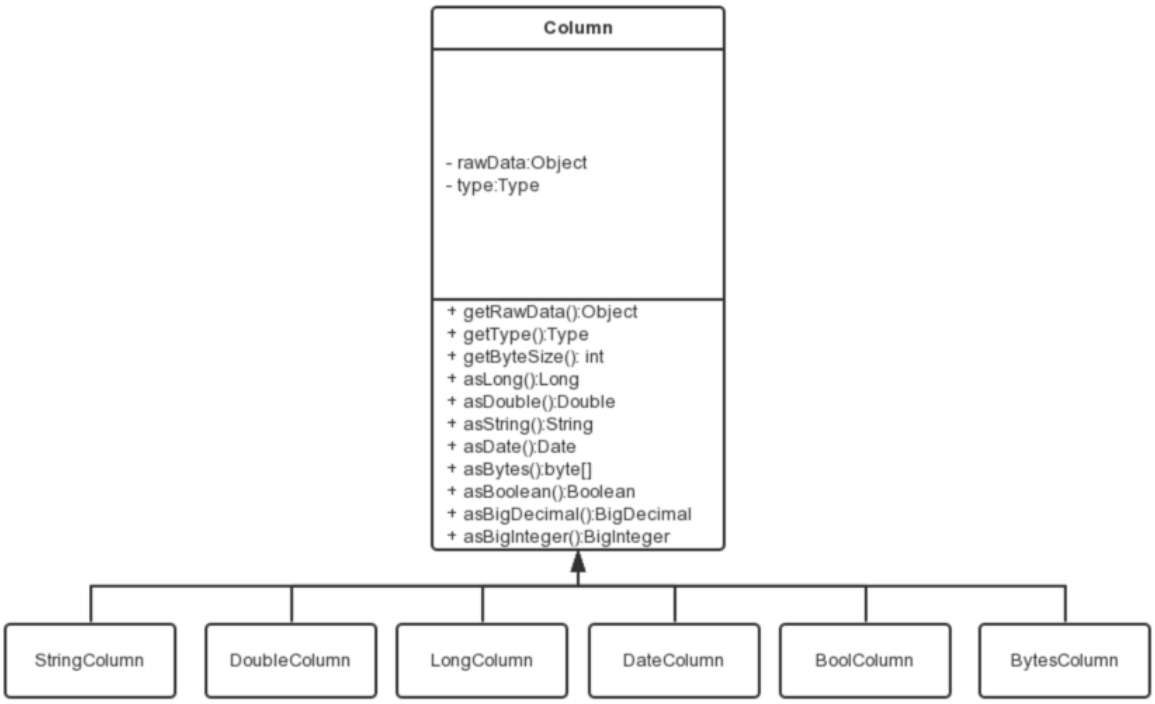
6.6 类型转换

为了规范源端和目的端类型转换操作，保证数据不失真，Addax 支持六种内部数据类型：

- Long：定点数 (Int、Short、Long、BigInteger 等)。
- Double：浮点数 (Float、Double、BigDecimal(无限精度) 等)。
- String：字符串类型，底层不限长，使用通用字符集 (Unicode)。
- Date：日期类型。
- Bool：布尔值。
- Bytes：二进制，可以存放诸如 MP3 等非结构化数据。

对应地，有 DateColumn、LongColumn、DoubleColumn、BytesColumn、StringColumn 和 BoolColumn 六种 Column 的实现。

Column 除了提供数据相关的方法外，还提供一系列以 as 开头的数据类型转换转换方法。



Columns

Addax 的内部类型在实现上会选用不同的 java 类型：

类型之间相互转换的关系如下：

6.7 脏数据处理

6.7.1 什么是脏数据

目前主要有三类脏数据：

1. Reader 读到不支持的类型、不合法的值。
2. 不支持的类型转换，比如：`Bytes` 转换为 `Date`。
3. 写入目标端失败，比如：写 mysql 整型长度超长。

6.7.2 如何处理脏数据

在 `Reader.Task` 和 `Writer.Task` 中，功过 `AbstractTaskPlugin.getPluginCollector()` 可以拿到一个 `TaskPluginCollector`，它提供了一系列 `collectDirtyRecord` 的方法。当脏数据出现时，只需要调用合适的 `collectDirtyRecord` 方法，把被认为是脏数据的 `Record` 传入即可。

用户可以在任务的配置中指定脏数据限制条数或者百分比限制，当脏数据超出限制时，框架会结束同步任务，退出。插件需要保证脏数据都被收集到，其他工作交给框架就好。

6.8 加载原理

1. 框架扫描 `plugin/reader` 和 `plugin/writer` 目录，加载每个插件的 `plugin.json` 文件。
2. 以 `plugin.json` 文件中 `name` 为 key，索引所有的插件配置。如果发现重名的插件，框架会异常退出。
3. 用户在插件中在 `reader/writer` 配置的 `name` 字段指定插件名字。框架根据插件的类型（`reader/writer`）和插件名称去插件的路径下扫描所有的 jar，加入 `classpath`。
4. 根据插件配置中定义的入口类，框架通过反射实例化对应的 `Job` 和 `Task` 对象。