链滴

DataX 预研

作者：hu244785010

原文链接：https://ld246.com/article/1683275803789

来源网站：链滴

<p><img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20200831.jpg?imageView2/1/w/960/h/540/interlace/1/q/100"></p>
<h2 id="DataX-预研">DataX 预研</h2>
<h2 id="预研背景">预研背景</h2>
<p>DataX 是一个异构数据源离线同步工具。</p>
<p>现场使用 ADMA 调度 DataX 来进行数据同步，计划用 D-Dolphin 替代 ADMA，因此 D-Dolph
n 也需要支持 DataX。</p>
<p>我们已经给 D-Dolphin 开发了 Synchronous 任务，与 DataX 相比，Synchronous 任务支持
数据源类型比较少。</p>
<p>D-Dolphin 原生已经支持 DataX 任务了，不过还比较简陋，因此需要对这个任务进行改进。</
>
<h2 id="DataX-介绍">DataX 介绍</h2>
<p>DataX 是阿里巴巴开源的一个异构数据源离线同步工具，致力于实现包括关系型数据库（MySQ
、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
。</p>
<p>项目地址：<a href="https://ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fal
baba%2FDataX" target="_blank" rel="nofollow ugc">https://github.com/alibaba/DataX
/p>
<h3 id="设计理念">设计理念</h3>
<p>为了解决异构数据源同步问题，DataX 将复杂的网状的同步链路变成了<strong>星型数据链路<
strong>，DataX 作为中间传输载体负责连接各种数据源。当需要接入一个新的数据源的时候，只需
将此数据源对接到 DataX，便能跟已有的数据源做到无缝数据同步。如下图所示：</p>
<p><img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2023/05/image-uqh2s5b.png?imageView2/2/interlace/1/format/jpg"></
>
<h3 id="支持的数据源插件">支持的数据源插件</h3>
<p>DataX 目前已经有了比较全面的插件体系，主流的 RDBMS 数据库、NOSQL、大数据计算系统
已经接入，如下表所示：</p>
<table>
<thead>
<tr>
<th align="left">类型</th>
<th align="left">数据源</th>
<th align="left">Reader(读)</th>
<th align="left">Writer(写)</th>
<th align="left">文档</th>
</tr>
</thead>
<tbody>
<tr>
<td align="left">RDBMS 关系型数据库</td>
<td align="left">MySQL</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left"><a href="https://ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2
alibaba%2FDataX%2Fblob%2Fmaster%2Fmysqlreader%2Fdoc%2Fmysqlreader.md" target="_
lank" rel="nofollow ugc">读 、<a href="https://ld246.com/forward?goto=https%
A%2F%2Fgithub.com%2Falibaba%2FDataX%2Fblob%2Fmaster%2Fmysqlwriter%2Fdoc%2Fmy
qlwriter.md" target="_blank" rel="nofollow ugc">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Oracle</td>
<td align="left">√</td>

<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">OceanBase</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">SQLServer</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">PostgreSQL</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">DRDS</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>

<td align="left">Kingbase</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">通用 RDBMS(支持所有关系型数据库)</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left">阿里云数仓数据存储</td>
<td align="left">ODPS</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">ADB</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">ADS</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">OSS</td>
<td align="left">√</td>
<td align="left">√</td>

<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">OCS</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Hologres</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">AnalyticDB For PostgreSQL</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left">阿里云中间件</td>
<td align="left">datahub</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">SLS</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left">阿里云图数据库</td>
<td align="left">GDB</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、<a href="https://ld246.com/forward?goto=https%3A2F%2Fgithub.com%2Falibaba%2FDataX%2Fblob%2Fmaster%2Fgdbwriter%2Fdoc%2Fgdbwrit

r.md" target="_blank" rel="nofollow ugc">写</td>
</tr>
<tr>
<td align="left">NoSQL 数据存储</td>
<td align="left">OTS</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Hbase0.94</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Hbase1.1</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Phoenix4.x</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Phoenix5.x</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、<a href="https://ld246.com/forward?

oto=https%3A%2F%2Fgithub.com%2Falibaba%2FDataX%2Fblob%2Fmaster%2Fhbase20xsqlwriter%2Fdoc%2Fhbase20xsqlwriter.md" target="_blank" rel="nofollow ugc">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">MongoDB</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Cassandra</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left">数仓数据存储</td>
<td align="left">StarRocks</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">ApacheDoris</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">ClickHouse</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Hive</td>
<td align="left">√</td>

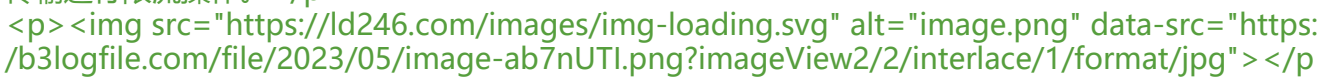
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">kudu</td>
<td align="left"></td>
<td align="left">√</td>
<td align="left">写</td>
</tr>
<tr>
<td align="left">无结构化数据存储</td>
<td align="left">TxtFile</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">FTP</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">HDFS</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">Elasticsearch</td>
<td align="left"></td>
<td align="left">√</td>

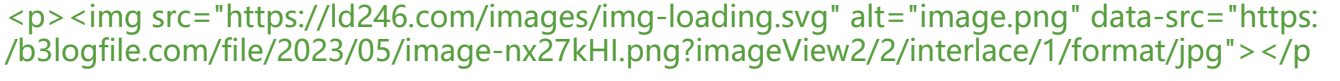
<td align="left">写</td>
</tr>
<tr>
<td align="left">时间序列数据库</td>
<td align="left">OpenTSDB</td>
<td align="left">√</td>
<td align="left"></td>
<td align="left">读</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">TSDB</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
<tr>
<td align="left"></td>
<td align="left">TDengine</td>
<td align="left">√</td>
<td align="left">√</td>
<td align="left">读 、写</td>
</tr>
</tbody>
</table>
DataX 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，只需要任意加上种插件，就能无缝对接其他数据源。详情请看：DataX 数据源指南
<h3 id="当前使用现状">当前使用现状</h3>
DataX 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 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。
<h2 id="DataX-框架设计">DataX 框架设计</h2>
DataX 本身作为离线数据同步框架，采用 Framework + plugin&nsp;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。
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，则由框架来处理。

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，并将据写入到目的端。

Framework：Framework 用于连接 Reader 和 Writer，作为两者的数据输通道，并处理缓冲，流控，并发数据转换等核心技术问题。

<h3 id="核心架构">核心架构</h3>
DataX 3.0 开源版本支持单机多线程模式完成同步作业运行。
<h3 id="核心概念">核心概念</h3>
<h4 id="Job">Job</h4>
DataX 完成单个数据同步的作业，我们称之为 Job，DataX 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。DataX Job 模块是单个作业中枢管理节点，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 管等功能。
DataX 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 TaskGrop 任务完成后 Job 成功退出。否则，异常退出，进程退出值非 0。
<h4 id="Task">Task</h4>
Task 是 DataX 作业的最小单元，每一个 Task 都会负责一部分据的同步工作。DataX Job 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。
<h4 id="Task-Group">Task Group</h4>
Task Group 是一组 Task 的集合，根据 DataX 的公平分配策略，公平地配 Task 到对应的 TaskGroup 中。一个 TaskGroup 对应一个 TaskGroupContainer，负执行一组 Task。
Job、Task 和 Task Group 三者之间的关系可以用如下图表示：

根据切分策略将一个 Job 切分成多个 Task，根据分配策略<strng>将多个 Task 组成一个 TaskGroup。
<blockquote>
感觉 TaskGroup 没必要设计成多个，用一个来调度就够了，网上也没有相关的说法。个人推测在分布式环境下 TaskGroup 是运行在不同的机器上的，而开源版本是商业版本的阉割版，不支持分式，所以就显得这个设计有点多余。
</blockquote>
<h4 id="Channel">Channel</h4>
DataX 会单独启动一条线程运行一个 Task，而 Task 会持有一个 Channel 用作 Reader 与 Writer 的数据传输媒介，DataX 的数据流向都是按照 Reader—>Channel—&gt Writer 的方向流转。Channel 作为传输通道，即能充当缓冲层，同时还能对数传输进行限流操作。


<h4 id="Transformer">Transformer</h4>
DataX 的 transformer 模式同时还提供了强大的数据转换功能，DataX 默提供了丰富的数据转换实现类，用户还可以根据项目自身需求，扩展数据转换。


<h3 id="DataX调度流程">DataX 调度流程</h3>
举例来说，用户提交了一个 DataX 作业，并且配置了 20 个并发，目的是一个 100 张分表的 mysql 数据同步到 odps 里面。 DataX 的调度决策思路是

DataX Job 根据分库分表切分成了 100 个 Task。
根据 20 个并发，DataX 计算共需要分配 4 个 TaskGroup（每个 TaskGroup 默认最大 5 个并

）。
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 Task。

<h2 id="DataX-六大核心优势">DataX 六大核心优势</h2>
<h3 id="可靠的数据质量监控">可靠的数据质量监控</h3>
<h5 id="完美解决数据传输个别类型失真问题">完美解决数据传输个别类型失真问题</h5>
DataX 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 DataX 3.0 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，让数据可以完整无的传输到目的端。
<h5 id="提供作业全链路的流量-数据量运行时监控">提供作业全链路的流量、数据量运行时监控</h>
DataX3.0 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，予用户更多性能排查信息。
<h5 id="提供脏数据探测">提供脏数据探测</h5>
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)，这数据 DataX 认为就是脏数据。DataX 目前可以实现脏数据精确过滤、识别、采集、展示，为用户提多种的脏数据处理模式，让用户准确把控数据质量大关！
<h3 id="丰富的数据转换功能">丰富的数据转换功能</h3>
DataX 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，另外还提供自动 groovy 函数，让用户自定义转换函数。详情请看 DataX 的 transformer 详细介绍。
<h3 id="精准的速度控制">精准的速度控制</h3>
还在为同步过程对在线存储压力影响而担心吗？新版本 DataX3.0 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}
<h3 id="强劲的同步性能">强劲的同步性能</h3>
DataX3.0 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 DataX 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，个作业一定可以打满网卡。另外，DataX 团队对所有的已经接入的插件都做了极致的性能优化，并且了完整的性能测试。性能测试相关详情可以参照每单个数据源的详细介绍：DataX 数据源指南
<h3 id="健壮的容错机制">健壮的容错机制</h3>
DataX 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报错停止。因此稳定性是 DataX 的基本要求，在 DataX 3.0 的设计中，重点完善了框架和插件的稳性。目前 DataX3.0 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<stron>重试，保证用户的作业稳定运行。

线程内部重试 DataX 的核心插件都经过团队的全盘 review，不同网络交互方式都有不同的重试策略。
线程级别重试 目前 DataX 已经可以实现 TaskFailover，针对于中失败的 Task，DataX 框架可以做到整个 Task 级别的重新调度。

<h3 id="极简的使用体验">极简的使用体验</h3>
部署和使用简单方便。
DataX 在运行日志中打印了大量信息，其中包括传输速度，Reader、Writer 性能，进程 CPU，JM 和 GC 情况等等。

传输过程中打印传输速度、进度等
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:

/b3logfile.com/file/2023/05/image-une0jzy.png?imageView2/2/interlace/1/format/jpg">

传输过程中会打印进程相关的 CPU、JVM 等
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:/b3logfile.com/file/2023/05/image-n5sII9h.png?imageView2/2/interlace/1/format/jpg">

在任务结束之后，打印总体运行情况
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:/b3logfile.com/file/2023/05/image-rcLacNz.png?imageView2/2/interlace/1/format/jpg">

<h3 id="全局配置">全局配置</h3>
DataX 的全局配置文件为 conf/core.json。
{ "entry": { // 这两个配置实际上是没用的，需要在启动命令那里设置 "jvm": "-Xms1 -Xmx1G", "environment": {} }, "common": { "column": { // 字段的日期转换式 "datetimeFormat": "yyyy-MM-dd HH:mm:ss", "timeFormat": "HH:mm:ss", "dateFormat": "yyyy-MM-dd", "extraFormats":["yyyyMMdd"], "timeZone": "GT+8", "encoding": "utf-8" } }, // 定义了全局的系统参数，如果不指定会使用默认值。 "core": { "dataXServer": { // 代码中没用到这些配置，应该是商业版本用的 "address": "http://localhost:7001/api", "timeout": 10000, "reportDataxLog": false, "reportPerfLog": false }, "transport": { "channel": { // 用于统计和限，目前只有这个实现类 "class": "com.alibaba.datax.core.transport.channel.memory.MeoryChannel", "speed": { // 流控参数，限制通道的默认传输速率，-1 表示不制。 "byte": -1, // 流控参数，限制通道的传输记录个数，-1 表示不限制。 "record": -1 }, // 流控采样间隔 "flowControlInterval": 20, // 限制 channel 中队列的大小，即最多缓存的 record 个数。 "capacity": 512, // 限制 record 占用的内存大小，单位为字节。默认值为 64MB，如果不指定此参数，则占用内大小会被配置为 8MB。 "byteCapacity": 67108864 }, "exchanger": { // Reader 和 Writer 交换数据的桥梁 "class": "com.alibaba.datax.core.plugin.BufferedRecordExchanger", // Record 缓冲区大小 "bufferSize": 32 } }, "container": { "job": { // 这两个参数控制汇报当前状态的时间间隔，要减小汇报的间间隔，需要同时调小这两个参数 "reportInterval": 10000, "sleepInterval": 1000 }, "taskGroup": { // 单个 TaskGroup 的最大并发数 "channel": 5 }, "trace": { // 是否打印详细的统计信息 "enable": "false" } }, "statistics": { "collector": { "plugin": { // 用于记录脏数的类，这个类的作用是把脏数据打印到控制台中，目前只有这个实现类 "taskClass": "co.alibaba.datax.core.statistics.plugin.task.StdoutPluginCollector", // 限制最多打印多条脏数据日志，防止脏数据太多而刷屏 "maxDirtyNumber": 10 } } }}
<code>capacity</code> 和 <code>byteCapacity</code> 两个参数决定每个 channel 能缓存的记录数量和内存占用情况。如果要调整相应参数，您需要按照 DataX 实际的行环境进行配置。例如 MySQL 中每个 record 都比较大，那么可以考虑适当调高 <code>byteCapacity</code>，调整该参数时还请同时考虑机器的内存情况。
<h3 id="作业配置">作业配置</h3>
DataX 的作业配置文件一般在 job 目录下，在启动 DataX 时候需要指定作业配置文件的路径。
下面是一个从 PostgreSQL 数据库同步到 PostgreSQL 的作业配置文件：
{ "job": { "setting": { "speed": { // 指定 channel 个数，该参数与并发数密切相关。 "channel": 3 // 设置传输速度，单位为 byte/s，DataX 运行会尽可能达到该速度但是不超过它. "byte": 1048576 }, // 出错限制 "errorLimit": { // 出错的 record 条数上限，当大

该值即报错。 "record": 0, // 出错的 record 百分比上限 1.0 表示 100%，0.02 表示 2% "percentage": 0.02 } }, // 数据源信息 "content": [{ "reader": { // 读取 PostgreSQL "name": "postgresqlreader", "parameter": { // 数据库连接用户名 "username": "xx", // 数据库连接密码 "password": "xx", // table 模式下可以定需要查询的列 "column": ["id", "name"], // 切分主键 "splitPk": "id", "connection": [{ // jdbc 连接串，配多个的话是 HA，询可用的 IP，而不是同时从多个数据源读数据 "jdbcUrl": ["jdbc:postgresql://ost:port/database"], // 读取的表名 "table": ["table"] }] } }, "writer": { // 写入到 PostgreSQL "name": "ostgresqlwriter", "parameter": { "username": "xx", "password": "xx", "olumn": ["id", "name"], // 写入数据到目的表前，会先执行这里的准语句，常用于清理旧数据 "preSql": ["delete from test"], // 写入据到目的表后，会执行这里的标准语句 "postSql": [], // 一次性批量提交的记录数大小该值可以极大减少 DataX 与 PostgreSql 的网络交互次数，并提升整体吞吐量。 // 但是该值置过大可能会造成 DataX 运行进程 OOM 情况。 "batchSize": 2000, "connection": [{ // jdbc 连接串 "jdbcUrl": "jdbc:/datax", // 写入的表名 "table": ["test"] }] } } }] }}

<h3 id="数据分片">数据分片</h3>

在 job.json 中指定了 <code>splitPk</code>&nbs;字段，DataX 会将表中数据按照 <code>splitPk</code> 切分成 n 段，一个 Task 责一段数据的同步工作。

由于 DataX 的实现方式是按照 <code>splitPk</code> 字段分段查询数据库表那么 <code>splitPk</code> 字段的选取应该尽可能选择分布均匀且有索的字段，例如主键 ID、唯一键等字段。如果不指定 <code>splitPk</code>&nbs;字段，则 DataX 将不会进行数据的切分，并行度会变为 1。

目前 <code>splitPk</code> 仅支持切分主键为一个，并且类型为整或者字符串类型，如果用户指定其他非支持类型，PostgresqlReader 将报错！

<blockquote>
如果用 querySQL 模式，task 数量为固定为 sql 语句的条数，在此式下 channel 的最大值即为 sql 语句的条数。
</blockquote>

<blockquote>
为了保证同步数据的一致性，要么不配置 <code>splitPk</code> 字段使用单线迁移数据，要么确保数据迁移期间停止该 MySQL 数据库的服务。
</blockquote>

例如，配置 channel 数为 2，splitPk 为 <code>id</code>，则数据分片的执行日志为：

023-02-02 22:09:11.157 [job-0] INFO SingleTableSplitUtil - split pk [sql=SELECT MIN(id), AX(id) FROM datax_user] is running... 2023-02-02 22:09:11.255 [job-0] INFO SingleTableSplit til - After split(), allQuerySql=[select id,username,age,create_time from datax_user where (1 lt;= id AND id < 10001) select id,username,age,create_time from datax_user where (10001 <= id AND id < 20001) select id,username,age,create_time from datax_user where (2000 <= id AND id < 30001) select id,username,age,create_time from datax_user where (300 1 <= id AND id < 40001) select id,username,age,create_time from datax_user where (40 01 <= id AND id < 50001) select id,username,age,create_time from datax_user where (5 001 <= id AND id < 60001) select id,username,age,create_time from datax_user where (0001 <= id AND id < 70001) select id,username,age,create_time from datax_user where 70001 <= id AND id < 80001) select id,username,age,create_time from datax_user where (80001 <= id AND id < 90001) select id,username,age,create_time from datax_user where (90001 <= id AND id <= 100000) select id,username,age,create_time from datax_user w ere id IS NULL].
关系型数据库在单表情况下的分片数计算算法：<code>分片数=channel数*分片系数+1</code>，片系数（splitFactor）可以在配置文件中修改，默认值为 5；<code>+1</code> 是为了同分片键为 null 的数据。上例中的分片数=2×5+1=11。

多表和其他数据源类型的分片数计算算法在此不做分析，感兴趣的可以看源码：<code>com.alibaba.datax.core.job.JobContainer#doReaderSplit</code>
<h3 id="动态传参">动态传参</h3>
假设有 <code>table</code> 和 <code>where</code> 两个动态参数。
作业配置使用参数：
{ "reader": { "name": "mysqlreader", "parameter": { "username": "root", "password": "root", "column": ["id"], "splitPk": "", "connection": [{ "table": ["{table}"], "jdbcUrl": ["***:3306/test"] }], "where": "{where}" } }}
启动命令中设置参数：
python /datax/bin/datax.py -p"-Dtable=test -Dwhere='1=1'"
<h2 id="DataX-脏数据处理">DataX 脏数据处理</h2>
<h3 id="什么是脏数据-">什么是脏数据？</h3>
目前主要有三类脏数据：

Reader 读到不支持的类型、不合法的值。
不支持的类型转换，比如：<code>Bytes</code> 转换为 <code>Date</code>。
写入目标端失败，比如：写 mysql 整型长度超长。

<h3 id="脏数据的展示">脏数据的展示</h3>
DataX 如果检测到脏数据，会在控制台中打印出来：
2023-02-01 16:32:20.390 [0-0-0-writer] ERROR StdoutPluginCollector - 脏数据: {"exception":"ERROR: null value in column "username" of relation "datax_user2" violates not-null constraint\n 详细：Failing row contains (11, null, 79, 2023-02-01 16:01:29.591).","record":[{"byteSize":2,"index":0,"rawData":11,"type":"LONG"},{"byteSize":0,"index":1,"type":"STRING"},{"byteSize":2,"index":2,"rawData":79,"type":"LONG"},{"byteSize":8,"index":3,"rawData":1675238489591,"type":"DATE"}],"type":"writer"}
在 Job 结束后也会打印脏数据的总数：

<h3 id="脏数据的处理">脏数据的处理</h3>
Job 支持用户对于脏数据的自定义监控和告警，包括对脏数据最大记录数阈值（record 值）或者脏数据占比阈值（percentage 值），当 Job 传输过程出的脏数据大于用户指定的数量/百分比，DataX Job 报错退出。
也可以使用 DataX 的数据转换功能来处理脏数据 。
退出时候的日志内容：
2023-02-01 16:41:31.710 [job-0] ERROR Engine - 经 DataX 智能分析,该任务最可能的错误因是:com.alibaba.datax.common.exception.DataXException: Code:[Framework-14], Description:[DataX 传输脏数据超过用户预期，该错误通常是由于源端数据存在较多业务脏数据导致，请仔细检查 DataX 汇报的脏数据日志信息, 或者您可以适当调大脏数据阈值 .]. - 脏数据条数检查不通过，限制是[]条，但实际上捕获了[2]条.
<h3 id="单条记录过长导致脏数据">单条记录过长导致脏数据</h3>
如果数据行过长超过限制（默认为 64MB），会变成脏数据，这个时候需要调大 core.json 中 <code>core.transport.channel.byteCapacity</code> 参数。
2023-02-02 11:04:26.370 [0-0-16-reader] ERROR StdoutPluginCollector - 脏数据: {"exception":"单条记录超过大小限制，当前限制为:6","record":[{"byteSize":5,"index":0,"rawData":80009,"type":"LONG"},{"byteSize":12,"index":1,"rawData":"Rikki Colato","type":"STRING"},{"byteSize":2,"index":2,"rawData":15,"type":"LONG"},{"byteSize":8,"index":3,"rawData":1675238490584,"type":"DATE"}],"type":"reader"}2023-02-02 11:04:26.673 [0-0-18-reader] INFO CommonRdbmsReader$Task - Finished read record by Sql: [select id,username,age,create_time from datax_user wh

re (90001 <= id AND id < 95001)] jdbcUrl:[jdbc:/testdb].
<h3 id="脏数据导致降速">脏数据导致降速</h3>
脏数据会导致这一批次的数据回滚，改为单次提交，如果脏数据太多会导致频繁的回滚操作，进步让 JVM 触发 GC，所以要尽量避免脏数据。
2023-02-01 16:31:44.579 [0-0-0-writer] WARN CommonRdbmsWriter$Task - 回滚此次写, 采用每次写入一行方式提交. 因为:Batch entry 787 INSERT INTO datax_user2 (id,username,age,ceate_time) VALUES('11'::int,NULL::varchar,'79'::int4,'2023-02-01 16:01:29.591+08'::timestamp) as aborted: ERROR: null value in column "username" of relation "datax_user2" violates not-nul constraint 详细：Failing row contains (11, null, 79, 2023-02-01 16:01:29.591). Call getNextException to see other errors in the batch.
<h2 id="DataX-数据转换-Transformer-">DataX 数据转换（Transformer）</h2>
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，包括剪列、转换列等工作，可以借助 ETL 的 T 过程实现(Transformer)。ataX 包含了完整的 E(Extract)、T(Transformer)、L(Load)支持。
运行模型：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2023/05/image-6gggKnd.png?imageView2/2/interlace/1/format/jpg">
支持的转换函数手册：DataX/transformer.md at master · alibaba/DataX · GitHub
<h2 id="DataX-同步Hive">DataX 同步 Hive</h2>

写入过程中失败的话会有残留数据，需要手动删除，这个可以通过 D-Dolphin 来实现自动删除；
字段有特殊字符比如分隔符的话需要使用 Transformer 进行数据转换，或者不用 <code>EXTFILE</code> 格式；
不要在 Windows 环境运行 HDFS 的同步任务，因为 Windows 环境的路径分隔符是 <code>\</code>，而 HDFS 路径的分隔符是 <code>/</code>，拼接路径的时候会出现问题；
DataX 没有直接支持 Hive，而是通过 HDFS 来实现 Hive 数据库的同步，为了方便用户使用，在 D-Dolphin 中需要读取 Hive 表的元信息，然后再填充到 DataX 作业的配置信息中；
对 Hive 分区表的支持不好，需要自己预先创建分区的路径，可以通过 D-Dolphin 来实现自动创；
同步完后 Hive 最好执行一下 <code>MSCK REPAIR TABLE</code> 命令，修元数据信息，因为如果不是通过 hive 的 insert 等插入语句， 很多分区信息在 metastore 中是没有，这步可以让 D-Dolphin 在同步完成后自动执行；

<h2 id="DataX-调优">DataX 调优</h2>
首先我们知道，传输受两个因素影响：

网络本身的带宽等硬件因素造成的影响；
DataX 本身的参数。

即当觉得 DataX 传输速度慢时，需要从上述两个方面着手开始排查。
<h3 id="网络本身的带宽等硬件因素造成的影响">网络本身的带宽等硬件因素造成的影响</h3>
此部分主要需要了解网络本身的情况，即从源端到目的端的带宽是多少（实际带宽计算公式），时使用量和繁忙程度的情况，从而分析是否是本部分造成的速度缓慢。以下提供几个思路：

可使用从源端到目的端 <code>scp</code>，<code>python http</code>, <ode>nethogs</code> 等观察实际网络及网卡速度；
结合监控观察任务运行时间段时，网络整体的繁忙情况，来判断是否应将任务避开网络高峰运行

观察任务机的负载情况，尤其是网络和磁盘 IO，观察其是否成为瓶颈，影响了速度。

<h3 id="提高每个-Channel-的速度">提高每个 Channel 的速度</h3>
在 DataX 内部对每个 Channel 会有严格的速度控制，分两种，一种是控制每秒同步的记录数，外一种是每秒同步的字节数，默认值是-1（无限制）；如果设置了这个值，可以根据具体硬件情况适调大 byte 速度或者 record 速度，一般设置 byte 速度。
<h3 id="提高-Channel-个数">提高 Channel 个数</h3>
<h4 id="方式一-配置全局-Byte-限速以及单-Channel-Byte-限速">方式一：配置全局 Byte 限速以单 Channel Byte 限速</h4>
Channel 个数 = 全局 Byte 限速 / 单 Channel Byte 限速
全局：
{ "core": { "transport": { "channel": { "speed": { "byte": 1048576 } } } } }
单个：
{ "job": { "setting": { "speed": { "byte": 5242880 } } } }
<code>core.transport.channel.speed.byte</code>=1048576，<code>job.setting.speed.byte/code>=5242880，所以 Channel 个数 = 全局 Byte 限速 / 单 Channel Byte 限速=5242880/1048576=5 个
<h4 id="方式二-配置全局-Record-限速以及单-Channel-Record-限速">方式二：配置全局 Record 限速以及单 Channel Record 限速</h4>
Channel 个数 = 全局 Record 限速 / 单 Channel Record 限速
<h4 id="方式三-直接配置-Channel-个数--优先级最低-">方式三：直接配置 Channel 个数 (优先级低)</h4>
{ "job": { "setting": { "speed": { "channel" : 5 } } }}
只有在上面两种未设置才生效，上面两个同时设置是取值最小的作为最终的 channel 数。
<blockquote>
Channel 个数并不是越多越好，原因如下：

Channel 个数的增加，带来的是更多的 CPU 消耗以及内存消耗。
如果 Channel 并发配置过高导致 JVM 内存不够用，会出现的情况是发生频繁的 Full GC，导出度会骤降，适得其反。

</blockquote>
<h3 id="提高批次大小">提高批次大小</h3>
适当的提高批量写入的批次大小（batchSize），也可以有效地提高吞吐率。
以 PostgreSQL 为例：
{ "job": { "content": [{ "writer": { "name": "postgresqlwriter", "parameter": { "batchSize": 2048 } } }] } }
batchSize

描述：一次性批量提交的记录数大小，该值可以极大减少 DataX 与 PostgreSql 的网络交互次数并提升整体吞吐量。但是该值设置过大可能会造成 DataX 运行进程 OOM 情况。
必选：否
默认值：1024

<h3 id="提高-JVM-堆内存">提高 JVM 堆内存</h3>
提升 DataX Job 内 Channel 并发数时，内存的占用会显著增加，因为 DataX 作为数据交换通，在内存中会缓存较多的数据。例如 Channel 中会有一个 Buffer，作为临时的数据交换的缓冲区，在部分 Reader 和 Writer 的中，也会存在一些 Buffer，为了防止 OOM 等错误，调大 JVM 的堆内。
<h4 id="查看GC信息">查看 GC 信息</h4>
在 Job 开始执行前和执行完毕后控制台中会打印 GC 的信息，可根据这些信息来进行调优。
执行之前打印的信息：

2023-02-01 17:15:03.532 [main] INFO Engine - the machine info => osInfo: Graa VM Community 17 17.0.4+8-jvmci-22.2-b06 jvmInfo: Linux amd64 4.4.0-22621-Micro oft cpu num: 12 totalPhysicalMemory: -0.00G freePhysicalMemory: -0.0 G maxFileDescriptorCount: -1 currentOpenFileDescriptorCount: -1 GC Names [G1 Young Generation, G1 Old Generation] MEMORY_NAME | allocation_size | init_size CodeHeap 'profiled nmethods' | 117.21MB 2.44MB G1 Old Gen | 1,024.00MB | 970.00MB G1 Survivor Space | -0.00MB | 0.00MB CodeHeap 'non-profiled nmethods' | 117.21MB | 2.44MB C mpressed Class Space | 1,024.00MB | 0.00MB Metaspace | -0.00MB | 0.00MB G1 Eden Space | 0.00MB | 54.00MB CodeHeap 'non-nmethods' | 5.57MB | 2.44MB
执行完毕打印的信息：
2023-02-01 17:15:45.581 [job-0] INFO JobContainer - [total cpu info] => averageCpu | maxDeltaCpu | minDeltaCpu -1. 0% | -1.00% | -1.00% [total gc info] => NAME | totalGCCount | maxDeltaGCCount | minDeltaGCCount | total CTime | maxDeltaGCTime | minDeltaGCTime G1 Young Generation | 3 | 3 | 3 | 0.067s | 0.067s | 0.067s G1 Old Generation | 0 | 0 | 0 | 0.000s | 0.000s | 0.0 0s
<h4 id="调整-JVM-xms-xmx-参数的方式">调整 JVM xms xmx 参数的方式</h4>

一种是直接更改 datax.py 脚本中的 <code>DEFAULT_JV </code> 变量；

另一种是在启动的时候，加上对应的参数，如下：
python datax/bin/datax.py --jvm="-Xms8G -Xmx8G" XXX.json

<h2 id="DataX-的部署和使用">DataX 的部署和使用</h2>
参考文档：Da aX/userGuid.md at master · alibaba/DataX · GitHub
<h2 id="DataX-分布式">DataX 分布式</h2>
DataX 是阿里云 DataWorks 的开源版本，功能上有阉割，不支持分布式。网上也有一些 DataX 分布式的设计的文章，都是需要引入额外的调度平台和组件来实现，并且需要对 DataX 进行二次开 。
对于我们的 D-Dolphin 平台，目前可以实现对不同 DataX 作业的分布式调度，局限性是一个同 任务只能调度给一台 Worker 执行，没办法把一个同步任务分片后调度给多台 Worker 执 。
如果单个同步作业的数据量很大，单机性能不够的话，可以把一个同步作业拆分成多 同步作业，每个同步作业同步一部分数据，通过 <code>where</code> 条件手 进行数据分片，这种方法的问题就是不方便管理和监控，也不好实现高可用。
如果需要把一个同步任务分片后调度，需要对 DataX 进行深度定制化开发，开发难度比较大， 且我们不一定能改现场的 DataX 组件。
<h2 id="DataX-的约束和缺陷">DataX 的约束和缺陷</h2>

数据分片功能在数据分布不均的情况下的分片效果不好；
不支持实时数据同步；
不支持分布式，需要通过调度平台来实现；

对增量同步的支持不太好，需要自己传参来实现增量同步， 这个可以通过调度平台来解决；
DataX 导入过程存在三块逻辑，pre 操作、导入操作、post 操作，其中任意一环报错，DataX 业报错。由于 DataX 不能保证在同一个事务完成上述几个操作，因此有可能数据已经落入到目标端 也就是会有脏数据。

目前有两种解法，第一种配置 pre 语句，该 sql 可以清理当天导入数据， DataX 每次导入时候 以把上次清理干净并导入完整数据。 第二种，向临时表导入数据，完成后再 rename 到线上表。

<h2 id="自定义开发datax插件">自定义开发 datax 插件</h2>
https://github.com/2447850100/datax_parquet
<h2 id="结论">结论</h2>

DataX 支持的数据源很多，部署也比较简单，接入我们的调度平台后可以让我们的平台支持更多数据同步场景；
没有对 GBase 数据库的直接支持，应该可以使用 RDBMSReader 和 RDBMSWriter 来实现，这个需要测试一下；
对分布式的支持不好，单个作业的数据量很大的话速度可能会很慢，可以通过拆分作业来减轻这问题；
对 Hive 的支持不好，需要对 HDFS 插件进行二次开发；
接入我们的平台后需要在部署每个 Worker 机器的同时都部署一个 DataX，也可以通过 Worker 分组功能来只给某一批机器部署 DataX；

<h3 id="DataX-任务的改进方向">DataX 任务的改进方向</h3>

添加对 Hive 同步的支持；
添加对 table 模式的支持，原版只支持 querySql 模式，没法进行数据分片；
添加更多可设置的调优参数；
增加对脏数据的监控和展示，原版是直接输出到任务日志里，不够醒目；
DataX 实现分布式功能（待定），这个开发难度较大，需要对 DataX 进行深度定制开发；

<h3 id="FAQ">FAQ</h3>

需支持在 dolphin 创建 dataX 任务，当前业务场景是数据存储在 HDFS

Dolphin 原版已经支持 DataX 任务了，不过比较简陋，需要改进。

传输完数据后，需要在目标集群使用 Hive/Spark SQL 刷表分区、调用其他调度产品的某些接口

可以在我们的调研平台执行完同步作业后自动执行此操作。

源集群的数据需均匀的分布到多个 dataX 节点去处理

DataX 不支持分布式，单作业分布式运行的开发难度较大。

当前业务场景：31 个前置集群分别与 1 个核心集群通过 dataX 进行数据传输，dataX 节点部署核心集群，共 60 台服务器，日均约 303.69TB 的数据传输，dataX 任务也是通过核心的调度工具（AMA）发起调度，执行成功后自动触发后续任务

可通过 Worker 分组来控制 DataX 在核心集群运行，通过我们调度平台来配置后续任务。

<blockquote>
参考

DataX/introduction.md at master · alibaba/DataX · GitHub
图解 DataX 核心设计原理 - 讯云开发者社区-腾讯云 (tencent.com)
使用 ataX 同步 (aliyun.com)
DataX 的限速与调优 - UCloud 云社区
DataX/transformer.md at master · alibaba/DataX · GitHub
DataX 在有赞大数据平台的实践 (youzan.com
分布式 dataX 详细 (落地) 设计-今日头条 (toutiao.com)DataX 预研

</blockquote>