

Otter4使用介绍

七锋 2013-03-06

Agenda

1. 同步需求
2. 性能指标
3. 使用&运维

业务场景

1. 杭州/美国异地机房双向同步

- a. 业务性 (定义同步表, 同步字段)
- b. 隔离性 (定义同步通道, 对应一个具体业务, 多个通道之间互相隔离)
- c. 关联数据 (同步db数据的同时, 需要同步图片, 比如产品表)
- d. 双A写入 (避免回环同步, 冲突处理, 数据一致性保证)
- e. 事务性 (没有严格的事务保证, 定义表载入顺序)
- f. 异构性 (支持mysql/oracle)

2. 扩展业务

- a. 数据仓库增量数据 (整行记录, 根据变更主键反查)
- b. 业务cache更新 (更新db成功的同时, 刷新下cache中的值)
- c. 数据全库迁移 (建立任务队列表/触发全库记录变更)
- d. 多库合并同步 (product/product_detail需要尽可能保证加载顺序)

设计关注要点

硬性要求：

1. 数据不能丢失 (变更数据一定要成功应用到目标库)
2. 数据最终一致性 (双向两边记录要保证最终一致性)

客观因素：

1. 中美网络延迟 (平均200ms)
2. 中美传输速度 (2~6MB/s)
3. 文件同步 (20000条记录可达800MB文件)
4. 同步按需隔离 (不同业务之间同步互不影响,同步有快慢)
5. 事务性支持 (允许业务定义表的同步加载的顺序性)

otter目前支持了什么?

1. 单向同步

mysql/oracle互相同步

2. 双向同步

无冲突变更

3. 文件同步

本地/aranda文件

4. 双A同步

冲突检测&冲突补救

5. 数据迁移

中间表/行记录同步

otter初步性能指标

吞吐量：

1. insert 30~40w/min
2. delete 60w/min

latency：

1. 本地机房+单向同步 100ms
2. 中美机房+单向/双向同步 2s
3. 中美机房+文件 10s

重要：

1. load并行线程设置很重要，取决目标库载入能力
2. latency的几个经验值，要根据数据量和高峰期做继续评估

otter4 vs otter3

otter3 :

- a. 文件同步 1000 / min, 60MB/min
- b. 数据记录 20000 / min

otter4 :

- a. 文件同步 8000 / min, 500MB/min
- b. 数据记录 400000 / min

otter4相比于otter3, 是一个数量级上的飞跃

otter"慢"在哪里?

类似产品:

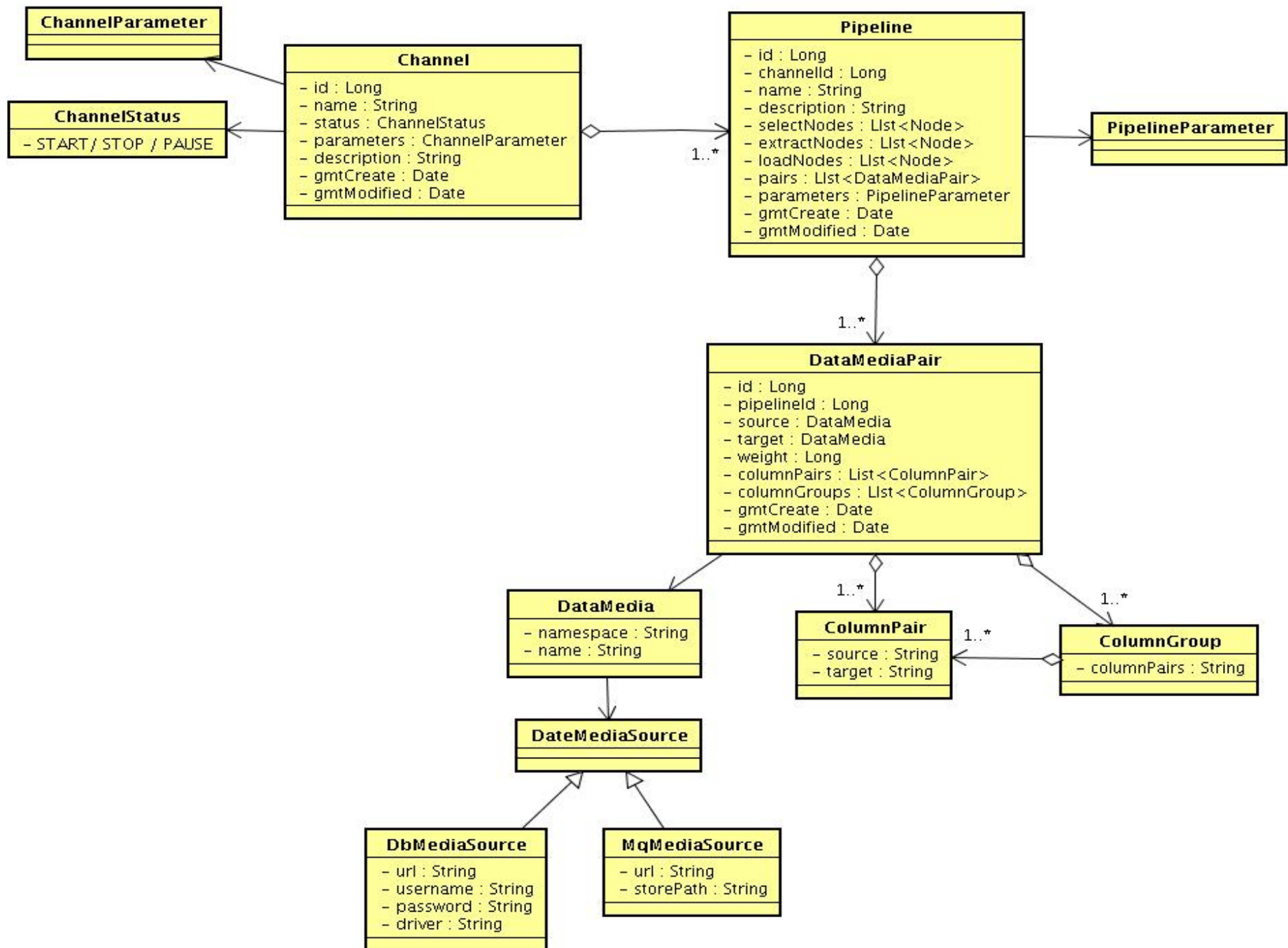
- a. 精卫 延迟<100ms
- b. drc 延迟<1s

otter"慢"点:

- a. 中美200ms延迟 vs 青岛70ms延迟
- b. 中美2~6MB带宽 vs 青岛千兆光纤

使用&运维

如何配置一个otter同步



名词解释

Pipeline：从源端到目标端的整个过程描述，主要由一些同步映射过程组成

Channel：同步通道，单向同步中一个Pipeline组成，在双向同步中有两个Pipeline组成

DateMediaPair：根据业务表定义映射关系，比如源表和目标表，字段映射，字段组等

DateMedia：抽象的数据介质概念，可以理解为数据表/mq队列定义

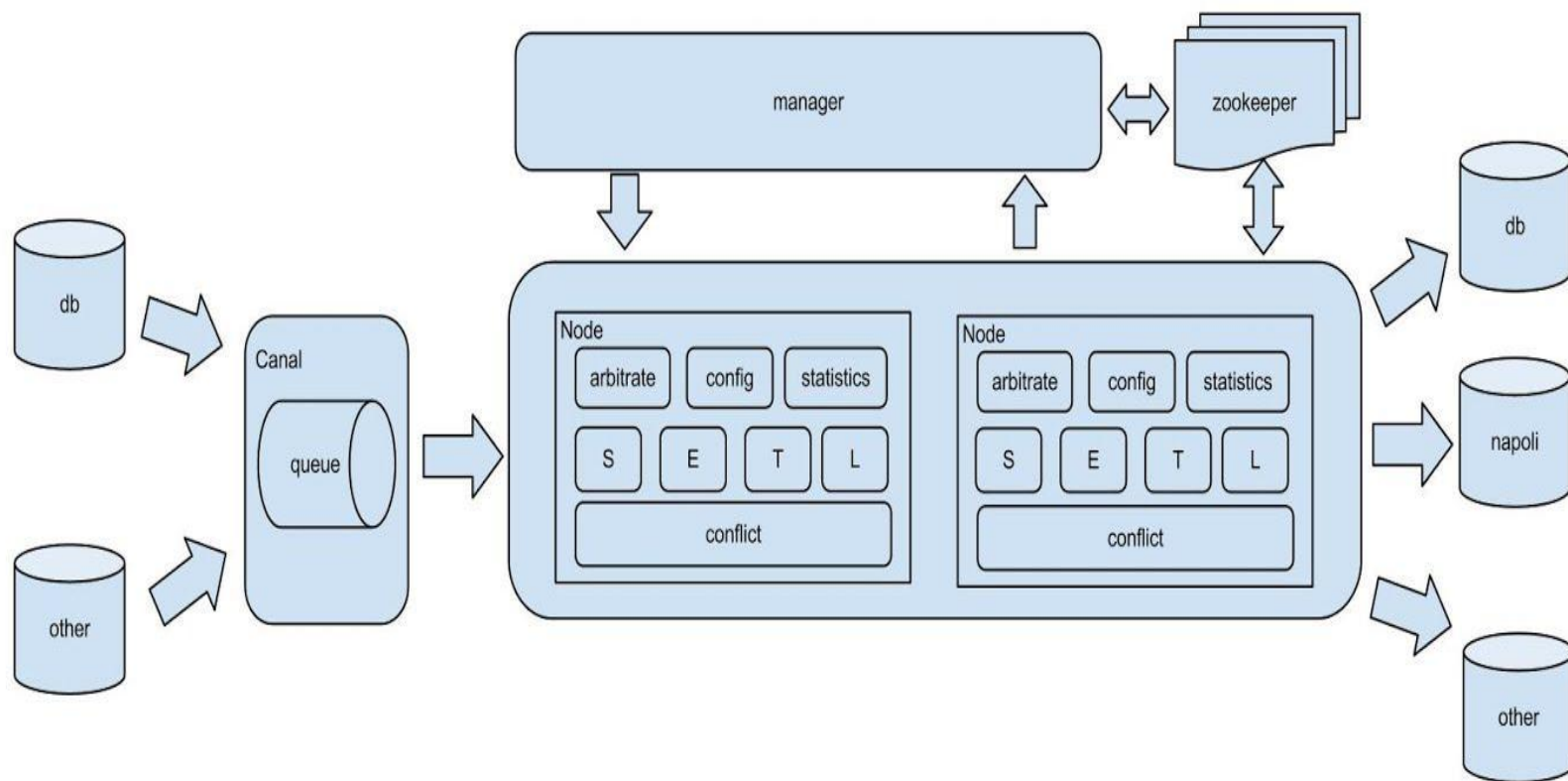
DateMediaSource：抽象的数据介质源信息，补充描述DateMedia

ColumnPair：定义字段映射关系

ColumnGroup：定义字段映射组

Node：处理同步过程的工作节点，对应一个jvm

整体架构



整体架构

otter整体模块

- manager (提供web页面进行同步管理)
- arbitrate (分布式调度，可跨IDC机房)
- node (同步过程setl)
- canal / eromanga (同步数据来源)

大集群化部署

- 1个manager集群 + 多个IDC机房node组成

1. 创建Channel

参数解释：

a. 同步一致性

i. 基于介质 (反查数据库获取字段当前值)

ii. 基于当前变更 (使用binlog的字段内容)

b. 同步模式

i. 列模式 (实际变更哪个字段，只同步变更字段)

ii. 行模式 (变更任意一个字段，目标库存在更新，不存在则插

入)

c. 冲突补救

i. 回环补救 (保证数据一致性的算法)

几种组合：

a. 基于当前变更 + 列模式 (常用，性能最高)

b. 基于当前变更 + 行模式 (全量数据订正，比如修改gmt_modified + 1秒)

c. 基于介质 + 行模式 (回退到某个时间点进行消费，不能让旧版本值覆盖目标库的新值)

数据一致性

业务场景：

- a. 多地写入
- b. 同一记录，同时变更

同一：具体到某一张表，某一条pk，某一字段

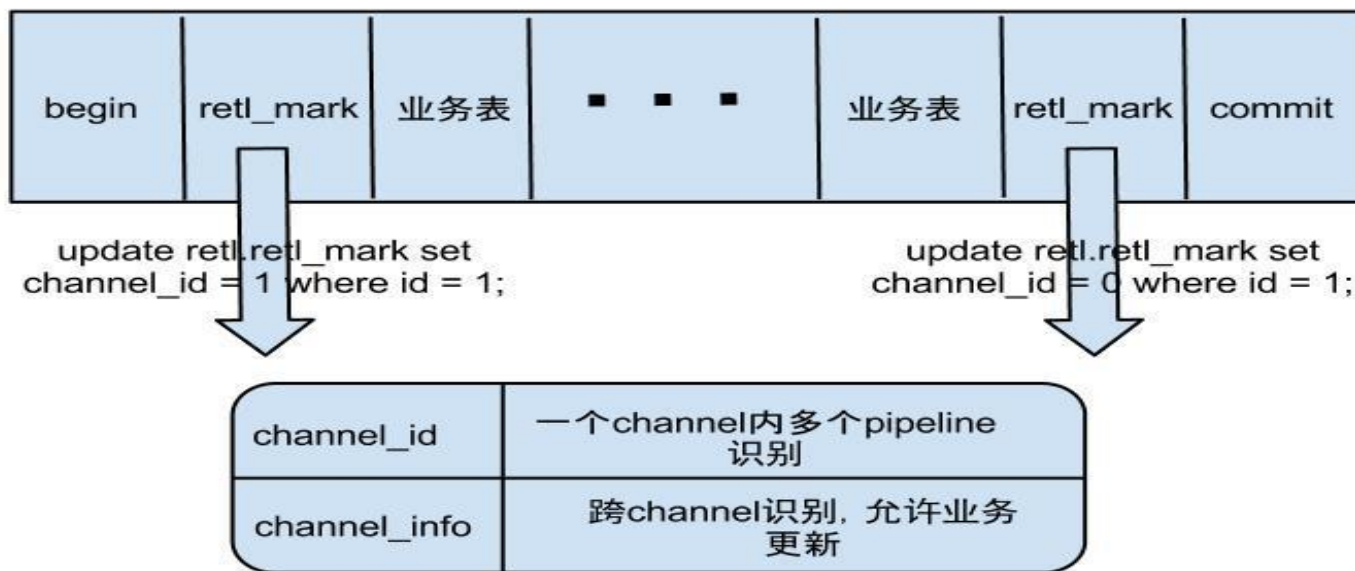
同时：A地写入的数据在B地还未可见的一段时间范围

方案：

- 1. 检测（事前处理）
- 2. 补救（事后处理）

双向同步(避免回环)

代表数据库的一个事务



几点注意:

1. `retl.ret_l_mark`表, 默认初始化1000条记录. 300一下属于otter内部系统使用, 300 ~ 1000, 属于业务系统使用

2. `retl_mark`表`channel_info`的变更需要和数据库当前值不一致, 否则会出现屏蔽同步失败 (mysql针对update前后值一样, 不记录binlog)

双向同步 (避免回环)

方案:

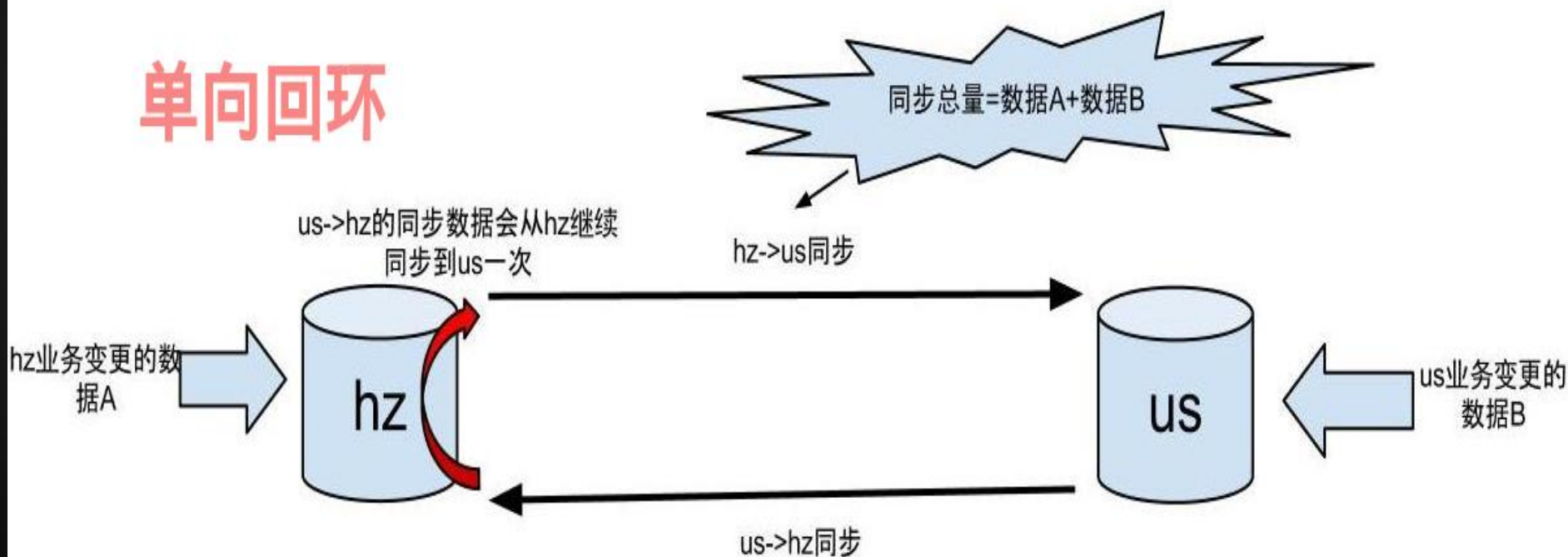
- a. 某方向pipeline同步时会在load时启用事务，头和尾都更新一次otter系统表
- b. 反方向pipeline获取到变更数据，解析事务头和尾辨别是否是otter产生

特殊业务场景： $A \leftrightarrow B \rightarrow C$ (A和B双向，B和C单向)

- a. A/B更新otter系统表产生的系统标识为A和B同步的channel id
- b. B->C的同步时，解析到A->B的系统标识，不是当前B->C的channelid，忽略系统标示，继续同步

双写同步(最终一致性)

单向回环



双写同步(最终一致性)

流程：

- us->hz同步的数据，会再次进入hz->us队列
- hz->us同步的数据，不会进入us->hz队列(回环终止)

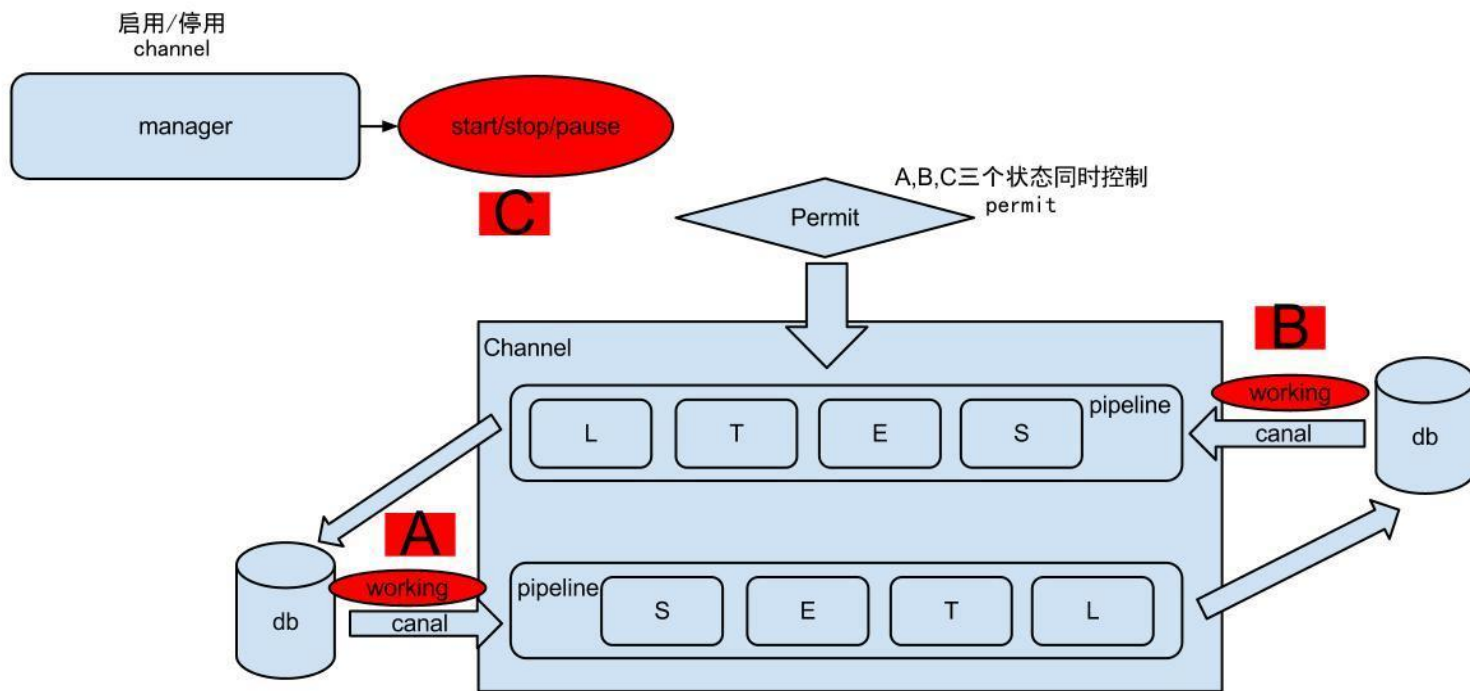
存在的问题：

a. 存在同步延迟时,会出现版本丢失/数据交替性变化

解决方案：

- a. 反查数据库同步 (以数据库最新版本同步，解决交替性)
- b. 字段同步 (降低冲突概率)
- c. 同步效率 (同步越快越好，降低双写导致版本丢失概率)

双写同步(最终一致性)



注意：A,B,C三点状态都正常才允许进行同步(解决数据单向覆盖)

2. 创建Pipeline

参数解释：

- a. 并行度
- b. 线程数 (数据库/文件同步)
- c. 是否主站点 (数据一致性，分站到主站回进行单边回环，建议将主要写入站点做为主站点)
- d. 同步数据来源
 - i. 目前仅支持canal (otter的另一个子项目，解析mysql binlog, 已开源)
 - ii. Destination名字：对应于canal的name，根据名字自动载入
 - iii. 消费端ID：目前随意设置，无要求
 - iii. 消费数据参数：
 - 1. 批次数量：根据数据变更量定义
 - 2. 超时时间：如果设置为-1，即时获取，有多少取多少
 - 3. 数据大小 (规划中)

canal设置

2. 创建Pipeline

e. 高级参数

1. 使用batch

- i. 针对tddl/cobar, 不支持batch模式

2. 是否跳过Load异常

- i. 忽略异常, 优先保证同步延迟

3. 仲裁器调度模式

- i. memory模式, 单机房同步, 效率最高, 开销 = 0ms
- ii. rpc模式, 跨机房同步, 多节点调度, 开销 = 2 * 中美网络延迟

迟

4. 负载均衡算法 (Stick粘性选择, 配合rpc模式, 调度开销最低)

5. 数据传输模式 (针对多节点同步, 小数据rpc, 大数据file + 多线程下

载)

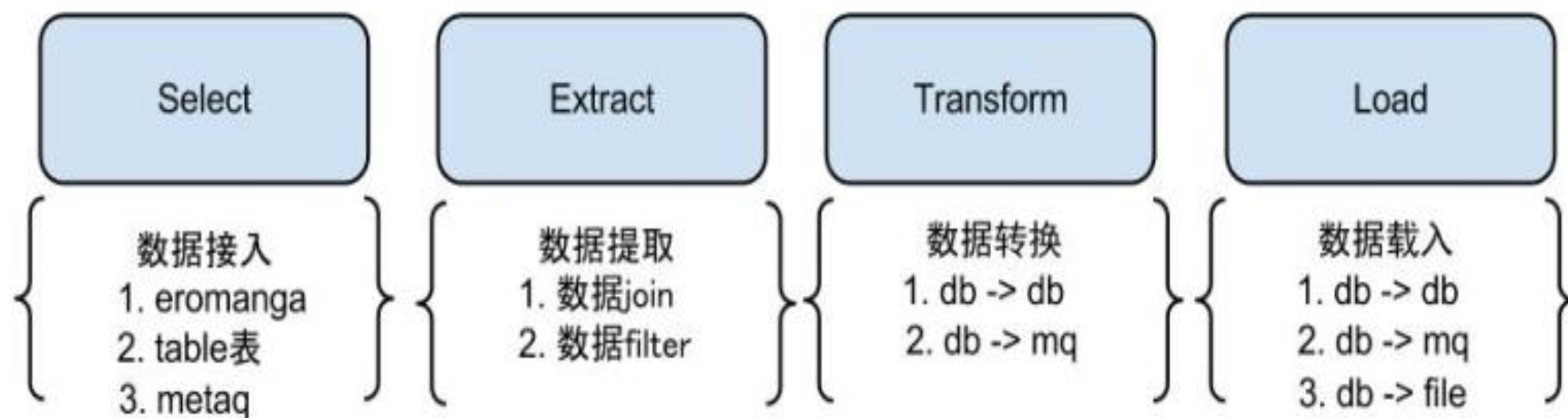
6. 日志记录

- i. select日志, mysql解析后的详细日志
- ii. load日志, 数据写入到数据库的记录(包含affect = 0 / 1的记录)

7. 文件重复对比 (兼容otter3的处理)

8. 跳过自由门数据

核心模块设计



核心模块设计

1. Select

解决不同源接入问题

2. Extract

解决数据join，数据filter，数据process

3. Transform

解决数据转换：字段映射，异构介质

4. Load

解决不同源输出问题

设计关注点

1. 如何解决extract/transform I/O瓶颈? ?

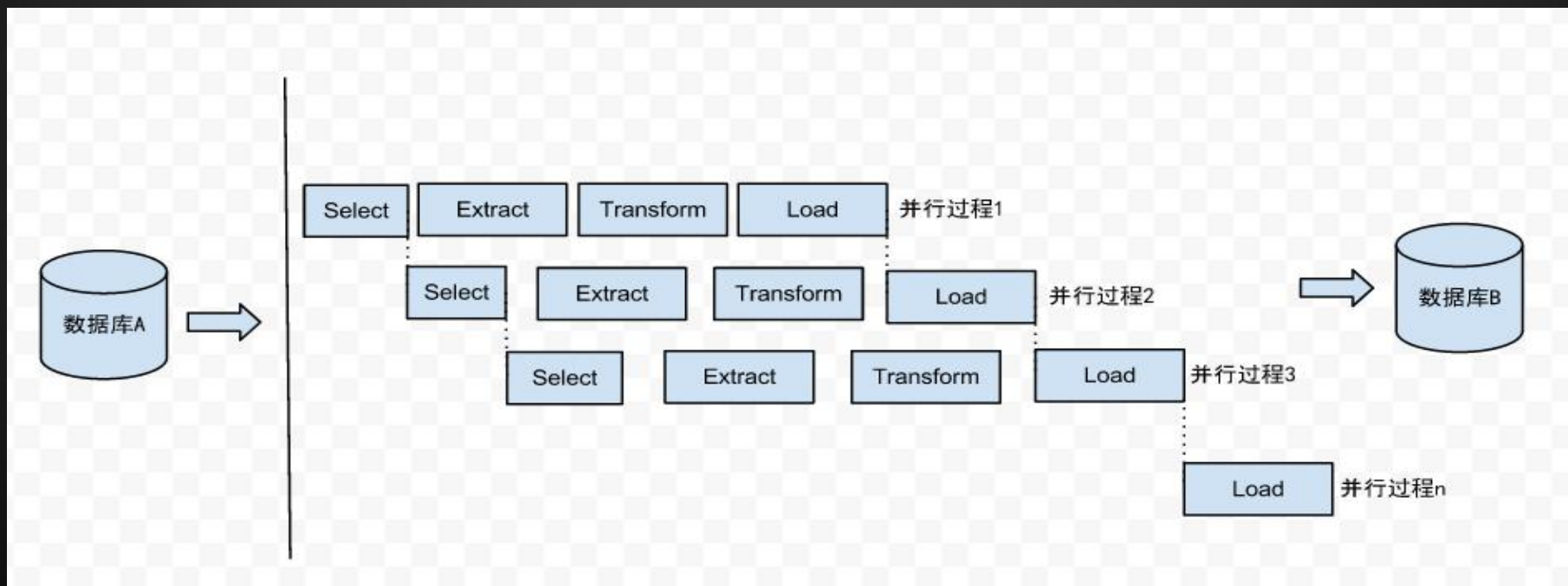
- 反查源数据库
- 中美网络延迟
- 附件打包/传输

2. 单机调度 or 分布式调度? 单节点 or 多节点?

- 存在中美跨机房同步
- 存在杭州同机房内数据同步

并行化调度

- a. select/load串行 (保证数据一定是按照select的顺序加载)
- b. extract/transform并行



解决: extract/transform I/O瓶颈, 减少latency

并行化调度

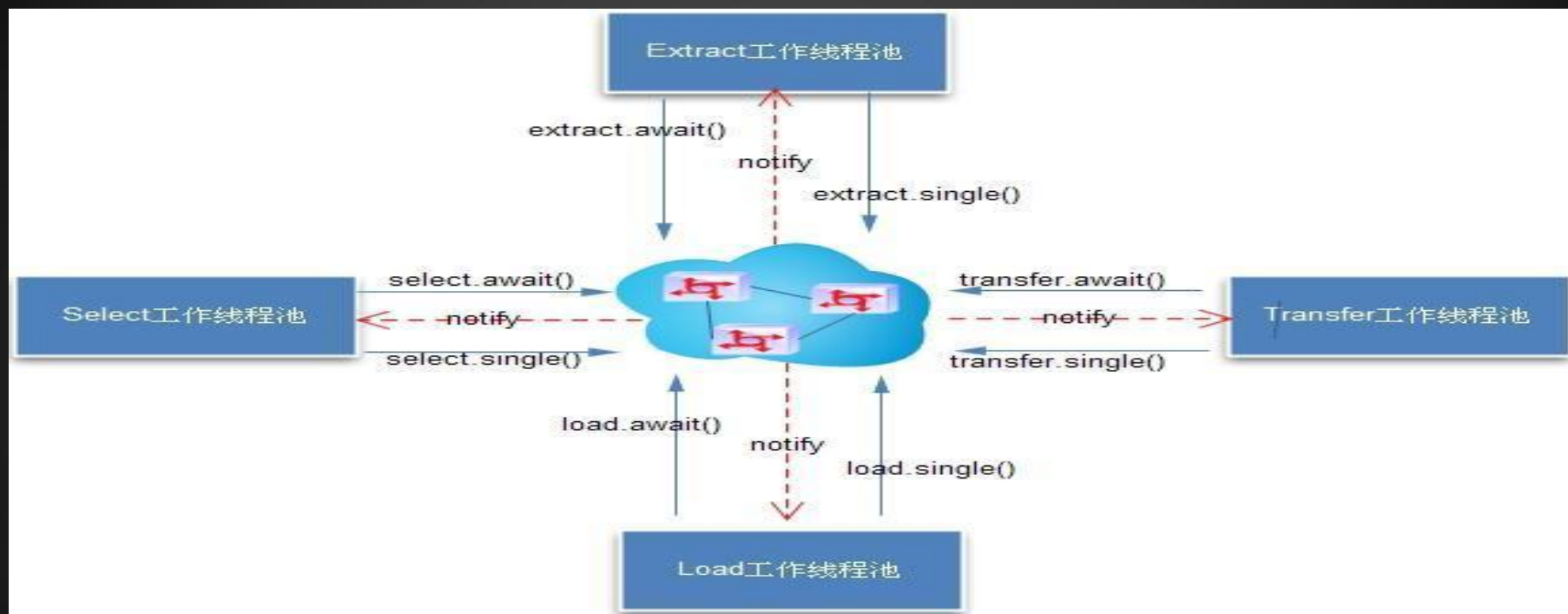
相关参数：

1. 并行度为5
2. 数据库Load为160ms
3. 网络延迟为200ms
4. S阶段binlog解析延迟 50ms
5. E阶段，纯数据同步近视为0ms，
6. T阶段和Termin，会有一次中美网络调用

Latency计算结果：

- a. 并行化 = $(S + E + T + Termin) + L * 5 = 1250ms$
- b. 串行化 = $(S + E + T + L + Termin) * 5 = 3050ms$

SEDA编程模型



“分布式锁”调度机制

- a. `await`模拟object获取锁操作
- b. `notify`被唤醒后提交任务到thread pools
- c. `single`模拟object释放锁操作，触发下一个stage

SEDA编程模型

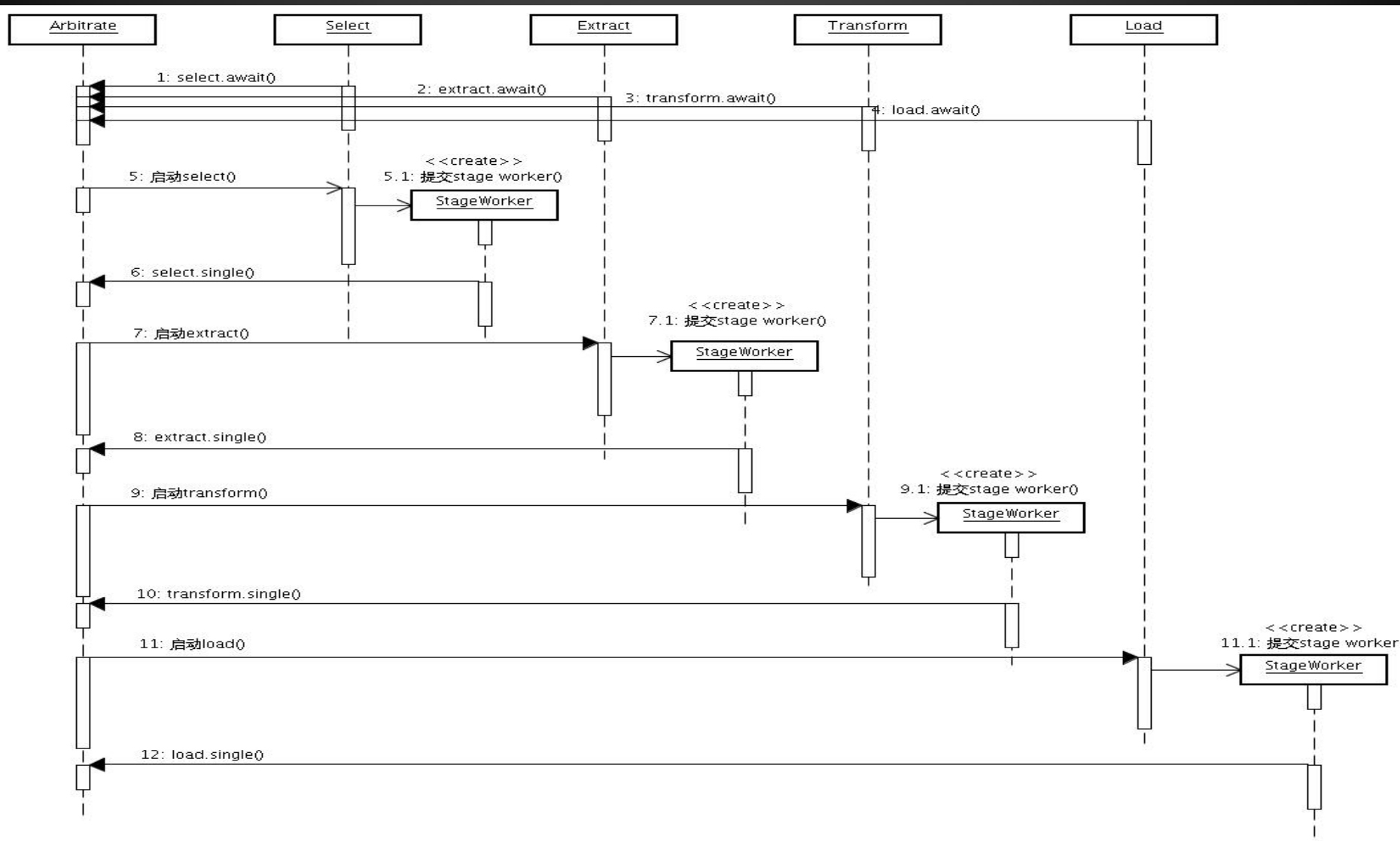
调度版本实现：

- in memory (单机版内存调度)
- rpc call (分布式基于rpc调用完成通知)
- zookeeper watcher (分布式基于watcher完成通知)

解决：

- a. 同机房调度 (单node节点 + in memory调度)
- b. 跨机房同步 (双node节点 + 分布式调度)

SEDA编程模型



SEDA编程模型

stage间数据传递: pipe

- stage | pipe | stage
- 最优实现调度

pipe实现:

- a. in memory
- b. rpc call
- c. file(gzip) + http多线程下载

SEDA编程模型

特殊性：

1. 变更数据可靠性 (保证数据不丢)

解决： 2pc

说明： select提供get/ack协议，一个S/E/T/L完成后ack

2. 调度可管理性 (支持启动/关闭/挂起运维)

解决： 2pc

说明： S/E/T/L接收rollback/commit，释放该批次资源

3. 创建映射规则

- a. 定义数据源
- b. 定义数据介质
- c. 建立映射规则
- d. 建议字段映射
- e. 建立字段组

3. 创建映射规则

a. 定义数据源

定义：

mysql: jdbc:mysql://10.20.144.15:3306/tddl_test_1

oracle : jdbc:oracle:thin:@10.20.144.29:1521:OINTEST

diamond : diamond://group?appKey=key1&groupKey=key2 (更改 key1 和 key2)

配置完成后，验证下数据库链接。

3. 创建映射规则

b. 定义数据介质

定义:

1. 单表模式 : alibaba1949.product
2. 多表模式 : sku[1-64].product_sku,havana_0000.member_[0000-0015]
3. 正则模式 : havana_*.member_.* (通配), *.*(全库)

注意:

- a. 表统计都是基于一个映射规则的定义, 多表/正则会进行合并统计.

3. 创建映射规则

C. 建立映射规则

操作:

1. 选择数据介质 (源和目标)
2. 定义weight (确定加载顺序)
 - a. weight数字越大代表越重要, 越靠后面执行.
比如product/product_detail, product的weight要大于product_detail
3. FileResolver类定义
 - a. 根据数据库记录, 拼装图片地址返回给otter. 比如将值按逗号分隔等
4. EventProcessor类定义
 - a. 数据过滤, 比如忽略status = 'new'状态的同步
 - b. 数据修改, 比如c字段的值 = a字段 + b字段

说明: 支持批量创建

3. 创建映射规则

b. 定义数据介质

定义:

1. 单表模式 : alibaba1949.product
2. 多表模式 : sku[1-64].product_sku,havana_0000.member_[0000-0015]
3. 正则模式 : havana_*.member_.* (通配), *.*(全库)

注意:

- a. 表统计都是基于一个映射规则的定义, 多表/正则会进行合并统计.

3. 创建映射规则

d. 建议字段映射

操作： 在创建映射规则，点击下一步，即进入字段映射的配置页面.

字段映射：

- a. 解决字段名字不同
- b. 解决字段需要同步/忽略
- c. 解决一对多字段

说明： 字段类型不同默认支持，需要业务保证类型精度

3. 创建映射规则

e. 建立字段组

操作： 在创建映射规则，点击下一步，即进入字段映射的配置页面.

字段组：

- a. 解决多个字段原子变更
- b. 解决FileResolver依赖多个字段构造图片地址

比如：

- 1. id,image_path,version三个字段决定一张图片
- 2. 有任何一个字段变更，保证三个都可见，否则保证三个不可见(构造不出图片URL)

注意： FileResolver需要对字段做null判读

4. Canal设置

参数解释:

a. 数据源类型

i. mysql: 需要指定slaveld, canal将自己伪装为mysql slave获取binlog

ii. oracle: 指定oracle erosa地址信息, 非oracle地址

iii. localbinlog: 指定本地目录的binlog进行消费

b. 位点定义 (不配置, 默认就是当前位置)

i. 文件位置:

```
{"journalName":"mysql.bin000001","position":106};
```

ii. 时间戳: {timestamp":1362591698000}; #13位精确到毫秒的时间戳

c. 存储机制

i. memory模式, 支持定义buffer size.

d. HA机制

i. heartbeat / tddl

e. 心跳配置

注意: canal name定义必须和pipeline定义保持一致

Canal子项目

提供异构数据源的接入(类eromanga解决方案)

canal特性：

- 可嵌入otter中部署，解析数据不落地
- 支持mysql/oracle源数据接入
- 解决mysql数据库主备切换
- 解决多库合并
- 流式api数据获取接口（贴合otter并行调度模型）

4. 增加监控项

参数解释：

a. 监控项目

1. 延迟：定义为数据从源库写入，到通过otter写出到目标库的时间差
2. 异常：s/e/t/l处理过程中抛出的异常，定义关键字进行接收异常
3. Process超时：定义为一次s/e/t/l的调度时间
4. Position超时：定义为最后一次位点更新时间

b. 阈值

1. 1800@09:00~21:00，按时间范围定义

c. 报警间隔 (重复报警会压制，有效利用短信资源)

d. 发送人KEY (对应于dragoon2.5中的KV监控名称)

e. 自动恢复

1. 针对出现网络异常可尝试自动恢复，减少人肉运维成本

监控设计

流程：

- a. 上帝之手(dragon), 定时触发monitorTrigger.htm
- b. monitorTrigger触发进行监控项检查
- c. 检查失败, 使用PassiveSender, 推送给dragon
- d. 如果自动恢复开启, 加入自动恢复队列, 重启同步

注意点：





- a. 上帝之手, 目前配置为5分钟触发一次
- b. 异常报警, node即时推送给manager, 即时报警

优势： 有效控制报警内容, 准确描述异常情况

使用&运维

如何了解一个otter同步情况

1. 同步状态

96	lj-channel-27	停止	单向	 查看	 启用	 编辑	 删除
92	lj-channel-25	运行	单向	 查看	 停用		
90	lj-oracle-oracle-02	挂起	单向	 查看	 停用	 解挂	 编辑  删除

几种状态：

- a. 停止
- b. 运行

c. 挂起 (标红显示，可以理解为暂定)

一种特殊的状态，hold住s/e/t/l调度，但不释放系统资源

2. 运行状态

Pipeline管理 channel管理 > Pipeline管理

序号	Pipeline名字	并行度	主站点	mainstem状态	延迟时间(秒)	最后同步时间	最后位点时间	监控数	操作
479	hz_us	5	false	工作中	1	2013-3-7 11:29:25	2013-3-7 11:29:43	4	查看 监控 日志
297	us_hz	5	true	工作中	2	2013-3-7 11:29:43	2013-3-7 11:29:42	4	查看 监控 日志

pipeline状态：

- 定位中 (标红显示, canal还未拿到第一条binlog数据)
- 工作中

延迟时间：

- 最后一批同步数据的从源库写入到otter同步到目标库的时间差

最后同步时间 vs 最后位点时间

- 同步时间, 代表当前数据库中映射关系表数据最后一次消费成功的时间
- 位点时间, 代表当前数据库中所有表数据最后一次消费成功的时间

同步时间 < 位点时间

3. 数据统计

几种纬度：

- a. 延迟时间
- b. 吞吐量 (实时 + 历史)
- c. SEDA调度状态
- d. insert/update/delete数据统计
- e. 文件数量/大小统计

映射关系列表

Channel管理 > Pipeline管理 > 映射关系列表

映射关系列表

吞吐量

延迟时间

同步进度

历史吞吐量

冲突字段

监控管理

日志记录

4. 日志查询

日志记录

Channel管理 > Pipeline管理 > 日志记录

映射关系列表

吞吐量

数据积压

同步进度

历史吞吐量

冲突字段

日志记录

SEARCH

请输入关键字(目前支持log内容关键字搜索)



序号	Channel信息	Pipeline信息	Node信息	日志标题	日志内容	发生时间
					关闭详细信息信息	
1927911	icbu_p4p_distdb[49-64]	hz_us	manager	PIPELINETIMEOUT	pid:89 elapsed 2470 seconds no sync	2012-10-7 3:51:19
1927907	icbu_p4p_distdb[49-64]	hz_us	manager	PIPELINETIMEOUT	点击查看详细信息	2012-10-7 3:46:19

几种纬度：
监控项 / pipeline

同步性能优化

1. 数据最小化

- a. 数据合并
- b. 数据压缩

2. 数据并行化

- a. S/E/T/L并行调度
- b. join并行化
- c. load并行化 (pk hash + weight)

数据合并算法

1. insert + insert -> insert (数据迁移+数据增量场景)
2. insert + update -> insert (update字段合并到insert)
3. insert + delete -> delete
4. update + insert -> insert (数据迁移+数据增量场景)
5. update + update -> update
6. update + delete -> delete
7. delete + insert -> insert
8. delete + update -> update (数据迁移+数据增量场景)
9. delete + delete -> delete

说明.

1. insert/行记录update 执行merge sql, 解决重复数据执行
2. 合并算法执行后, 单pk主键只有一条记录, 解决并行load的效率

load并行化

pk hash算法：

需求描述：提升同步性能，按table粒度并行时，改善大表同步问题

解决方案：根据table + pk hash后进行并行提交

优化方案：合并相同执行sql的pk hash结果，进行batch提交 (id排序，mysql顺序写，减少网络交互)

weight算法：（业务事务性支持）

业务需求：事务中顺序更新offer_detail，offer表，同步时插入保证顺序

解决方案：定义offer_detail(weight=1),offer(weight=2)，按权重从小到大插入，保证在一个批次数据中offer_detail的变更要优先于offer表变更插入

load并行化

pk hash + weight混合算法：

- a. 根据weight不同，构建多个weight bucket
- b. 按weight顺序，对每个weight bucket执行pk hash算法

pk hash + weight混合 + 多库复制：

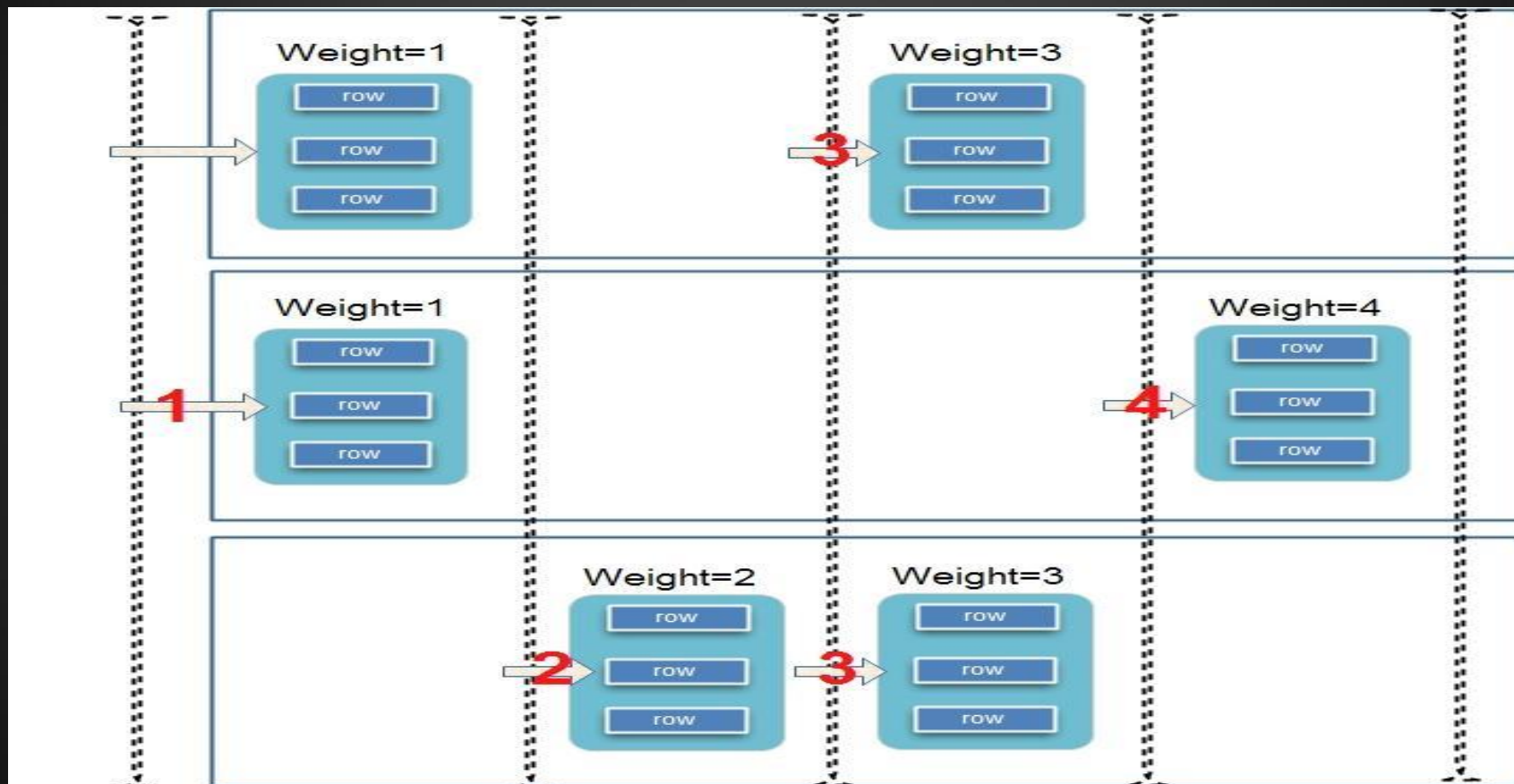
业务描述：

- a. 数据库load完成后，发送数据到mq，或者更新cache
- b. 会员变更数据，需要同步到多个目标数据库

算法描述：

- a. 每个库创建一份load实例，并接入weight controller调度
- b. 每个库按pk hash+weight混合算法进行调度，单库的weight bucket的调度由weight controllert的统一控制

load并行化



二维线程池weight调度:

纬度一: 多库载入, 纬度二: 单库pk hash

otter4.0支持同步功能内容

1. 同步映射

- a. 1 : 1 映射, (offer -> offer, 最简单业务)
- b. n : 1映射, (offer[1-32] -> offer)
- c. 1 : n 映射, (offer -> offer , offer_log) 数据多路复

制

2. 视图同步

- a. 表名不同 (ocndb.member -> crmg.cbu_member)
- b. 字段名不同 (member_id -> vaccount_id)
- c. 字段类型不同 (number(11,2) -> varchar(32))
- d. 字段个数不同 (1:n映射,1个字段复制到目标多个字

otter4.0支持同步功能内容

3. 数据join

- a. 根据变更column获取关联图片
- b. 根据pk join数据库整行记录 (dw业务)
- c. 根据变更字段，查询group字段进行同步 (字段组)
- d. 根据pk查询关联表记录 (规划中)

4. 数据filter

- a. 业务自定义扩展代码 (EventProcessor)

otter4.0支持同步功能内容

5. 同步一致性 & 同步模式

- a. 基于介质 + 行记录同步 (数据订正,数据迁移)
- b. 基于当前变更 + 字段模式 (正常同步业务)

Otter4使用约定

1. 同步表必须有主键
2. oracle表不允许使用blob/clob (mysql无此限制)
3. 数据订正 (几种case需要和otter团队沟通)
 - a. 纯数据订正超过1000w
 - b. 带文件订正超过50w
 - c. 非映射关系表订正超过5000w (otter4正在做优化, 尽早解除限制)
4. 新通道上线步骤 (当前)
 - a. 明确同步需求
 - i. 单向 / 双向 / 双写(需要明确主要写入站点) / 文件同步
 - b. 全量数据初始化
 - i. 行记录 + gmt_modified修改
 - ii. 插入同步记录到retl_buffer表

Otter4使用约定

5. 数据表字段变更

- a. 只允许新增字段到末尾（删除字段慎重）
- b. 字段新增先加目标库，再加源库
- c. 双向同步，新增字段建议无默认值（可确保同步无挂起）

6. 图片同步，需要先写图片，后插数据

otter4同步延迟比较低，如果先写数据，后写图片或者两者并发写，就会有一定的概率拿到数据后，反查没有图片，导致图片同步丢失

Otter常见FAQ

1. 同步隔离性

- a. otter pipeline按表级别定义同步映射，不同pipeline互不影响
- b. 接入erosa+canal，按库存储数据，不同表同步会存在一定影响

2. 同步延迟

取决目标数据库可接受的load并发度 + 地域之间的网络延迟

3. 核心竞争力

- a. 并行调度模型，(缓解extract/transform I/O latency问题)
- b. 双向同步 / 双A同步 (避免回环同步 / 冲突检测)
- c. pk hash + weight并行载入 (极大的提升同步性能)
- d. 接入canal，高效获取增量数据，并按变更字段同步 (高效,低latency)
- e. 同步映射 / 视图同步 / 数据join / 数据filter (强大的功能支持)

otter资源

1. otter manger

<http://otter.alibaba-inc.com>

2. 相关文档

<http://b2b-doc.alibaba-inc.com/display/RC/Otter>

<http://b2b-doc.alibaba-inc.com/display/opentech/otter>

3. 需求平台

<http://agile.alibaba-inc.com/browse/OTTER>

TKS!