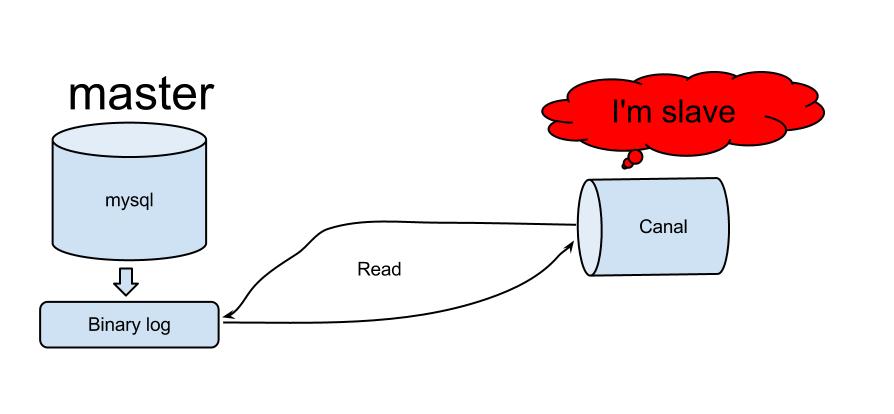
**准备环境：**

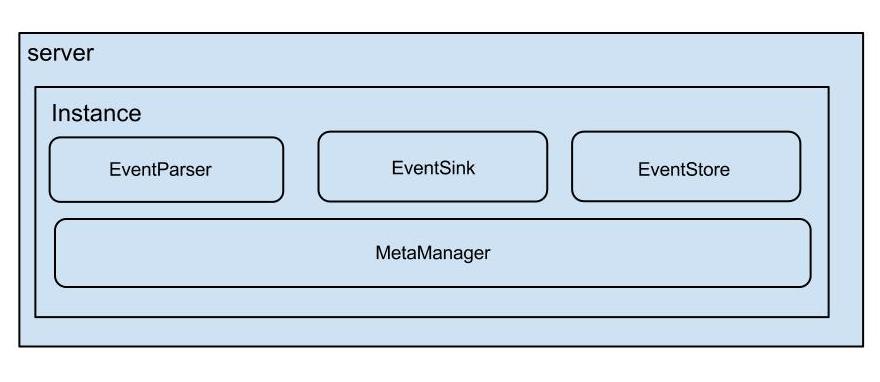
1. JDK1.8
2. canal.deployer-1.1.2，canal服务端
3. zookeeper-3.4.13，用于多个canal注册服务
4. mysql 5.6

**工作原理：**

****

1. canal模拟mysql slave的交互协议，伪装自己为mysql slave，向mysql master发送dump协议
2. mysql master收到dump请求，开始推送binary log给slave(也就是canal)
3. canal解析binary log对象(原始为com.google.protobuf.ByteString类型byte流)

**Canal内部架构：**



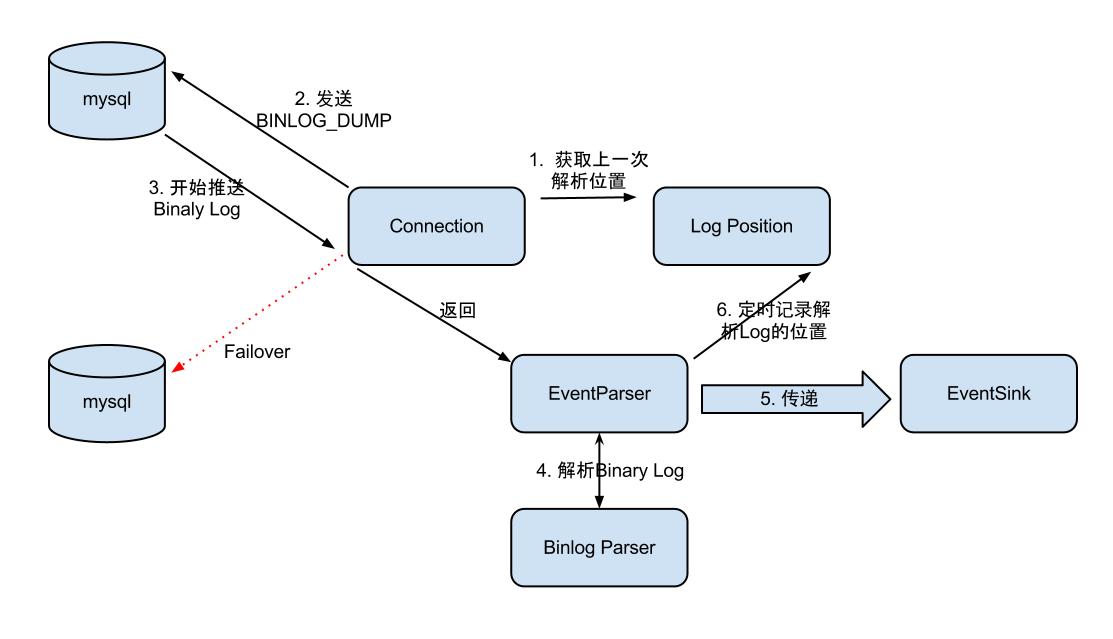
说明：

* server代表一个canal运行实例，对应于一个jvm
* 每一个数据库实例就是一个Instance
* instance对应于一个数据队列 （1个server对应1..n个instance)

instance模块：

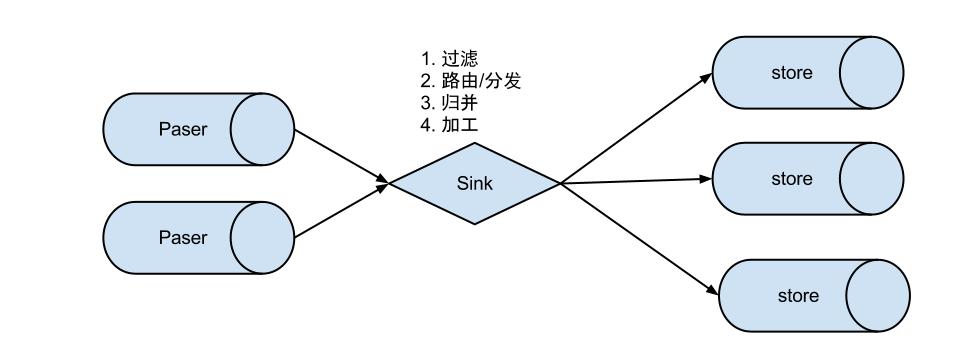
* eventParser (数据源接入，模拟slave协议和master进行交互，协议解析)
* eventSink (Parser和Store链接器，进行数据过滤，加工，分发的工作)
* eventStore (数据存储)
* metaManager (增量订阅&消费信息管理器)

**EventParser设计：**



1. Connection获取上一次解析成功的位置 (如果第一次启动，则获取初始指定的位置或者是当前数据库的binlog位点)
2. Connection建立链接，发送BINLOG\_DUMP指令  
   // 0. write command number  
   // 1. write 4 bytes bin-log position to start at  
   // 2. write 2 bytes bin-log flags  
   // 3. write 4 bytes server id of the slave  
   // 4. write bin-log file name
3. Mysql开始推送Binaly Log
4. 接收到的Binaly Log的通过Binlog parser进行协议解析，补充一些特定信息  
   // 补充字段名字，字段类型，主键信息，unsigned类型处理
5. 传递给EventSink模块进行数据存储，是一个阻塞操作，直到存储成功
6. 存储成功后，定时记录Binaly Log位置

EventSink设计



说明：

* 数据过滤：支持通配符的过滤模式，表名，字段内容等
* 数据路由/分发：解决1:n (1个parser对应多个store的模式)
* 数据归并：解决n:1 (多个parser对应1个store)
* 数据加工：在进入store之前进行额外的处理，比如join

**数据1:n业务**

为了合理的利用数据库资源， 一般常见的业务都是按照schema进行隔离，然后在mysql上层或者dao这一层面上，进行一个数据源路由，屏蔽数据库物理位置对开发的影响，阿里系主要是通过cobar/tddl来解决数据源路由问题。

所以，一般一个数据库实例上，会部署多个schema，每个schema会有由1个或者多个业务方关注

**数据n:1业务**

同样，当一个业务的数据规模达到一定的量级后，必然会涉及到水平拆分和垂直拆分的问题，针对这些拆分的数据需要处理时，就需要链接多个store进行处理，消费的位点就会变成多份，而且数据消费的进度无法得到尽可能有序的保证。

所以，在一定业务场景下，需要将拆分后的增量数据进行归并处理，比如按照时间戳/全局id进行排序归并.

**EventStore设计**

**Store模式:**

**memory-instance.xml介绍：**

所有的组件(parser , sink , store)都选择了内存版模式，记录位点的都选择了memory模式，重启后又会回到初始位点进行解析

特点：速度最快，依赖最少(不需要zookeeper)

场景：一般应用在quickstart，或者是出现问题后，进行数据分析的场景，不应该将其应用于生产环境

file-instance.xml介绍：

所有的组件(parser , sink , store)都选择了基于file持久化模式，注意，不支持HA机制.

特点：支持单机持久化

场景：生产环境，无HA需求，简单可用.

**default-instance.xml介绍：**

所有的组件(parser , sink , store)都选择了持久化模式，目前持久化的方式主要是写入zookeeper，保证数据集群共享.

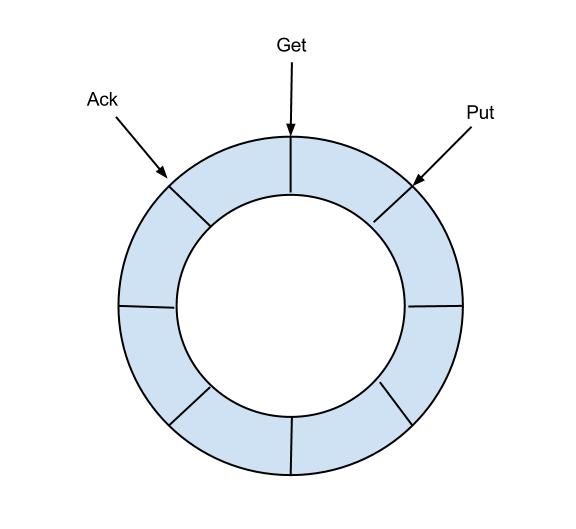
特点：支持HA

场景：生产环境，集群化部署.

**group-instance.xml介绍：**

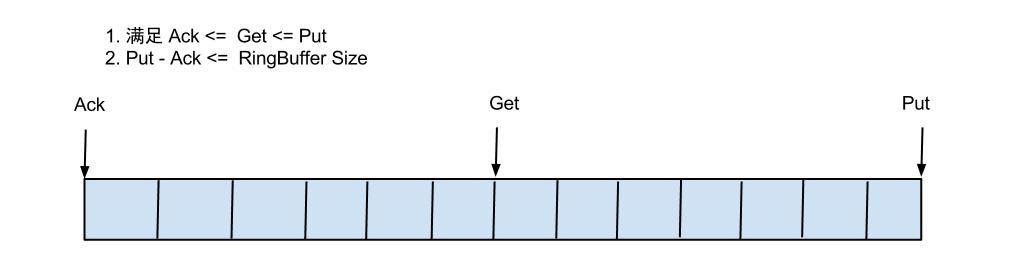
主要针对需要进行多库合并时，可以将多个物理instance合并为一个逻辑instance，提供客户端访问。

场景：分库业务。 比如产品数据拆分了4个库，每个库会有一个instance，如果不用group，业务上要消费数据时，需要启动4个客户端，分别链接4个instance实例。使用group后，可以在canal server上合并为一个逻辑instance，只需要启动1个客户端，链接这个逻辑instance即可.



定义了3个cursor

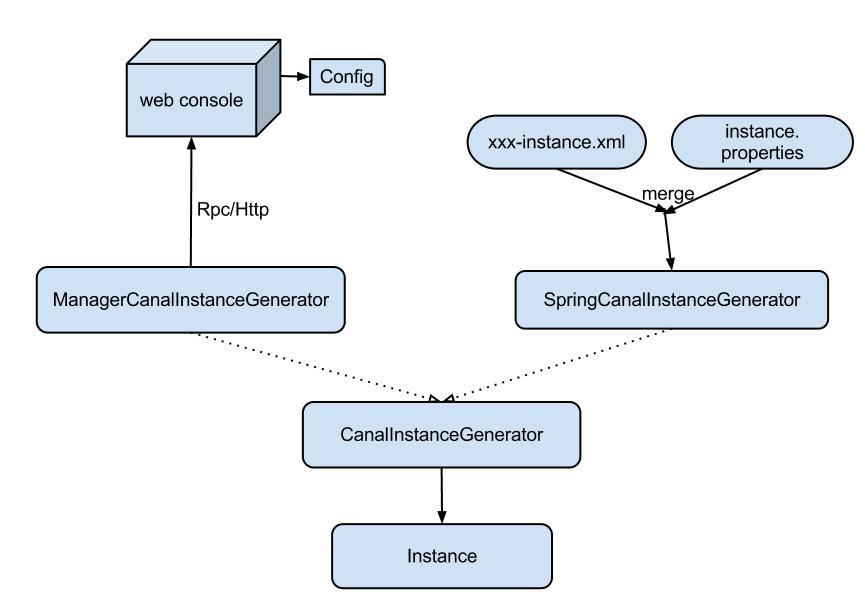
* Put : Sink模块进行数据存储的最后一次写入位置
* Get : 数据订阅获取的最后一次提取位置
* Ack : 数据消费成功的最后一次消费位置

借鉴Disruptor的RingBuffer的实现，将RingBuffer拉直来看：  


实现说明：

* Put/Get/Ack cursor用于递增，采用long型存储
* buffer的get操作，通过取余或者与操作。(与操作： cusor & (size - 1) , size需要为2的指数，效率比较高)

**Instance设计:**

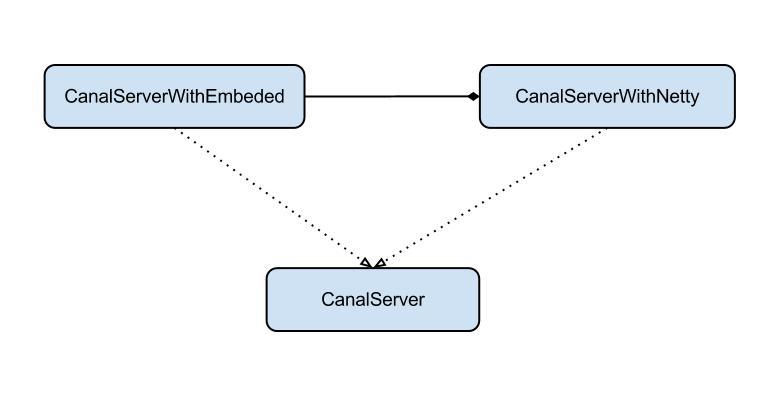


instance代表了一个实际运行的数据队列，包括了EventPaser,EventSink,EventStore等组件。

抽象了CanalInstanceGenerator，主要是考虑配置的管理方式：

* manager方式： 和你自己的内部web console/manager系统进行对接。(目前主要是公司内部使用)
* spring方式：基于spring xml + properties进行定义，构建spring配置.

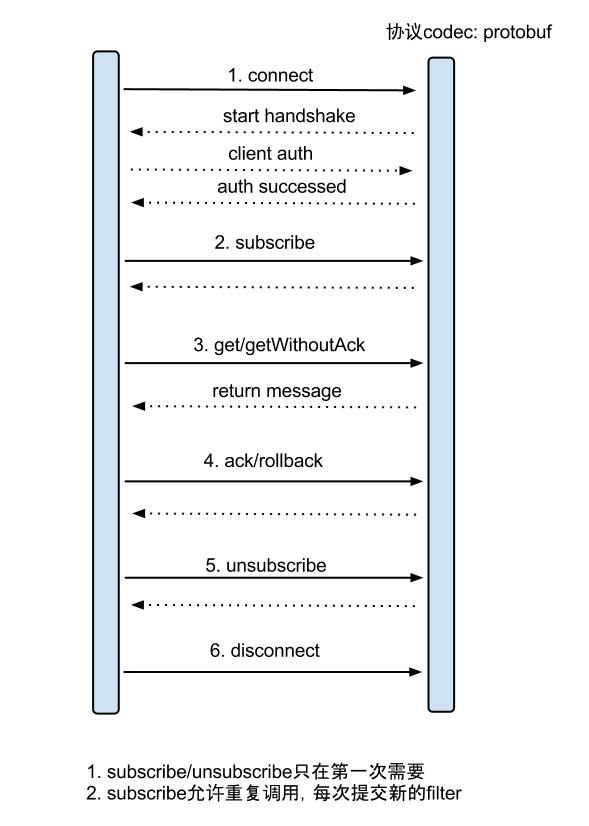
**Server设计:**



server代表了一个canal的运行实例，为了方便组件化使用，特意抽象了Embeded(嵌入式) / Netty(网络访问)的两种实现

* Embeded : 对latency和可用性都有比较高的要求，自己又能hold住分布式的相关技术(比如failover)
* Netty : 基于netty封装了一层网络协议，由canal server保证其可用性，采用的pull模型，当然latency会稍微打点折扣，不过这个也视情况而定。(阿里系的notify和metaq，典型的push/pull模型，目前也逐步的在向pull模型靠拢，push在数据量大的时候会有一些问题)

**增量订阅/消费设计：**



具体的协议格式，可参见：[CanalProtocol.proto](https://github.com/alibaba/canal/blob/master/protocol/src/main/java/com/alibaba/otter/canal/protocol/CanalProtocol.proto)

get/ack/rollback协议介绍：

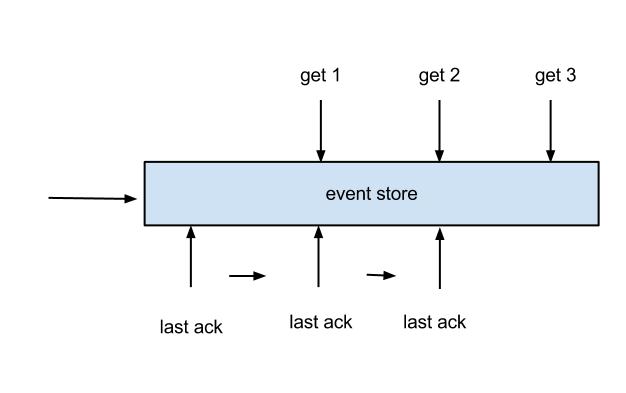
* Message getWithoutAck(int batchSize)，允许指定batchSize，一次可以获取多条，每次返回的对象为Message，包含的内容为：  
  a. batch id 唯一标识  
  b. entries 具体的数据对象，对应的数据对象格式：[EntryProtocol.proto](https://github.com/alibaba/canal/blob/master/protocol/src/main/java/com/alibaba/otter/canal/protocol/EntryProtocol.proto)
* void rollback(long batchId)，顾命思议，回滚上次的get请求，重新获取数据。基于get获取的batchId进行提交，避免误操作
* void ack(long batchId)，顾命思议，确认已经消费成功，通知server删除数据。基于get获取的batchId进行提交，避免误操作

canal的get/ack/rollback协议和常规的jms协议有所不同，允许get/ack异步处理，比如可以连续调用get多次，后续异步按顺序提交ack/rollback，项目中称之为流式api.

流式api设计的好处：

* get/ack异步化，减少因ack带来的网络延迟和操作成本 (99%的状态都是处于正常状态，异常的rollback属于个别情况，没必要为个别的case牺牲整个性能)
* get获取数据后，业务消费存在瓶颈或者需要多进程/多线程消费时，可以不停的轮询get数据，不停的往后发送任务，提高并行化. (作者在实际业务中的一个case：业务数据消费需要跨中美网络，所以一次操作基本在200ms以上，为了减少延迟，所以需要实施并行化)

流式api设计：



* 每次get操作都会在meta中产生一个mark，mark标记会递增，保证运行过程中mark的唯一性
* 每次的get操作，都会在上一次的mark操作记录的cursor继续往后取，如果mark不存在，则在last ack cursor继续往后取
* 进行ack时，需要按照mark的顺序进行数序ack，不能跳跃ack. ack会删除当前的mark标记，并将对应的mark位置更新为last ack cusor
* 一旦出现异常情况，客户端可发起rollback情况，重新置位：删除所有的mark, 清理get请求位置，下次请求会从last ack cursor继续往后取

### 数据对象格式：EntryProtocol.proto

Entry

Header

logfileName [binlog文件名]

logfileOffset [binlog position]

executeTime [binlog里记录变更发生的时间戳]

schemaName [数据库实例]

tableName [表名]

eventType [insert/update/delete类型]

entryType [事务头BEGIN/事务尾END/数据ROWDATA]

storeValue [byte数据,可展开，对应的类型为RowChange]

RowChange

isDdl [是否是ddl变更操作，比如create table/drop table]

sql [具体的ddl sql]

rowDatas [具体insert/update/delete的变更数据，可为多条，1个binlog event事件可对应多条变更，比如批处理]

beforeColumns [Column类型的数组]

afterColumns [Column类型的数组]

Column

index [column序号]

sqlType [jdbc type]

name [column name]

isKey [是否为主键]

updated [是否发生过变更]

isNull [值是否为null]

value [具体的内容，注意为文本]

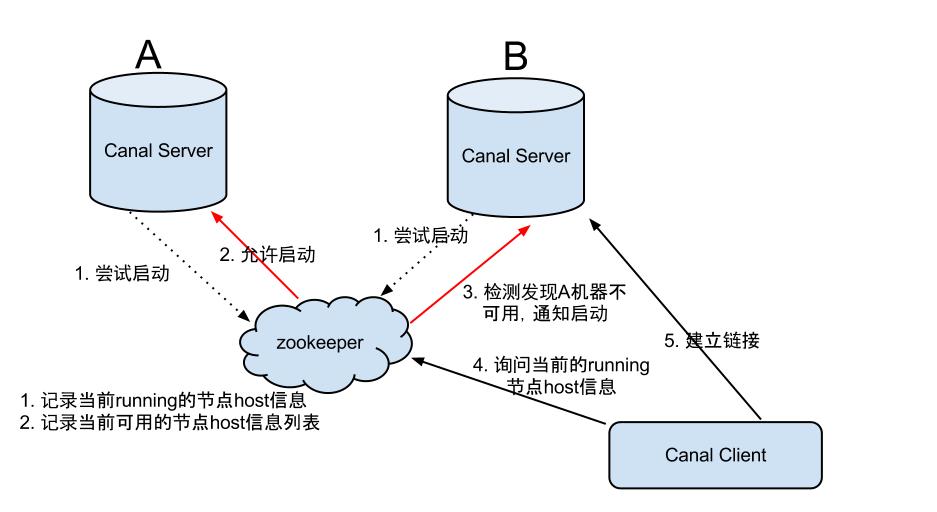
**HA机制设计:**

canal的ha分为两部分，canal server和canal client分别有对应的ha实现

* canal server: 为了减少对mysql dump的请求，不同server上的instance要求同一时间只能有一个处于running，其他的处于standby状态.
* canal client: 为了保证有序性，一份instance同一时间只能由一个canal client进行get/ack/rollback操作，否则客户端接收无法保证有序。

整个HA机制的控制主要是依赖了zookeeper的几个特性，watcher和EPHEMERAL节点(和session生命周期绑定)，可以看下我之前zookeeper的相关文章。

Canal Server:



大致步骤：

1. canal server要启动某个canal instance时都先向zookeeper进行一次尝试启动判断 (实现：创建EPHEMERAL节点，谁创建成功就允许谁启动)
2. 创建zookeeper节点成功后，对应的canal server就启动对应的canal instance，没有创建成功的canal instance就会处于standby状态
3. 一旦zookeeper发现canal server A创建的节点消失后，立即通知其他的canal server再次进行步骤1的操作，重新选出一个canal server启动instance.
4. canal client每次进行connect时，会首先向zookeeper询问当前是谁启动了canal instance，然后和其建立链接，一旦链接不可用，会重新尝试connect.

Canal Client的方式和canal server方式类似，也是利用zookeeper的抢占EPHEMERAL节点的方式进行控制.

**配置说明：**

1. Mysql：

Windows：修改my.ini文件

Linux:修改my.cnf文件

#开启log\_bin，指定log\_bin所在目录，如果目录不存在需要手动创建，否则mysql无法正常启动

log-bin=/usr/local/mysql/binlog/mysql-bin

#选择ROW模式，可以记录修改前后的数据，在 STATEMENT 或 MIXED 模式下, Binlog 只会记录和传输 SQL 语句（以减少日志大小），而不包含具体数据

binlog-format=ROW

#指定serverid，因为canal模拟slave，不能与canal的serverid重复

server\_id = 100

创建MYSQL salve用户

CREATE USER canal IDENTIFIED BY 'canal';

GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'canal'@'%';

FLUSH PRIVILEGES;

2） canal.deployer

修改canal.deployer-1.1.2\conf中canal.properties

canal.zkServers = 127.0.0.1:2181

#数据库instance name，多个以逗号分隔

#在conf下创建相同文件名的文件夹，并在文件夹下创建instance.properties配置文件

canal.destinations = example

#参考EventStore设计中的说明

canal.instance.global.spring.xml = classpath:spring/default-instance.xml

如需修改其他配置参考<https://github.com/alibaba/canal/wiki/AdminGuide>

修改canal.deployer-1.1.2\conf\examplez中instance.properties配置文件

#slaveId不能与mysql中的server\_id重复

canal.instance.mysql.slaveId=2000

#数据库实例地址端口

canal.instance.master.address=192.168.43.242:3306

#数据库slave用户密码

canal.instance.dbUsername=canal

canal.instance.dbPassword=canal

#默认数据库名称

canal.instance.defaultDatabaseName =test

#备份库表过滤器，可在客户端中重写

#1) ".\\..\*" : 所有库表，

#2)"test\\..\*" : TEST库下所有库表，

#3)"test\\.tab.\*" : TEST库下tab开头库表，

#4)"test\\.test1" : TEST下test1表，

#5)多组合逗号分隔

canal.instance.filter.regex=.\*\\..\*

**Canal客户端：**

* 1. Maven依赖



* 1. 创建基于zookeeper的客户端



客户端程序中涉及到的数据类型

**EntryType:**

***TRANSACTIONBEGIN, ROWDATA, TRANSACTIONEND***

**EventType**

***QUERY ,INSERT ,UPDATE, DELETE, CREATE, ALTER, TRUNCATE ,***

***RENAME, CINDEX, DINDEX, GTID, XACOMMIT, XAROLLBACK,***

***MHEARTBEAT***

Entry数据内容：

***TRANSACTIONBEGIN:***

header {

version: 1

logfileName: "mysql-bin.000015"

logfileOffset: 1723

serverId: 100

serverenCode: "UTF-8"

executeTime: 1548426631000

sourceType: MYSQL

schemaName: ""

tableName: ""

eventLength: 72

}

entryType: TRANSACTIONBEGIN

storeValue: " $"

***ROWDATA:***

UPDATE:

**Entry UPDATE SQL**

**rowChange**



**Entry UPDATE数据：StoryValue为byte流数据**



**RowChange：**



Update可以获取到更新之前和更新之后的数据：

RowData rowData = rowChage.getRowDatasList().get(0);

List<Column> beColumns = rowData.getBeforeColumnsList()

List<Column> afColumns = rowData.getAfterColumnsList()

rowData中重要参数：name(字段名)，isKey(是否主键)，updated(是否更新)

**INSERT:SQL**

**RowChange**

 **Entry 数据**

**RowChange**



DELETE:



rowChange:

eventType: QUERY

sql: "DELETE from tab\_user where userid=\'7\'"

Entry





***TRANSACTIONEND:***



**MYSQL binlog查询命令**

A.查询第一个(最早)的binlog日志：

mysql> show binlog events\G;

B.指定查询 mysql-bin.000021 这个文件：

mysql> show binlog events in 'mysql-bin.000021'\G;

C.指定查询 mysql-bin.000021 这个文件，从pos点:8224开始查起：

mysql> show binlog events in 'mysql-bin.000021' from 8224\G;

D.指定查询 mysql-bin.000021 这个文件，从pos点:8224开始查起，查询10条

mysql> show binlog events in 'mysql-bin.000021' from 8224 limit 10\G;

E.指定查询 mysql-bin.000021 这个文件，从pos点:8224开始查起，偏移2行，查询10条

mysql> show binlog events in 'mysql-bin.000021' from 8224 limit 2,10\G;

**Canal adapter适配器：**