**OceanBase多集群事件通知机制**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
| **1** | 0.1 |  | 新建文档 | 2012/12/20 | 晓楚 |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |
| **6** |  |  |  |  |  |

目录

[需求 1](#_Toc343848909)

[设计 2](#_Toc343848910)

[实现 2](#_Toc343848911)

[\_\_trigger\_event表 2](#_Toc343848912)

[\_\_trigger\_event行数据到 Trigger事件的转化 3](#_Toc343848913)

[Trigger事件的处理 3](#_Toc343848914)

[特殊考虑 3](#_Toc343848915)

[数据结构 3](#_Toc343848916)

[\_\_trigger\_event schema 3](#_Toc343848917)

[ObTriggerMsg 4](#_Toc343848918)

[ObRsTriggerEvent 4](#_Toc343848919)

[Trigger应用 5](#_Toc343848920)

# 需求

多集群中，存在这样一种需求：主集群希望将一个事件广播到所有备集群，使得备集群也能感知到事件的发生。

例如create table sample (pk int); 操作，主集群创建好表sample表后会刷新schema，并将新的schema通知给本集群的全部server，只有这样create好的表格才能被集群中的server查询到。但是备集群并不知道创建表格事件，因此在备集群上执行select \* from sample;会报告sample表不存在。如果有一种广播机制，能够将创建表格的事件通知给所有备集群，备集群就能够去主动的获取并刷新schema。

# 设计

多集群中，主备集群主要通过日志建立联系。当主集群希望广播一个事件的时候，它可以写一条特殊的事件日志。备集群的主UpdateServer(UPS)重放日志的过程中会检查日志类型，当发现日志为事件日志时则说明主集群通知了一个事件，它将该事件转发给备集群的RootServer(RS)，由RootServer决定采取什么样的相应。如下图所示：

上图中，有两点需要注意：

1. 事件通知的源不限于主集群，备集群也可以发出事件通知。
2. 主主UPS也可以向自己的RS发出事件，处理方式与备集群一致（图中没有画出来）

# 实现

## \_\_trigger\_event表

系统中加入一个内部表\_\_trigger\_event，一个事件通知就是用REPLACE语句向\_\_trigger\_event表中插入一条数据。

## \_\_trigger\_event行数据到 Trigger事件的转化

新插入\_\_trigger\_event的一条数据会以ObUpsMutator的形式发送给主主UPS。主主UPS会通过日志同步的方式将这个Mutator发送给主备UPS、各个备集群的备主UPS，备主UPS也会通过日志同步的方式将这个Mutator发送给自己的备备UPS。仅仅两种UPS关注这个Mutator：主主UPS、备主UPS，其余UPS都忽略该Mutator。

主主UPS在commit数据、备主UPS在回放数据阶段会检查每一个Mutator是否是Trigger产生的Mutator。为了尽可能降低查看Mutator的开销，UPS每遇到一个Mutator都会先只检索该Mutator的第一个cell，如果该cell的table id是OB\_TRIGGER\_EVENT\_TABLE\_TID则相当于收到一个Trigger事件。

## Trigger事件的处理

这个Trigger事件用TriggerMsg来描述，UPS将这个TriggerMsg转发给自己的RootServer，RootServer根据TriggerMsg.Type来决定执行何种操作。

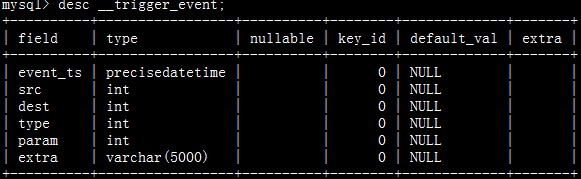
ObRootWorker::rt\_handle\_trigger\_event()中有一个大的switch case，用于根据TriggerMsg.Type采取操作。

## 特殊考虑

为了确保主集群bootstrap成功后就能够使用Trigger机制，\_\_trigger\_event表需要在bootstrap阶段就建立成功，包括建立空tablet。

# 数据结构

## \_\_trigger\_event schema



## ObTriggerMsg

ObTriggerMsg描述了一个Trigger的源、目的、类型和参数，目前关注的只有类型，其余几个参数还没有充分利用起来，有待进一步定义。

class ObTriggerMsg{

public:

ObTriggerMsg(){}

~ObTriggerMsg(){}

NEED\_SERIALIZE\_AND\_DESERIALIZE;

public:

int64\_t src;

int64\_t dest;

int64\_t type;

int64\_t param;

};

## ObRsTriggerEvent

这个类用于触发一个Trigger事件，实质就是向\_\_trigger\_event表中插入一条数据。当新建一个表时，调用ObRsTriggerEvent::create\_table()就可以将新建表的消息通知给所有集群。

class ObRsTriggerEvent : public ObTriggerEvent{

public:

ObRsTriggerEvent()

{

}

~ObRsTriggerEvent(){}

int create\_table();

int drop\_tables();

int notify\_slave\_refresh\_schema();

/\* 在这里添加更多的事件 \*/

private:

DISALLOW\_COPY\_AND\_ASSIGN(ObRsTriggerEvent);

int refresh\_new\_schema();

};

# Trigger应用

1. 利用Trigger来“Bootstrap”备集群

备集群不进行bootstrap，但是它需要一个机会来构建备集群内部表schema。为了解决这个问题，可以在主集群bootstrap完成后触发一个事件，备集群收到这个事件后就开始构建核心表的schema和空tablet，并写data/rs/ first\_tablet\_meta文件（用于切换成主的时候识别自己是否已经bootstrap过）

【END】