Cedar 0.2 用户使用手册与操作指南

> 本手册旨在向Cedar的使用人员或者对Cedar感兴趣的数据库爱好者，介绍如何使用Cedar系统，包括如何安装、部署、启动集群，如何执行SQL操作，介绍系统码、配置项等。
>
> 如果想要了解Cedar每一个版本的新增功能以及具体细节，请参考Cedar 0.2的版本说明文档。如果想要了解OceanBase的开源版本，请参考OceanBase相应的参考手册。

注：本手册参考OceanBase 0.4的相关开源文档。

Cedar 0.2 系统简介

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 系统简介	郭进伟	

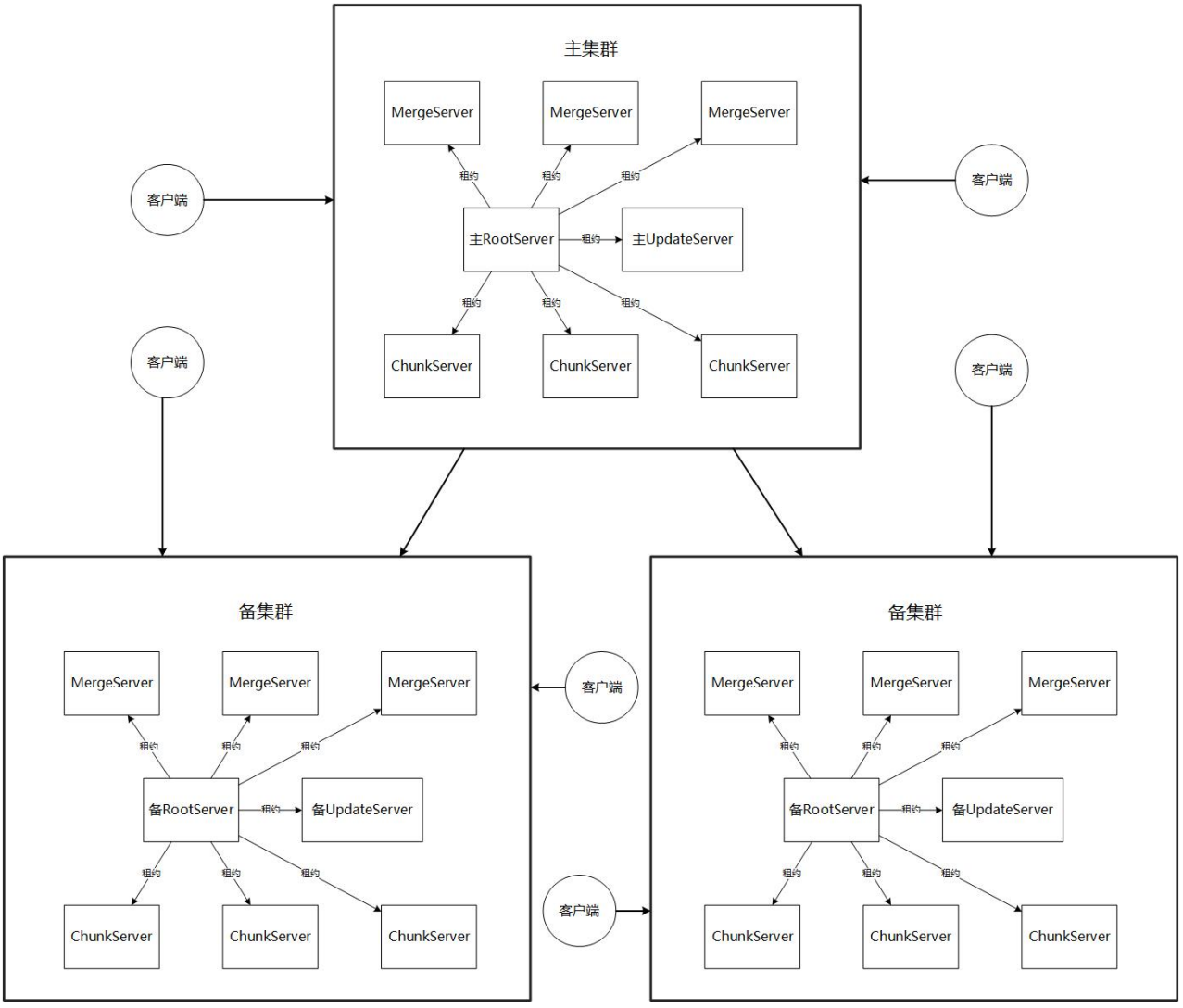

图1 Cedar系统架构图

Cedar是由华东师范大学数据科学与工程研究院基于OceanBase 0.4.2 研发的可扩展的关系数据库。设计目标是支持数百TB的海量数据，支持数十万TPS、数百万QPS的访问量，并且支持跨行跨表事务。

Cedar系统中有4个重要的模块，分别是主控服务器RootServer、更新服务器UpdateServer、基准数据服务器ChunkServer以及合并服务器MergeServer。RootServer用于管理整个集群中的所有Server，协调各个Server的工作，UpdateServer管理增量数据，ChunkServer管理基准数据，MergeServer接收客户端的操作请求，将基准数据和增量数据进行融合。

Cedar可部署为多集群架构，一个主集群，多个备集群，如图1所示，每个集群内包含一台主控服务器RootServer、一台更新服务器UpdateServer、多台基准数据服务器ChunkServer以及合并服务器MergeServer。

Cedar有如下几点重要的特性：

基线数据和增量数据分离的存储架构

Cedar系统内部按照时间线将数据划分为基准数据和增量数据，基准数据是只读的，所有的修改更新到增量数据中，系统内部通过合并操作定期将增量数据融合到基准数据中。采取这样的架构，使得更新操作的执行更加高效。

高可用的多集群主备架构，支持自动的主集群选举、主备集群切换机制

采取事务日志强同步&强一致的机制，当主集群出现故障宕机时，多个备集群可以迅速、自动地选出唯一的、新的主集群，使系统能够持续地提供对外服务。

功能丰富、方便易用

Cedar系统除了具备传统关系型数据库的基本功能，例如：读写事务、存储过程、游标（存储过程中该功能暂不可用）等，还新增了二级索引、非主键更新、BloomFilter Join、SemiJoin等特色功能，支持MySQL客户端、Java客户端以及C客户端，支持快照隔离级别、表锁机制，并对存储过程、事务提交流程、日志同步机制进行了优化，每个功能模块的设计思路和设计细节可参考Cedar 0.1 和Cedar 0.2 的开发文档。

联系我们

研发团队：华东师范大学数据科学与工程研究院Cedar项目组。

地址：上海市普陀区中山北路3663号。

邮政编码：200062。

对于文档内容有任何疑问，可通过发送邮件至cedar.tp@gmail.com，与我们联系，我们会在第一时间给予您反馈。

集群部署、启动

Cedar 0.2 安装环境配置指南

修订历史

版本修订	日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 安装环境配置指南	王嘉豪	

文档说明

本文档主要介绍如何在一台新服务器上配置成Cedar 0.2 可以运行的环境。

1 准备工作

需要确保服务器已经满足以下要求：

（1）服务器安装了CentOS 6.5 版本的系统。

（2）服务器网络正常，配置好了CentOSBase和epel软件源。

可以使用CentOS阿里云软件源，配置过程如下：

```
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.co
m/repo/Centos-6.repo

wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/e
pel-6.repo

yum makecache
```

（3）防火墙关闭。

可通过以下命令关闭防火墙：

```
service iptables stop
chkconfig iptables off
```

（4）服务器NTP时钟配置正常。

2 安装依赖的软件包

2.1 依赖的软件包简介

软件包	说明
liblzo2	是一个压缩库，Cedar需要用它来压缩静态数据，所需版本liblzo2.06。
Snappy	是 Google 出品的压缩库，Cedar使用 Snappy 压缩静态数据，所需版本Snappy1.0.3。
工具组	libtoolize 2.2.6，autoconf 2.66，automake 1.11.1。
libnuma	Cedar中使用了 NUMA，因此需要 libnuma 支持。
libaio	Cedar中用到了 AIO，需要 libaio 的支持。下面通过安装 libaio 来添加numa 相关的头文件和库。
gtest 和 gmock	非必须安装，可选。
其他	在编译Cedar时，还需要使用“libcurl-devel”、“openssl-devel”、“readline-devel”、“ncurses-devel 和“mysql-devel”。
tbsys 和 tbnet	tbsys 主要对操作系统服务进行封装，tbnet 主要提供网络框架。Cedar 依赖于这两个库。
libonev	libonev 是 Cedar中新的网络通讯框架。
mysql-dev	mysql开发库。

2.2 安装依赖包

在CentOS 6.5 系统，可以直接使用yum安装，操作命令如下：

```
yum install lzo-devel snappy-devel numactl-devel libaio-devel openssl-devel readline-devel
ncurses-devel mysql-devel libnone-devel lua-devel
rpm -ivh t-db-congo-drcmessage-0.1.1-26.el6.x86_64.rpm
rpm -ivh libonev-0.1-1.x86_64.rpm
```

2.3 用户环境配置

在用户编译Cedar时，需要在shell环境里配置如下几个变量：

```
#指定tb-common-utils库的目录
export TBLIB_ROOT=~/tb-common-utils

export LD_LIBRARY_PATH=$:/usr:/usr/lib:/usr/local/lib:/lib:$TBLIB_ROOT/lib:/usr/local/lib64
export ONEV_ROOT=/usr/lib/libonev
export ONEV_LIB_PATH=$ONEV_ROOT/lib
export DRC_ROOT=/home/ds
```

注：完成上述操作后，安装、启动Cedar系统的准备工作就完成了，安装、启动Cedar的相关操作可参考文档《Cedar 0.2 安装启动指南》。

Cedar 0.2 安装启动指南

修订历史

版本修订	日期	修订描述	作者	备注
Cedar 0.2	20160801	Cedar 0.2 安装启动指南	李宇明	

文档说明

本文档主要介绍如何在多台已经配置好环境的服务器上启动Cedar系统的流程（若服务器环境还未配置，可参考《Cedar 0.2安装环境配置指南》）。

注：本文档的内容参考自《OceanBase 0.4.2 安装指南》。

1 创建数据磁盘挂载点

数据磁盘用于存放UpdateServer和ChunkServer的数据。如果您挂载磁盘，那么UpdateServer和ChunkServer的数据将存放到挂载的磁盘中，否则，将存放在挂载点中。
创建UpdateServer和ChunkServer数据磁盘挂载点的操作步骤如下：

1. 以admin用户分别登录UpdateServer和ChunkServer所在的Cedar服务器。
2. 执行以下命令，创建磁盘挂载目录。

```
sudo mkdir /data
```

3. 执行以下命令，将“/data”目录赋给“admin”用户。

```
sudo chown admin /data
```

4. 根据磁盘规划和服务器规划创建挂载点。

> 说明：如果不能采用“for”语句创建，您可以根据规划在相应的服务器中直接使用mkdir命令逐个创建。

UpdateServer：

```
for disk in {1..8}; do mkdir -p /data/$disk/ups_data; done;
```

ChunkServer：

```
for disk in {1..8}; do mkdir -p /data/$disk; done;
```

2 创建各Server所需的目录

启动RootServer、UpdateServer和ChunkServer需要创建文件存放目录。

1. 以admin用户登录RootServer和UpdateServer所在的Cedar服务器。
2. 执行以下命令，创建数据存放目录。

```
mkdir -p /home/admin/cedar/data
```

3. 执行以下命令，创建RootServer所需目录。

```
mkdir -p /home/admin/cedar/data/rs
mkdir -p /home/admin/cedar/data/rs_commitlog
```

4. 执行以下命令，创建UpdateServer所需目录。

```
mkdir -p /home/admin/cedar/data/ups_commitlog
mkdir -p /home/admin/cedar/data/ups_data/raid0
mkdir -p /home/admin/cedar/data/ups_data/raid1
mkdir -p /home/admin/cedar/data/ups_data/raid2
mkdir -p /home/admin/cedar/data/ups_data/raid3
```

5. 执行以下命令，建立UpdateServer与数据存放磁盘的软连接。

```
ln -s /data/1/ups_data
/home/admin/cedar/data/ups_data/raid0/store0
ln -s /data/2/ups_data
/home/admin/cedar/data/ups_data/raid0/store1
ln -s /data/3/ups_data
/home/admin/cedar/data/ups_data/raid1/store0
ln -s /data/4/ups_data
/home/admin/cedar/data/ups_data/raid1/store1
ln -s /data/5/ups_data
/home/admin/cedar/data/ups_data/raid2/store0
ln -s /data/6/ups_data
/home/admin/cedar/data/ups_data/raid2/store1
ln -s /data/7/ups_data
/home/admin/cedar/data/ups_data/raid3/store0
```

```
ln -s /data/8/ups_data
/home/admin/cedar/data/ups_data/raid3/store1
```

6. 以admin用户登录ChunkServer所在的Cedar服务器。
7. 执行以下命令，创建数据存放目录。

```
mkdir -p /home/admin/cedar/data
```

8. 在ChunkServer挂载的磁盘中创建sstable存放的目录“obtest/sstable”。
   注意：“obtest”与APP名称相同。

```
for disk in {1..8}; do mkdir -p /data/$disk/obtest/sstable; done;
```

9. 执行以下命令，建立ChunkServer与数据存放磁盘的软连接。

```
for disk in {1..8};
do ln -s /data/$disk /home/admin/cedar/data/$disk; done;
```

3 安装Cedar

若Cedar源码已经完成编译、安装流程，可跳过此节。

1. 进入Cedar源代码所在目录。
2. 执行以下命令，初始化安装。

```
sh build.sh init
```

3. 执行以下命令，指定安装目录“/home/admin/cedar”。

```
./configure --prefix=/home/admin/cedar --with-release=yes --with-test-case=no;
```

4. 依次执行以下命令，编译安装程序。

```
make -j 10 -C src/
make -j 10 -C tools/
```

5. 执行以下命令，安装Cedar。

```
make install
```

4 启动各Server

启动RootServer、UpdateServer、ChunkServer和MergeServer的方法如下：

1. 以admin用户分别登录各Cedar服务器。
2. 执行以下命令，进入Cedar安装目录。

```
cd /home/admin/cedar
```

3. 启动RootServer、UpdateServer、ChunkServer和MergeServer。

> 启动ChunkServer前请先启动RootServer，否则ChunkServer在一段时间后会自动结束进程。
>
> 依次在“10.11.1.209”、“10.11.1.210”、“10.11.1.211”中，启动RootServer、UpdateServer、ChunkServer、MergeServer (说明：obtest为“2 创建各Server所需目录”中创建sstable所在的目录。)

```
bin/rootserver -r 10.11.1.209:2500 -i eth0 -s 10.11.1.209:2500@1#10.11.1.210:2500@2#10.1
1.1.211:2500@3 -C 1
bin/updateserver -r 10.11.1.209:2500 -p 2700 -m 2701 -i eth0
bin/mergeserver -r 10.11.1.209:2500 -p 2800 -z 2828 -i eth0 -t 1ms
bin/chunkserver -r 10.11.1.209:2500 -p 2600 -n obtest -i eth0
bin/mergeserver -r 10.11.1.209:2500 -p 2800 -z 2880 -i eth0
```

```
bin/rootserver -r 10.11.1.210:2500 -i eth0 -s 10.11.1.209:2500@1#10.11.1.210:2500@2#10.1
1.1.211:2500@3 -C 2
bin/updateserver -r 10.11.1.210:2500 -p 2700 -m 2701 -i eth0
bin/mergeserver -r 10.11.1.210:2500 -p 2800 -z 2828 -i eth0 -t 1ms
bin/chunkserver -r 10.11.1.210:2500 -p 2600 -n obtest -i eth0
bin/mergeserver -r 10.11.1.210:2500 -p 2800 -z 2880 -i eth0
```

```
bin/rootserver -r 10.11.1.211:2500 -i eth0 -s 10.11.1.209:2500@1#10.11.1.210:2500@2#10.1
1.1.211:2500@3 -C 3
bin/updateserver -r 10.11.1.211:2500 -p 2700 -m 2701 -i eth0
bin/mergeserver -r 10.11.1.211:2500 -p 2800 -z 2828 -i eth0 -t 1ms
bin/chunkserver -r 10.11.1.211:2500 -p 2600 -n obtest -i eth0
bin/mergeserver -r 10.11.1.211:2500 -p 2800 -z 2880 -i eth0
```

RootServer参数含义:

参数	说明
-r	需要启动的RootServer的IP地址和服务端口。主备RootServer时，为需要启动的RootServer的VIP地址和服务端口。 格式：-r [IP]:[Port]
-i	设置绑定的网卡。 格式：-i [NIC Name]
-s	设置系统中所有集群的RootServer的IP地址和服务端口，用“#”隔开。 格式：-s [IP]:[Port]# [IP]:[Port]# [IP]:[Port]
-C	设置集群ID，必须为数字。 格式：-C [Cluster ID]

UpdateServer参数含义:

参数	说明
-r	所在集群的RootServer的IP地址和端口。主备RootServer时，为所在集群的RootServer的VIP地址和服务端口。 格式：-r [IP]:[Port]
-p	设置当前UpdateServer的服务端口。 格式：-p [Port]
-m	每日合并操作时，ChunkServer请求合并数据所用的端口。 格式：-m [Port]

-i	设置绑定的网卡。 格式：-i [NIC Name]

ChunkServer参数含义:

参数	说明
-r	所在集群的RootServer的IP地址和端口。主备RootServer时，为所在集群的RootServer的VIP地址和服务端口。 格式：-r [IP]:[Port]
-p	设置当前ChunkServer的服务端口。格式：-p [Port]
-n	APP名称。与“4.6 创建各Server所需目录”中sstable的父目录名称保持一致。 格式：-n [APP Name]
-i	设置绑定的网卡。 格式：-i [NIC Name]

MergeServer参数含义:

参数	说明
-r	所在集群的RootServer的IP地址和端口。主备RootServer时，为所在集群的RootServer的VIP地址和服务端口。 格式：-r [IP]:[Port]
-p	设置当前MergeServer的服务端口。 格式：-p [Port]
-z	设置MergeServer的MySQL的协议端口。 格式：-z [Port]
-i	设置绑定的网卡。 格式：-i [NIC Name]

Listener参数含义:

参数	说明
-r	所在集群的RootServer的IP地址和端口。主备RootServer时，为所在集群的RootServer的VIP地址和服务端口。 格式：-r [IP]:[Port]
-p	设置Listener的服务端口。格式：-p [Port]
-z	设置Listener的MySQL的协议端口。缺省值为2828，不建议修改。 格式：-z [Port]
-i	设置绑定的网卡。 格式：-i [NIC Name]
-t	将该MergeServer进程指定为Listener。 格式：-t lms

4. 以admin用户登录某个RootServer所在的Cedar服务器（之后该服务器上的集群将会被设为主集群，本文档以设定“10.11.1.210”上的集群为主集群为例）。

> 说明：boot_strap操作仅第一次启动集群后需要执行。set_obi_master_first操作每次启动集群之后都需要执行（建议启动各Server之后，等待约30秒后执行set_obi_master_first命令，确保命令被正确执行）。

5. 执行以下命令，进入Cedar安装目录。

```
cd /home/admin/cedar
```

6. 依次执行以下命令，完成手动设置主集群以及初始化Cedar。

```
bin/rs_admin -r 10.11.1.210 -p 10000 set_obi_master_first
bin/rs_admin -r 10.11.1.210 -p 10000 -t 60000000 boot_strap
```

参数	说明
-r	RootServer的IP地址。主备RootServer时，为RootServer的VIP地址。 格式：-r [IP]
-p	RootServer的端口号。 格式：-r [Port]
-t	命令的超时时长。 单位：微秒。 格式：-t [Time] boot_strap

小窍门：在“/home/admin/cedar/bin”目录下，执行./rs_admin命令，可以查看help信息。

至此，Cedar集群的启动就完成了，可以通过Mysql客户端连接Cedar数据库进行相关的更新、查询操作，这里提供一种方法，详细内容可参考文档《Cedar 0.2客户端连接指南》。

```
mysql -h 10.11.1.210 -P2880 -uadmin -padmin
```

- IP为MergeServer的IP地址。
- 端口号为MySQL协议端口。
- Cedar的初始“用户名/密码”为“admin/admin”。

SQL操作指南

Cedar 0.2 数据类型及函数支持

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 数据类型及函数支持	涂云山	

1 数据类型

数据类型	说明	字段
Bigint/int/integer/mediumint/smallint/tinyint	在Cedar中，Bigint、int、integer、mediumint、smallint和tinyint无论语义还是实现都是等价的，存储为8字节有符号整型。	MYSQL_TYPE_LONGLONG
binary/char/varchar/varbinary	字符串，使用单引号。在Cedar中，varchar、char、binary和varbinary等价，均存储为varchar类型。这种类型的比较使用的是字节序。此外，在数据库建表时定义的varchar列的最大长度也是不起作用的。例如varchar(32)，实际上可以插入大于32字节的串。	MYSQL_TYPE_VAR_STRING
bool	布尔类型，表示True或者False。	MYSQL_TYPE_TINY
createtime	Cedar数据库提供的特殊的数据类型，用于记录本行数据第一次插入时的时间，由系统自动维护，用户不能直接修改。该类型的列不能作为主键的组成部分。	-

datetime/ timestamp	时间戳类型。Cedar不支持time、date等类型。注意：Cedar的时间戳格式必须为“YYYY-MM-DD HH:MI:SS”，否则插入的时间戳可能不正确。	MYSQL_TYPE_DATETIME
decimal	暂不支持。	-
double/real	表示8字节浮点数。在Cedar中，double和real等价，均存储为double类型。	MYSQL_TYPE_DOUBLE
float	表示4字节浮点数。	MYSQL_TYPE_FLOAT
modifytime	Cedar数据库提供的特殊的数据类型，用于记录本行数据最近一次被修改的时间，由系统自动维护，用户不能直接修改。该类型的列不能作为主键的组成部分。	-
numeric	暂不支持。	-

注意

- 在MySQL中，型如“1.2345”的字面量是作为decimal类型处理的，型如“1.2345e18”的字面量才作为浮点数处理。而目前版本的Cedar中，两者都作为浮点数类型double处理。
- 在MySQL中，在做表达式运算时，两个整数类型相除，结果是decimal类型。而目前版本的Cedar中，两者相除的结果为double类型。

2 函数

2.1 系统函数

函数	说明	举例
CAST(expr AS type)	将expr字段值转换为type数据类型。数据类型如上所示。	SELECT CAST(123 AS bool);
COALESCE(expr, expr, expr, …)	依次参考各参数表达式，遇到非NULL值即停止并返回该值。如果所有的表达式都是空值，最终将返回一个空值。	SELECT COALESCE(NULL,NULL,3,4,5), COALESCE(NULL,NULL,NULL);
CURRENT_TIME()	将当前日期和时间按照’YYYY-MM-DD HH:MM:SS.SSSSSS’格式的值返回。	SELECT CURRENT_TIME();
CURRENT_TIMESTAMP()	将当前日期和时间按照’YYYY-MM-DD HH:MM:SS.SSSSSS’格式的值返回。	SELECT CURRENT_TIMESTAMP();
STRICT_CURRENT_TIMESTAMP()	从UpdateServer所在服务器获取时间。	SELECT STRICT_CURRENT_TIMESTAMP();
HEX(str)	将字符串转化为十六进制数显示。	SELECT HEX(‘Cedar’);
LENGTH(str)	返回字符串的长度，单位	SELECT LENGTH(‘text’);

	为字节。	
LOWER()	将字符串转化为小写字母的字符。	SELECT LOWER(‘Hello’);
SUBSTR(str,pos,len)SUBSTR(str,pos)SUBSTR(str FROM pos)	返回一个子字符串，起始于位置pos，长度为len。使用FROM的格式为标准SQL语法。不带有len参数的时，则返回的子字符串从pos位置开始到原字符串结尾。pos值为负数时，pos的位置从字符串的结尾的字符数起。注意：len小于等于0时，返回结果为空字符串。	SELECT SUBSTR(‘abcdefg’,3), SUBSTR(‘abcdefg’,3,2), SUBSTR(‘abcdefg’,-3), SUBSTR(‘abcdefg’,3,-2);
TRIM([[{BOTH | LEADING | TRAILING}] [remstr] FROM] str)	删除字符串所有前缀和（或）后缀。若未指定BOTH、LEADIN或TRAILING,则默认为BOTH。remstr为可选项，在未指定情况下，删除空格。	SELECT TRIM(’ bar ‘), TRIM(LEADING ‘x’ FROM ‘xxxbarxxx’), TRIM(BOTH ‘x’ FROM ‘xxxbarxxx’), TRIM(TRAILING ‘x’ FROM ‘xxxbarxxx’);
UNHEX(str)	HEX(str)的反向操作，即将参数中的每一对十六进制数字理解为一个数字，并将其转化为该数字代表的字符。结果字符以二进制字符串的形式返回。	SELECT HEX(‘Cedar’), UNHEX(‘4f6365616e42617365’), 0X4f6365616e42617365，UNHEX(HEX(‘Cedar’));

2.2 聚集函数

函数	说明	举例
AVG()	返回指定组中的平均值，空值被忽略。	假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。
COUNT()	返回指定组中的行数。	假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。
MAX()	返回指定数据中的最大值。	支持：SELECT AVG(num), COUNT(num), MAX(num), MIN(num), SUM(num) FROM a;不支持：SELECT AVG(num, id);
MIN()	返回指定数据中的最小值。	支持：SELECT AVG(num), COUNT(num), MAX(num), MIN(num), SUM(num) FROM a;不支持：SELECT AVG(num, id);
SUM()	返回指定组中的和。	支持：SELECT AVG(num), COUNT(num), MAX(num), MIN(num), SUM(num) FROM a;不支持：SELECT AVG(num, id);

3 表达式

3.1 算术运算符

表达式	含义	举例
+	加法。	SELECT 2+3;

-	减法。	SELECT 2-3;
*	乘法。	SELECT 2*3;
/	除法，返回商。如果除数为“0”，则返回结果为“NULL”。	SELECT 2/3;
%或MOD	除法，返回余数。如果除数为“0”，则返回结果为“NULL”	SELECT 2%3, 2 MOD 3;

3.2 比较运算符

> 比较结果为真则返回“1”，为假则返回“0”，不确定则返回“NULL”。

表达式	含义	举例
=	等于。	SELECT 1=0, 1=1, 1=NULL;
=	大于等于。	SELECT 1>=0, 1>=1, 1>=2, 1>=NULL;
>	大于。	SELECT 1>0, 1>1, 1>2， 1>NULL;
<=	小于等于。	SELECT 1<=0, 1<=1, 1<=2, 1<=NULL;
<	小于。	SELECT 1<0, 1<1, 1<2， 1
!=或<>	不等于。	SELECT 1!=0, 1!=1, 1<>0, 1<>1, 1!=NULL, 1<>NULL;
BETWEEN	存在于指定范围。	SELECT 2 BETWEEN 1 AND 2, 3 BETWEEN 1 AND 2;
IN	存在于指定集合。	SELECT 2 IN (1, 2), 3 IN (1, 2);
IS NULL	为NULL。	SELECT 0 IS NULL, NULL IS NULL;
IS NOT NULL	不为NULL。	SELECT 0 IS NOT NULL, NULL IS NOT NULL;
LIKE	字符串通配符匹配。通配符包括“%”和“”： “%”表示匹配任何长度的任何字符，且匹配的字符可以不存在。 “”表示只匹配单个字符，且匹配的字符必须存在。如果你需要查找“a_c”，而不是“abc”时，可以使用Cedar的转义字符“\\”，即可以表示为“a_c”。	SELECT ‘abcd’ LIKE ‘ab%’, ‘abcd’ LIKE ‘%ab%’, ‘abcd’ LIKE ‘ab_’,’abcd’ LIKE ‘ab__’, ‘abcd’ LIKE ‘_ab__’; SELECT ‘abcd’ LIKE ‘ab_d’;

3.3 逻辑运算符

表达式	含义	举例
NOT	逻辑非。	SELECT NOT 0， NOT 1， NOT NULL;
AND	逻辑与。	SELECT (0 AND 0), (0 AND 1), (1 AND 1), (1 AND NULL);
OR	逻辑或。	SELECT (0 OR 0), (0 OR 1), (1 OR 1), (1 AND NULL);

3.4 拼接运算符

表达式	含义	举例

||	将值联结到一起构成单个值。	SELECT ‘a’||’b’;

3.5 优先级

优先级	运算符
1	* , / , % , MOD
2	+ , -
3	= , > , >= , < , <= , <> , != , IS , LIKE , IN
4	BETWEEN
5	NOT
6	AND
7	OR

> 在实际运用的时，我们可以用“()”将需要优先的操作括起，这样既起到优先操作的作用，又使其他用户便于阅读。

4 转义字符

转义字符	含义	转义字符	含义
\b	退格符。	\’	单引号。
\f	换页符。	\”	双引号。
\n	换行符。	_	_字符。
\r	回车符。	\%	%字符。
\t	tab字符。	\0	空字符(NULL)。
\	反斜线字符。		

Cedar 0.2 SQL使用指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 SQL使用指南	黄晨晨 王冬慧	

SQL（Structured Query Language）是一种组织、管理和检索数据库存储数据的计算机语言。由于Cedar完全兼容MySQL的网络协议，所以Cedar SQL用户可以使用MySQL客户端、Java客户端和C客户端连接Cedar。客户端的连接可参考《Cedar 0.2客户端连接指南》。

Cedar SQL语句中的关键字、表名、列名、函数名等均大小写不敏感。表名和列名都转换为小写之后存入Schema中，所以即使用户建表时候列名是大写的，查询的时候获得的列名也是小写。如果您需要保存大写字母，请使用双引号，例如：”Info”。 Cedar SQL语法遵循SQL92标准，单引号表示字符串；双引号表示表名、列名或函数名。双引号内可以出现SQL保留的关键字。 Cedar没有Database的概念，可以理解为一个Cedar集群只有一个Database，所以用户不需要也不能使用“USE DATABASE”语句来指定Database。

1 数据定义语言

1. **show tables 语句**

> 查看数据库中有哪些表。

2. **show index 语句**

> 显示索引：将一张表上的索引列出
>
> show index on <主表名>
> <主表名>：索引表的的主表

3. **create table 语句**

> 在数据库中创建新表。
>
> 格式：CREATE TABLE [IF NOT EXISTS] table_name (column_name data_type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT], …, PRIMARY KEY (column_name1, column_name2…)) [table_options_list];

table_options_lists参数设置如下：

参数	含义	举例
EXPIRE_INFO	在UpdateServer中的动态数据和ChunkServer中的静态数据合并时，满足表达式的行会自动删除。一般可用于自动删除过期数据。	EXPIRE_INFO = '$SYS_DATE > c1 + 24*60*60'，表示自动删除c1列的值比当前时间小24小时的行，其中"24*60*60"表示过期的微秒数。
TABLET_MAX_SIZE	这个表的Tablet最大尺寸，单位为字节，默认为256MB。	TABLET_MAX_SIZE = 268435456
REPLICA_NUM	这个表的tablet复本数，默认值为3。	REPLICA_NUM = 3
COMPRESS_METHOD	存储数据时使用的压缩方法名，目前提供的方法有以下三种：none（默认值，表示不作压缩）、lzo_1_0、 snappy_1_0。	COMPRESS_METHOD = 'none'
USE_BLOOM_FILTER	对本表读取数据时，是否使用Bloom Filter（0：默认值，不使用； 1：使用）。	USE_BLOOM_FILTER = 0
COMMENT	添加注释信息。	COMMENT='create by Bruce'

应用举例

```
CREATE TABLE test (c1 int primary key, c2 varchar) ;
```

注意事项

> • 使用"IF NOT EXISTS"时，即使创建的表已经存在，也不会报错，如果不指定时，则会报错。
> • "data_type"请参见"数据类型及函数支持"章节。
> • NOT NULL，DEFAULT，AUTO_INCREMENT用于列的完整性约束。在选项可以在SQL语句中出现，但目前尚未实现。

4. **create index 语句**

> 对指定表与列建立索引。
>
> CREAT INDEX <索引表名> ON <主表名>([<索引列名>,..]) STORING([<冗余列名>,..])
> <索引表名>：允许用户自定义，但部分特殊字符不支持使用。
> <主表名>：已经存在的数据表。
> <索引列名>：使用主表中的任意主键。
> <冗余列名>：使用主表中的非主键列（已经被选为索引列的列不支持冗余）.

应用举例

```
CREATE INDEX index_test ON test(student_id) STORING(student_name) ;
```

5. **alter table 语句**

> 修改已存在的表的设计。
>
> 格式：
> 增加列：ALTER TABLE table_name ADD [COLUMN] column_name data_type;
> 删除列：ALTER TABLE table_name DROP [COLUMN] column_name;

应用举例

```
ALTER TABLE test ADD c3 int;
ALTER TABLE test DROP c3;
```

6. **drop table 语句**

> 删除数据库中的表。
>
> 格式：DROP TABLE [IF EXISTS] table_name;

应用举例

```
DROP TABLE IF EXISTS test;
```

注意事项

> • 使用“IF EXISTS”时，即使要删除的表不存在，也不会报错，如果不指定时，则会报错。
>
> • 同时删除多个表时，用“,”隔开。

7. **drop index 语句**

> 将特定主表上的索引删除。
>
> DROP INDEX [<索引表名>,..] on <主表名>
> <索引表名>：已经存在的索引表，可缺省，表示删除主表上的所有索引。
> <主表名>：索引表的的主表。

应用举例

```
DROP INDEX ON test;
```

2 数据操作语言

8. **insert 语句**

> 添加一个或多个记录到表。
>
> 格式： INSERT INTO table_name [(column_name,…)]
> VALUES (column_values,…);

应用举例

```
INSERT INTO test VALUES (1, 'Hello Dase'),(2, 'Hello Cedar');
```

注意事项

> • [(column_name,…)]用于指定插入数据的列。
>
> • 同时插入多列时，用“,”隔开 。

9. **replace 语句**

> 语法和INSERT相同，语义有别：如果本行已经存在，则修改对应列的值为新值；如果不存在，则插入。
>
> 格式：REPLACE INTO table_name [(column_name,…)]
> VALUES (column_values,…);

应用举例

```
REPLACE INTO test VALUES (1, 'hello DaSE'),(2, 'hello Cedar');
```

注意事项

> • [(column_name,…)]用于指定插入数据的列。
>
> • 同时插入多列时，用“,”隔开。

10. **update 语句**

> 修改表中的字段值。
>
> 格式：UPDATE table_name SET column_name1=column_values
> [column_name2=column_values] …
> WHERE where_condition;

应用举例

```
UPDATE test SET c2 ='Hello Cedar' WHERE c1 = 2;
```

注意事项

> • [(column_name,…)]用于指定更新数据的列。
>
> • 同时更新多列时，用“,”隔开

11. **delete 语句**

> 删除表中符合条件的行。
>
> 格式：DELETE FROM table_name WHERE where_condition;

应用举例

```
DELETE FROM test WHERE c1 = 2;
```

注意事项

> • where_condition必须指定关于Primary Key的条件。
>
> • Cedar数据库目前只支持指定Primary Key条件的删除操作。

12. **select 语句**

> 查询表中的内容。
>
> SELECT [ALL | DISTINCT] select_list [AS other_name] FROM table_name [WHERE where_conditions] [GROUP BY group_by_list] [HAVING search_confitions] [ORDER BY order_list [ASC | DESC]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];
>
> • where：用来设置一个筛选条件，查询结果中仅包含满足条件的数据。
>
> • group by：用于进行分类汇总。
>
> • having：having子句与where子句类似，但是having字句可以使用累计函数（如SUM，AVG等）。
>
> • order by：用来按升序（ASC）或者降序（DESC）显示查询结果。不指定ASC或者DESC时，默认为ASC。
>
> • limit：强制 select 语句返回指定的记录数。

应用举例

```
select * from test limit 10;
```

注意事项

> • 当表中存在多条符合条件的记录时，在调用时会报错。
>
> • 当表中不存在符合条件的记录时，调用时不报错。
>
> • 当source和target数目不相等时，在调用存储过程时报错。

13. **join 语句**

JOIN连接分为内连接和外连接。外连接又分为左连接、右连接和全连接。两个表联接后，可以使用ON指定条件进行筛选。

SELECT [ALL | DISTINCT] select_list [AS other_name] FROM table_name [WHERE where_conditions] [GROUP BY group_by_list] [HAVING search_confitions] [ORDER BY order_list [ASC | DESC]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];

• where：用来设置一个筛选条件，查询结果中仅包含满足条件的数据。

• group by：用于进行分类汇总。

• having：having子句与where子句类似，但是having字句可以使用累计函数（如SUM，AVG等）。

• order by：用来按升序（ASC）或者降序（DESC）显示查询结果。不指定ASC或者DESC时，默认为ASC。

• limit：强制 select 语句返回指定的记录数。

应用举例

```
内连接：结果中只包含两个表中同时满足条件的行。
SELECT a.id, a.name, b.colour FROM a INNER JOIN b ON a.id = b.id;

左连接：结果中包含位于关键字LEFT [OUTER] JION左侧的表中的所有行，以及该关键字右侧的表中满足条件的行。
SELECT a.id, a.name, b.colour FROM a LEFT OUTER JOIN b ON a.id = b.id;

右连接：结果中包含位与关键字[RIGHT] [OUTER] JOIN右侧的表中的所有行，以及该关键字左侧的表中满足条件的行。
SELECT a.id, a.name, b.colour FROM a RIGHT OUTER JOIN b ON a.id = b.id;

全连接：结果中包含两个表中的所有行。
SELECT a.id, a.name, b.colour FROM a FULL OUTER JOIN b ON a.id = b.id;
```

注意事项

目前，Cedar的JOIN不支持USING子句，并且JOIN的连接条件中必须至少有一个等值连接条件。

14. **SemiJoin 语句**

指明进行半连接操作。

SELECT /*+SEMI_JOIN(小表表名,大表表名,小表连接列的列名,大表连接列的列名)*/ column_name(s) from 小表表名 inner join 大表表名 on 小表连接列的列名 = 大表连接列的列名;（其中/* */里面的内容为hint）

应用举例

```
SELECT /*+SEMI_JOIN(A,B,A.c2,B.c3)*/ A.c1, B.c1 FROM A INNER JOIN B ON A.c2 = B.c3;
```

注意事项

• 多表连接时，只能指定前两张表做semijoin。

• 只有两张表原来是inner join 的时候，才能指定这两张表做semijoin。

• 只有当大表的数据远远超过小表，并且两张表在连接列上的交集很小的时候，semijoin优化的效果最好，其他情况下的优化效果可能不是很好。

• 插入数据后，最好在每日合并后，再做查询操作，这样的优化效果会好一点。

15. **BloomFilter Join 语句**

指明多表连接使用BloomFilter Join。

SELECT /*+ JOIN(bloomfilter_join,merge_join,…)*/ <查询内容> from <表名> left (inner, right, full outer) join <表名> on <join条件>；

应用举例

```
SELECT /*JOIN(bloomfilter_join)*/ FROM test1 INNER JOIN test2 ON test1.a = test2.c;
```

注意事项

• bloomfilter join不支持在不同类型的连接列上做等值连接。

• 若hint中的join类型个数多于后面数据表的两两连接个数，则忽略hint中多余的join类 型；若hint中的join类型个数少于后面数据表的两两连接个数，则后面未指定的连接 都默认执行merge join。

• bloomfilter join不支持right、full out join算法，若指定了bloomfilter join类型而连接类型为right、full outer join，还是默认执行merge join。

• 对于select * from tab1,tab2 where tab1.col1=tab2.col2; 此类Join查询需要DBA改为inner join后再指定Hint，完成bloomfilter join；否则将执行原merge join操作。

• 只有当左表在连接列上不重复值的个数不超过100万，并且右表的数据行数减去左表 选择出的右表行数，即过滤掉的右表数据越大，bloomfilter join的优化效果越好。建议 过滤掉的右表数据大于500万行时使用bloomfilter join。 多表连接时，根据连接顺序对每两张表使用以上规则，以达到优化效果。

16. **create procedure 语句**

创建一个存储过程。该存储过程被存储在系统表__all_procedure中。

CREATE PROCEDURE 存储过程名([proc_parameter,proc_parameter…])
BEGIN
[routine_body]
END;

#proc_parameter: [IN | OUT | INOUT] param_name type
#type: Any valid Cedar data type
#routine_body: Valid SQL procedure statement or statements

应用举例

```
CREATE PROCEDURE demo_sp(IN @in_param int, OUT @out_patam int)
BEGIN
END
#创建了一个名称为demo_sp并且一个参数是int类型的输入参数，另一个是int类型的输出参数
```

注意事项

> • 参数前必须有in,out,inout修饰，且变量必须以@开始，如CREATE PROCEDURE pname(in @para int)…
>
> • IN变量如果调用外部变量，该变量必须set。
>
> • 参数项可以为空，但是()不能省略，如CREATE PROCEDURE pname()…
>
> • statament_list可以为空，表示该存储过程为一个空的存储过程。
>
> • routine_body包含合法的SQL过程语句,可以使用复合语句语法，一个存储过程只允许一个BEGIN…END语句块，暂时不支持多个。

17. **drop procedure 语句**

> 删除一个存储过程。将该存储过程从系统表__all_procedure中删除。
>
> DROP PROCEDURE [IF EXISTS] 存储过程名;

应用举例

```
DROP PROCEDURE IF EXISTS demo_sp;
```

注意事项

> IF EXISTS可以为空，若该项为空时，删除的存储过程不存在时被报错。若该项不为空，删除的存储过程无论存在与否，都不会报错。

18. **call 语句**

> 调用一个存储过程。用户将相应参数传进去，利用call语句执行存储过程。
> CALL 存储过程名([参数#1,…参数#1024]);

应用举例

```
CALL demo_sp(100, @out_param)
#调用名称为demo_sp的存储过程，第一个参数为输入参数，第二个参数为输出参数
```

注意事项

> • 若该存储过程无参数，调用时()不能省略。如call pname()。
>
> • IN变量在调用前必须declare。

19. **IF 语句**

条件分支语句。用来判定所给定的条件是否满足，根据判定的结果决定执行所走的分支。

IF expr THEN
  statement_list;
[ELSEIF expr THEN]
  statement_list;、
[ELSE]
  statement_list;
END IF;

应用举例

```
DECLARE @cond_param int default 0;
DECLARE @result_param int;
IF @cond_param > 0 THEN
  #如果cond_param大于0则把result_param赋值为100
  SET @result_param = 100;
ELSE
  SET @result_param = -100;
END IF;
```

注意事项

- IF语句中，ELSEIF和ELSE分支可以为空。
- 若分支存在，则分支下面的statemnet_list不可以为空，否则在创建存储过程时报错。

20. **WHILE 语句**

循环结构。当expr为真时，执行statement_list语句，当expr为假时，跳出循环。

WHILE expr DO
  statement_list ;
END WHILE;

应用举例

```
DECLARE @cond_param int default 0;
DECLARE @result_param int default 0;
WHILE @cond_param < 10 do
  #如果cond_param小于10则把result_param加上100
  SET @result_param = @result_param + 100;
END WHILE;
```

注意事项

若statement_list为空时，在创建存储过程时系统判断为死循环，报错。

21. **FOR LOOP 语句**

循环语句，由FOR控制循环迭代的次数，范围为[lower_bound, upper_bound)。

REVERSE：可选项，指定循环方式。默认的循环方式由下标(lower_bound)到上标(upper_bound)。使用该选项则从上标界到下标界。

lower_bound：循环范围的下标界。

upper_bound：循环范围的上标界。

```
FOR loop_counter IN [REVERSE] lower_bound TO upper_bound LOOP
   statement_list;
END LOOP;
```

应用举例

```
DECLARE @c2 BOOL DEFAULT;
DECLARE @c3 FLOAT DEFAULT 10.000;
DECLARE @c4 DOUBLE DEFAULT 5.0001;
DECLARE @c5 VARCHAR(10000) DEFAULT 'asdasdasedwqe221312dsd';
DECLARE @c6 TIMESTAMP DEFAULT '1993-03-07 10:08:24';
DECLARE @b INT DEFAULT 1;
FOR @c1 IN 1 TO 5 LOOP
INSERT INTO ptest VALUES (@c1, @c2, @c3, @c4, @c5, @c6);
END LOOP;
```

注意事项

- 循环的范围大小在循环执行之前就已经确定，不会在循环执行时发生改变。
- lower_bound必须小于upper_bound。若lower_bound大于等于upper_bound，则该循环不会被执行。
- 迭代变量loop_counter不需要declare，并且循环体内不能修改迭代变量。for-loop结束后，迭代变量会删除。
- FOR LOOP中嵌套FOR LOOP时，两个循环的迭代变量loop_counter不能相同。

22. **LOOP 语句（使用EXIT语句跳出循环）**

循环语句，由EXIT语句控制循环的结束。

WHEN expr：可选项。指定循环跳出的条件，若无该项，遇到EXIT直接跳出循环，否则，当expr满足时跳出循环。

```
LOOP
   statement_list;
EXIT [WHEN expr];
END LOOP;
```

应用举例

```
DECLARE @c1 INT DEFAULT 1;
DECLARE @c2 BOOL DEFAULT false;
DECLARE @c3 FLOAT DEFAULT 10.000;
DECLARE @c4 DOUBLE DEFAULT 5.0001;
DECLARE @c5 VARCHAR(10000) DEFAULT 'asdasdasedwqe221312dsd';
DECLARE @c6 TIMESTAMP DEFAULT '1993-03-07 10:08:24';
DECLARE @b INT DEFAULT 1;
LOOP
     INSERT INTO ptest VALUES (@c1, @c2, @c3, @c4, @c5, @c6);
EXIT WHEN @c1>10;
SET @c1=@c1+1;
END LOOP;
```

注意事项

> LOOP语句中至少要包含一个EXIT语句，该语句可以在CASE WHEN语句中，IF ELSE语句中。若无EXIT语句，在创建存储过程时报错。

23. **CASE WHEN 语句**

> 选择分支结构，根据case_expr走不同的分支。
>
> CASE case_expr
> WHEN when_expr THEN
> statement_list;
> …
> [ELSE]
> statement_list;
> END CASE;

应用举例

```
DEFAULT @c1 INT DEFAULT 10;
CASE @c1
    WHEN 5 THEN  proc_if values (1,2);
    WHEN 10 THEN INSERT INTO proc_if VALUES (2,2);
    ELSE INSERT INTO proc_if VALUES (3,2);
END CASE;
```

注意事项

> • ELSE分支可以为空。
>
> • 若分支存在，则分支下面的statemnet_list不可以为空，否则在创建存储过程时报错。
>
> • 在调用存储过程时，若CASE WHEN分支一个都没走到则会报错。即，CASEWHEN结构在执行时必须走且只走其中一个分支。
>
> • CASE WHEN在执行时，when_expr会和case_expr相比较，若两者不为同一种类型，则不能比较（数值类型除外）。例如，当case_expr为@para，@para类型为int时，when_expr为@para>5时，when_expr为bool类型，不能与case_expr比较。同样，当when_expr为'abcdef'时，也不能与case_expr比较。但是，在我们的方案中，int,double,float之间可以两两比较。如，（int）10=(float)10.00，(int)10！=(double)10.01，诸如此类的数值类型可以比较。

24. **SELECT INTO 语句**

从表中筛选出某些列对应赋值给变量。

SELECT source1,source2… INTO target1,target2… FROM tname

WHERE…;

应用举例

```
DECLARE @email_param varchar;
#定义一个变量用来存储SELECT INTO查询出来的结果

SELECT email INTO @email_param FROM user WHERE id = 1;
#user表由id和email字段组成
```

注意事项

- 当表中存在多条符合条件的记录时，在调用时会报错。
- 当表中不存在符合条件的记录时，调用时不报错。
- 当source和target数目不相等时，在调用存储过程时报错。

25. **DECLARE 语句**

定义某种类型的变量并赋值。

DECLARE var_name[,var_name…] type [DEFAULT value];

应用举例

```
DECLARE @param1 int default 0; #定义一个变量param1默认值为0
DECLARE @param2 int; #定义一个变量param2

#定义两个变量param3与param4默认值为0
DECLARE @param3, @param4 int default 0;
```

注意事项

- DECLARE语句只能出现在存储过程的开头部分，即，若出现其他语句后再使用DECLARE语句会在存储过程创建时报错。
- DEFAULT后面不支持表达式形式。
- var_name使用@+变量名。
- var_name不能与变量名重复。
- 若DEFAULT项不存在，则为系统object初始化时默认值（一般为0）。

26. **SET 语句**

给变量赋值。

DECLARE @param1 int;
SET @param1 = 10; #将param1赋值为10
SET @param1 = @param1 + 1; #将param1的值加1
SET @param1 = 3 + 5; #将param1的值赋值为等号右边表达式的值
SET var_name = expr [, var_name = expr] …;

注意事项

• 当var_name和expr类型不一样时，会发生由expr类型到var_name类型的强制转化。

• SET操作不支持+=运算符

Tips：

可以成为statement_list的语句有：

SELECT 语句、UPDATE 语句、INSERT 语句、DELETE 语句、SELECT INTO 语句、IF 构造、WHILE 构造、CASE 构造、SET 语句 、DECLARE 语句。

27. **union 语句**

用于合并两个或多个SELECT语句的结果集。

SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2;

应用举例

```
SELECT name FROM test1 UNION SELECT mingzi FROM test2;
```

注意事项

• UNION内部的SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条SELECT语句中的列的顺序必须相同。

• 默认地，UNION操作符选取不同的值。如果允许重复的值，请使用UNION ALL。

• UNION 结果集中的列名总是等于UNION中第一个SELECT语句中的列名。

28. **except 语句**

用于查询存在第一个集合中，但是不存在于第二个集合中的数据。

{（<SQL - 查询语句 1>）}
{ EXCEPT }
{（<SQL - 查询语句 2>）};

应用举例

```
SELECT * FROM test1 EXCEPT SELECT * FROM test2;
```

29. **intersect 语句**

> 用于查询存在第一个集合中，但是不存在于第二个集合中的数据。
>
> { （<SQL - 查询语句 1>）}
> { INTERSECT }
> { （<SQL - 查询语句 2> ）};

应用举例

```
SELECT * FROM a INTERSECT SELECT * FROM d;
```

30. **dual 语句**

> DUAL是一个虚拟的表，可以视为一个一行零列的表。当我们不需要从具体的表来取得表中数据，而是单纯地为了得到一些我们想得到的信息，并要通过SELECT完成时，就要借助一个对象，这个对象就是DUAL。一般可以使用这种特殊的SELECT语法获得用户变量或系统变量的值。
>
> SELECT [ALL | DISTINCT] select_list [FROM DUAL [WHERE where_condition]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];

应用举例

```
执行以下命令，计算2和3的积。

SELECT 2*3 FROM DUAL;
```

注意事项

> 当SELECT语句没有FROM子句的时候，语义上相当于FROM DUAL，此时，表达式中只能是常量表达式。

31. **select ... for update 语句**

> SELECT ... FOR UPDATE可以用来对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。即使用FOR UPDATE语句将锁住查询结果中的元组，这些元组将不能被其他事务的UPDATE，DELETE和FOR UPDATE操作，直到本事务提交。

应用举例

```
SELECT * FROM a where id = 1 FOR UPDATE;
```

注意事项

> 必须是单表查询。

32. **in 和 or 语句**

> Cedar支持逻辑运算“IN”和“OR”，其中IN可以直接定位到查询的数据，而OR需要进行全表扫描，因此我们推荐使用IN语句。
>
> SELECT [ALL | DISTINCT] select_list [FROM DUAL [WHERE where_condition]] [LIMIT {[offset,] row_count | row_count OFFSET offset}];

应用举例

```
以下两个句法等价，但第一个句法执行效率高。
SELECT * FROM test WHERE id IN (1,2);
SELECT * FROM test WHERE id = 1 OR id = 2;
```

注意事项

> 必须是单表查询。

3 事务处理相关

33. **transaction**

> 数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作。事务处理可以用来维护数据库的完整性，保证成批的SQL操作全部执行或全部不执行。
>
> 开启事务：
> START TRANSACTION [WITH CONSISTENT SNAPSHOT]; 或者BEGIN [WORK];
>
> 提交当前事务：
> COMMIT [WORK];
>
> 回滚当前事务：
> ROLLBACK [WORK];

应用举例

```
START TRANSACTION;
UPDATE test SET name = 'c' WHERE id = 3;
INSERT INTO test VALUES (4, 'a', 30);
COMMIT;
```

注意事项

• 我们采用datatime类型的值作为主键，在进行INSERT操作的时候，不能直接在SQL INSERT语句中采用CURRENT_TIME()函数来获取时间作为主键列的值，比如INSERT INTO a VALUES(4, ‘a’, 30, CURRENT_TIME())， 则会报错的。我们只能在INSERT之前，先用CURRENT_TIME()函数获取当前时间，然后在INSERT语句中直接用该确定的时间值。

• 由于事务有一个超时时间，所以当我们手动输入事务中的多个SQL语句的时候，由于打字时间太长，会导致整个事务超时。解决方法是：修改当前session中的系统变量ob_tx_timeout，使得该值进可能大，而不至于导致超时。

• 目前在Cedar的事务中执行SELECT语句，不能读取当前事务中未提交的数据。用户如果有这个需求，可以暂时用SELECT … FOR UPDATE语句代替。

• WITH CONSISTENT SNAPSHOT子句用于启动一个一致的读取。该子句的效果与发布一个START TRANSACTION，后面跟一个来自任何Cedar表的SELECT的效果一样。

• BEGIN和BEGIN WORK被作为START TRANSACTION的别名受到支持，用于对事务进行初始化。START TRANSACTION是标准的SQL语法，并且是启动一个ad-hoc事务的推荐方法。一旦开启事务，则随后的SQL数据操作语句（即INSERT，UPDATE，DELETE，不包括REPLACE）直到显式提交时才会生效。

34. **设置snapshot isolation**

设置会话隔离级别为snapshot isolation。

SET @@SESSION.tx_isolation=’REPEATABLE-READ’;

35. **lock table 语句**

为表添加排它锁

LOCK TABLE table_name;

应用举例

```
LOCK TABLE test;
```

注意事项

当事务执行结束时，相应的表锁会自动被释放。

4 数据库管理语句

36. **create user**

用于创建新的Cedar用户。

CREATE USER ‘user’ [IDENTIFIED BY [PASSWORD] ‘password’];

应用举例

```
CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
```

注意事项

• 新建用户仅拥有系统表”__all_server”和“__all_cluster”的SELECT权限，以及“__all_client”的REPLACE和SELECT权限。

• 必须拥有全局的CREATE USER权限或对“__all_user”表的INSERT权限，才可以使用CREATE USER命令。

• 新建用户后，“__all_user”表会新增一行该用户的表项。如果同名用户已经存在，则报错。

• 使用自选的IDENTIFIED BY子句，可以为账户给定一个密码。

• 此处密码为明文，存入“__all_user”表后，服务器端会变为密文存储下来。

• 同时创建多个用户时，用“,”隔开。

37. **drop user**

用于删除一个或多个Cedar用户。

DROP USER ‘user’;

应用举例

```
DROP USER 'sqluser02';
```

注意事项

• 必须拥有全局的CREATE USER权限或对“__all_user”表的DELETE权限，才可以使用DROP USER命令。

• 成功删除用户后，这个用户的所有权限也会被一同删除。

• 同时删除多个用户时，用“,”隔开。

38. **set password 和 alter user**

用于修改Cedar登录用户的密码。

SET PASSWORD [FOR ‘user’ =] ‘password’;
或者
ALTER USER ‘user’ IDENTIFIED BY ‘password’;

应用举例

```
SET PASSWORD FOR 'sqluser01' = 'abc123';

ALTER USER 'sqluser01' IDENTIFIED BY 'abc123';
```

注意事项

• 如果没有For user子句，则修改当前用户的密码。任何成功登陆的用户都可以修改当前用户的密码。
• 如果有For user子句，或使用第二种语法，则修改指定用户的密码。必须拥有对“__all_user”表的UPDATE权限，才可以修改制定用户的密码。

39. **rename user**

用于修改Cedar登录用户的用户名。

RENAME USER ‘old_user’ TO ‘new_user’;

应用举例

```
RENAME USER 'sqluser01' TO 'obsqluser01';
```

注意事项

• 必须拥有全局CREATE USER权限或者对__users表的UPDATE权限，才可以使用本命令。

• 同时修改多个用户名时，用“,”隔开。

40. **locked 和 unlocked**

锁定或者解锁用户。被锁定的用户不允许登录。

锁定用户：ALTER USER ‘user’ LOCKED;
解锁用户：ALTER USER ‘user’ UNLOCKED;

应用举例

```
锁定用户：ALTER USER 'obsqluser01' LOCKED;
解锁用户：ALTER USER 'obsqluser01' UNLOCKED;
```

注意事项

必须拥有对“__users”表的UPDATE权限，才可以执行本命令。

41. **grant**

用于系统管理员授予Cedar用户操作权限。

GRANT priv_type ON table_name TO ‘user’;

应用举例

```
GRANT ALL PRIVILEGES, GRANT OPTION ON * TO 'obsqluser01';
```

注意事项

• 给特定用户授予权限。如果用户不存在则报错。

• 当前用户必须拥有被授予的权限（例如,user1把表t1的SELECT权限授予user2，则user1必须拥有表t1的SELECT的权限），并且拥有GRANT OPTION权限，才能授予成功。

• 用户授权后，该用户只有重新连接Cedar，权限才能生效。

• 用“*”代替table_name，表示赋予全局权限，即对数据库中的所有表赋权。

• 同时把多个权限赋予用户时，权限类型用“,”隔开。

• 同时给多个用户授权时，用户名用“,”隔开。

pri_type参数表：

权限	说明
ALL PRIVILEGES	除GRANT OPTION以外所有权限。
ALTER	ALTER TABLE的权限。
CREATE	CREATE TABLE的权限。
CREATE USER	CREATE USER，DROP USER，RENAME USER和REVOKE ALL PRIVILEGES的权限。
DELETE	DELETE的权限。
DROP	DROP的权限。
GRANT OPTION	GRANT OPTION的权限。
INSERT	INSERT的权限。
SELECT	SELECT的权限。
UPDATE	UPDATE的权限。
REPLACE	REPLACE的权限。
SUPER	SET GLOBAL修改全局系统参数的权限。

42. **revoke**

用于系统管理员撤销Cedar用户的操作权限。

REVOKE priv_type ON table_name FROM ‘user’;

应用举例

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';
```

注意事项

• 用户必须拥有被撤销的权限（例如，user1要撤销user2对表t1的SELECT权限，则user1必须拥有表t1的SELECT的权限），并且拥有GRANT OPTION权限。

• 撤销“ALL PRIVILEGES”和“GRANT OPTION”权限时，当前用户必须拥有全局GRANT OPTION权限，或者对权限表的UPDATE及DELETE权限。

• 撤销操作不会级联。例如，用户user1给user2授予了某些权限，撤回user1的权限不会同时也撤回user2的相应权限。

• 用“*”代替table_name，表示撤销全局权限，即撤销对数据库中所有表的操作权限。

• 同时对用户撤销多个权限时，权限类型用“,”隔开。

• 同时撤销多个用户的授权时，用户名用“,”隔开。

• pri_type参数设置与“grant”类似。

43. **show grants**

用于系统管理员授予Cedar用户操作权限。

SHOW GRANTS [FOR user];

应用举例

```
SHOW GRANTS FOR 'sqluser01';
```

注意事项

• 如果不指定用户名，则缺省显示当前用户的权限。对于当前用户，总可以查看自己的权限。
• 如果要查看其他指定用户的权限，必须拥有对“__all_user”的SELECT权限。

44. **修改用户变量**

SET @var_name = expr;

应用举例

```
SET @a=2, @b=3; SET @c = @a + @b;

SELECT @a, @b, @c;
```

注意事项

• 用户变量的形式为@var_name，其中变量名var_name可以由当前字符集的文字数字字符、“.”、“_”和“$”组成。

• 每个变量的expr可以为整数、实数、字符串或者NULL值。

• 同时定义多个用户变量时，用“,”隔开。

45. **修改系统变量**

设置全局变量：
SET GLOBAL system_var_name = expr;或者SET @@GLOBAL.system_var_name = expr;

设置会话变量：
SET [SESSION | @@SESSION. | LOCAL | LOCAL. | @@]system_var_name = expr; - 35 -

查看系统变量：
注：如果指定GLOBAL或SESSION修饰符，则分别打印全局或当前会话的系统变量值；如果没有修饰符，则显示当前会话值。
SHOW [GLOBAL | SESSION] VARIABLES [LIKE ‘system_var_name’ | WHERE expr];

应用举例

```
1. 执行以下命令，修改会话变量中的SQL超时时间。
   SET @@SESSION.ob_tx_timeout = 900000;

2. 执行以下命令，查看SQL超时时间。
   SHOW SESSION VARIABLES LIKE 'ob_tx_timeout';
```

注意事项

全局变量的更改不影响目前已经连接的客户端的会话变量，即使客户端执行SET GLOBAL语句也不影响。

46. **修改系统配置项**

修改系统配置项：
ALTER SYSTEM SET param_name = expr [COMMENT ‘text’] [SCOPE = conf_scope] SERVER_TYPE = server_type [CLUSTER = cluster_id | SERVER_IP= ‘server_ip’ SERVER_PORT = server_port];

查看系统配置项：
SHOW PARAMETERS [LIKE ‘pattern’ | WHERE expr];

应用举例

```
1. 执行以下命令，修改Tablet副本数。
ALTER SYSTEM SET tablet_replicas_num=3 COMMENT 'Modify by Bruce' SCOPE = SPFILE SERVER_TYPE
= ROOTSERVER SERVER_IP= '10.10.10.2' SERVER_PORT= 1234;

2. 查看“tablet_replicas_num”。
SHOW PARAMETERS LIKE 'tablet_replicas_num';
```

注意事项

同时修改多个系统配置项时，用“,”隔开。

47. **设置字符编码**

应用举例

```
1. 执行以下命令，设置Cedar字符集为“utf8”。
   SET @@SESSION.ob_charset ='utf8';

2. 执行以下命令，查看Cedar字符集。
   SHOW VARIABLES LIKE 'ob_charset';
```

注意事项

> Cedar返回给用户的字符集的参数名为“ob_charset”，缺省值为“gbk”。在设置该字符集时，应注意会话变量和全局变量的区别。

5 预备执行语句

48. **预备执行语句**

> 用户先通过客户端发送一个预备语句把要执行的SQL数据操作语句发给服务器，服务器端会解析这个语句，产生执行计划并返回给客户端一个句柄（名字或者ID）。随后，用户可以使用返回的句柄和指定的参数反复执行一个预备好的语句，省去了每次执行都解析SQL语句的开销，可以极大地提高性能。
>
> PREPARE语句
> PREPARE stmt_name FROM preparable_stmt;
>
> EXECUTE语句
> EXECUTE stmt_name [USING @var_name [, @var_name] …];
>
> DEALLOCATE语句
> DEALLOCATE PREPARE stmt_name;
> 或者
> DROP PREPARE stmt_name;

应用举例

```
依次使用PREPARE查询表a中id=2的行。
PREPARE stmt1 FROM SELECT name FROM a WHERE id=?;
SET @id = 2;
EXECUTE stmt1 USING @id;
DEALLOCATE PREPARE stmt1;
```

注意事项

• preparable_stmt为SQL数据操作语句，预备好的语句在整个SQL会话期间可以使用stmt_name这个名字来执行。

• 数据操作语句（即SELECT, REPLACE, INSERT, UPDATE, DELETE）都可以被预备执行

• 在被预备的SQL语句中，可以使用问号（？）表明一个之后执行时才绑定的参数。问号只能出现在SQL语句的常量中。一个被预备的语句也可以不包含问号。

• 一个使用PREPARE语句预备好的SQL语句，可以使用EXECUTE语句执行。

• 如果预备语句中有问号指明的绑定变量，需要使用USING子句指明相同个数的执行时绑定的值。USING子句后只能使用SET语句定义的用户变量。

• 删除一个指定的预备语句。一旦删除，以后就不能再执行。

6 其他SQL语句

49. **show 语句**

> 查看指定表的列的信息。
> SHOW COLUMNS {FROM | IN} table_name [LIKE ‘pattern’ | WHERE expr];
>
> 查看可以用来建立指定表格的建表语句。
> SHOW CREATE TABLE table_name;
>
> 查看数据库中存在哪些表。
> SHOW TABLES [LIKE ‘pattern’ | WHERE expr];
>
> 查看全局或会话的系统变量，默认为当前会话。
> SHOW [GLOBAL | SESSION] VARIABLES [LIKE ‘pattern’ | WHERE expr];
>
> 查看各个配置项在各个Server实例上的值。
> SHOW PARAMETERS;
>
> 获得上一条语句执行过程中产生的“Warning”信息，不包含“Error”。
> SHOW WARNINGS [LIMIT [offset,] row_count] SHOW COUNT(*) WARNINGS;
>
> 查看所有用户的当前连接。
> SHOW processlist;

50. **kill 语句**

> 终止线程
>
> KILL [CONNECTION | QUERY] thread_id;

注意事项

• 每个与Cedar的连接都在一个独立的线程里运行，您可以使用SHOW PROCESSLIST;语句查看哪些线程正在运行，并使用KILL thread_id语句终止一个线程。Index列为thread_id。

• KILL CONNECTION与不含修改符的KILL一样：它会终止与给定的thread_id。

• KILL QUERY会终止连接当前正在执行的语句，但是会保持连接的原状。

51. **describe 语句**

> 等同于SHOW COLUMNS FROM
>
> DESCRIBE table_name [col_name | wild];
> 或者
> DESC table_name [col_name | wild];

52. **explain 语句**

> 可以输出SELECT, INSERT, UPDATE, DELETE, REPLACE等DML语句内部的物理执行计划。VERBOSE模式也会输出逻辑执行计划。DBA和开发人员可以根据EXPLAIN的输出来优化SQL语句。
>
> EXPLAIN [VERBOSE] {select_stmt | insert_stmt | update_stmt | delete_stmt | delete_stmt};

53. **when 语句**

> 当WHEN子句的条件满足的时候，才执行WHEN前的主句。
>
> statement WHEN expr|ROW_COUNT(statement) op expr;

应用举例

```
SELECT * FROM t1 WHERE c1 = 1000 FOR UPDATE WHEN ROW_COUNT(UPDATE t2 SET c1 = c1 +1 WHERE c
0 = 1000) >= 0;
```

注意事项

• statement为SELECT、FOR UPDATE、UPDATE 、DELETE、INSERT、REPLACE等语句。

• WHEN子句可以嵌套和组合。

54. **hint 语句**

> hint是一种特殊的SQL注释，SQL注释的一般语法和C语言的注释语法相同，即/* …*/。而hint的语法在此基础上加了一个加号，型如/+ … /。既然是注释，那么如果Server端不认识你SQL语句中的hint，它可以直接忽略而不必报错。这样做的一个好处是，同一条SQL发给不同的数据库产品，不会因为hint引起语法错误。hint只影响数据库服务器端内部优化的逻辑，而不影响SQL语句本身的语义。

目前Cedar中支持的hint如下：

hint	参数	适用语句	含义
READ_STATIC	无	SELECT	只读ChunkServer上的静态数据，数据不保证一致性。如果用户建表的时候加上了READ_STATIC属性，那么即使不使用本hint，用户的SELECT也是只读静态数据的。
HOTSPOT	无	UPDATE	在WHEN连接的复合语句中，表明本UPDATE所更新的行是“热点行”，执行计划在UpdateServer端会根据热点行排队，以减少锁冲突。
INDEX	两个参数，需指定数据表名test和索引创建后的内部表名index_internal_name	SELECT	指定进行查询操作时借助二级索引。
SEMI_JOIN	四个参数，parameter1为小表的表名，parameter2为大表的表名，parameter3为小表的连接列的 列名，parameter4为大表的连接列的列名。	SELECT	指明连接操作的类型为SemiJoin。
JOIN	一个参数，取值可为 bloomfilter_join 或 merge_join	SELECT	指明连接操作的类型是BloomBFilterJoin或MergeMJoin。

注意事项

• Cedar支持的hint有以下几个特点：

• 不带参数的，如/*+ KAKA */。

• 带参数的，如/*+ HAHA(param) */。

• 多个hint可以写到同一个注释中，用逗号分隔，如/*+ KAKA, HAHA(param) */。

• SELECT语句的hint必须近接在关键字SELECT之后，其他词之前。如：SELECT /*+ KAKA */ …。

• UPDATE语句的hint必须紧接在关键字UPDATE之后。

运维及辅助工具

Cedar 0.2 每日合并操作指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 每日合并操作指南	朱燕超	

> Cedar采用基准数据和增量数据分离的存储架构，当UpdateServer中的增量数据达到一定数量的时候，需要对基线数据和增量数据进行融合，即每日合并操作。

每日合并操作如下：

1. 主动发起大版本合并（工具在 cedar/bin 目录下）

```
ups_admin -a ups_ip -p ups_port -t major_freeze
```

- ups_ip：UpdateServer的IP地址。
- ups_port：UpdateServer的服务端口号。

2. 主动发起小版本合并

```
ups_admin -a ups_ip -p ups_port -t minor_freeze
```

- ups_ip：UpdateServer的IP地址。
- ups_port：UpdateServer的服务端口号。

3. 查看合并状态

```
rs_admin -r rs_ip -p rs_port stat -o merge
```

- rs_ip：RootServer的IP地址。
- rs_port：RootServer的端口号。

4. 设置两次升级主版本的最小时间隔（min_major_freeze_interval，单位：秒）。 小于该值，则执行主版本冻结失败。

```
mysql -h ms_ip -P service_port -uadmin -padmin

ALTER SYSTEM SET min_major_freeze_interval = 10 SERVER_TYPE = UPDATSERVER;
```

- ms_ip：任意MergeServer的IP地址。
- service_port：MySQL协议端口。

5. 修改CS 合并开启的延时（当收到新版本数据后，需要等待merge_delay_interval（单位：秒）时间后才开始合并）。

```
mysql -h ms_ip -P service_port -uadmin -padmin

ALTER SYSTEM SET merge_delay_interval = 10 SERVER_TYPE = CHUNKSERVER;
```

- ms_ip：任意MergeServer的IP地址。
- service_port：MySQL协议端口。

6. 修改CS 两次合并最小时间间隔（min_merge_interval，单位：秒）。

```
mysql -h ms_ip -P service_port -uadmin -padmin

ALTER SYSTEM SET min_merge_interval = 10 SERVER_TYPE = CHUNKSERVER;
```

- ms_ip：任意MergeServer的IP地址。
- service_port：MySQL协议端口。

Cedar 0.2 数据导入工具使用指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 数据导入工具使用指南	张燕飞	

文档说明

> 本文档主要介绍如何使用Cedar 0.2自带的数据导入工具Ob_import，利用该工具可以将大量数据从数据文件中高速传输进Cedar数据库中，本文将以实例的形式介绍操作数据导入工具所使用的命令，以及相关的参数设置。

1 案例介绍

基本命令如下：

```
./ob_import -t test_import -dbname test -h 182.119.80.51 -p 62500 -f dat/test_import.dat
```

注意要点：

1. 将数据从dat/test_import.dat文件导入test.test_import表，数据库集群RS的IP地址为182.119.80.51，LMS的端口为62500；
2. 未指定log文件，日志输出到屏幕；
3. 未指定bad文件，不产生脏数据文件；
4. 指定文件路径时，如文件不存在可自动创建，但相关的文件夹应当存在；

包含参数命令如下：

```
 ./ob_import -t test_import -dbname test -h 182.119.80.51 -p 62500 -f dat/import.dat -l imp
ort.log -badfile import.bad -delima 94 --rec-delima 10 -null-flag 64 --record-persql 50 --c
oncurrency 50 --ignore-merge -q 1000 --timeout 10 --buffersize 128
```

```
./ob_import --dbname test -t test_import -h 182.119.80.51  -p 62500 -l import.log --rep --b
adfile import.bad --delima 94 --rec-delima 10 --null-flag 64 -f import.dat  --column-mappin
g "5:v2,['default:)'];v1,3,substr(2,100);v0,2"  --concurrency 1 --ignore-merge
```

```
./ob_import_v2 --dbname test -t test_import -h 182.119.80.51 -p 62500 -l import.log --rep -
-badfile import.bad --record-persql 200 --delima 94 --rec-delima 10 --null-flag 64 -f impor
t.dat --column-mapping "i,0,[-5]; dcm,[43.987]; db,[1.23]; dt,['2015-9-8']; tm,['09:8:14'];
ts,['2015年9月17日8点59分59秒']; v,1,['填充默认值']" --concurrency 1 --ignore-merge
```

注意要点：

1. 可识别的日期格式：

> 1/31/2015-8:00:00　　2000-2-29-0-0-0　　20150826-5-5-5　　2015\8\26 5-5-5　　20150826
> 2015/8/26　　2015-1-2-23-59-59
> 2015年9月16日5点29分30秒　　2月29日2008年23点59分59秒

2. column-mapping参数用法：

> –column-mapping　“3”　　只导入前3列；
> –column-mapping　”na,0; ad,4”　　字段na从第0列导入，ad从第4列导入；
> –column-mapping　“fo,[‘app’]”　　字段fo不从文件导入，直接赋值‘app’；
> –column-mapping　”9:col5,1”　　col5->第1列，col0->第0列，col1->第2列…
> –column-mapping　“dr,5,substr(9,3)“　　字段dr从第5列的子串导入；

2 参数说明

参数	说明	默认值
-t　*tab_name*	数据导入的表名。	必须指定。
–dbname　*db_name*	数据导入表所在的数据库名。	必须指定。
-h　*host*	数据库集群RootServer的IP。	必须指定。
-p　*post*	数据库集群LMS的端口。	必须指定。
-f　*data_file*	指定导入的数据文件。	必须指定。
-l　*log_file*	指定导入过程的日志文件。	建议指定，缺省输出到屏幕。
-q　*queue_size*	缓冲队列的大小。	缺省为1000，不建议修改。
-g　*log_level*	输出日志信息的级别，依次有ERROR、WARN、INFO、TRACE和DEBUG五个级别。	缺省为INFO，不建议修改。
–ins　/　–rep	导入数据的方式，可以为insert方式	缺省ins，不建议修改。

	或replace方式。	
–badfile *bad_file*	指定导入失败的bad文件。	建议指定。
–rowbyrow	每批导入一条记录	缺省为每批导入多条记录，不建议修改。
–concurrency *con_num*	设置导入线程并发数。	缺省为10，建议设置成MS数量*10。
–buffersize *buffer-size*	设置批量导入数据块的大小。	缺省为256KB，不建议修改。
–trim *0/1/2*	清除数据前后空格：0表示清除前后空格，1表示清除前空格，2表示清除后空格。	缺省为不清除空格。
–record-persql *rec-num*	指定每条SQL语句中包含的最大记录个数。	缺省为50，不建议修改。
–ignore-merge	忽略数据库合并状态，强制导入数据。	缺省为合并时暂停导入，不建议修改。
–username *user_name*	指定连接数据库的用户名。	缺省为admin。
–passwd *passwd*	指定连接数据库的密码。	缺省为admin。
–timeout *timeout*	指定数据库超时时间。	缺省为10秒，不建议修改。
–delima *field_ascii*	指定数据文件字段分隔符的ASCII码。	缺省为15（‘^O’的ASCII码）。
–rec-delima *row_ascii*	指定数据文件记录分隔符的ASCII码。	缺省为10（‘\n’的ASCII码）。
–null-flag *null-ascii*	指定用于表示空值的字符的ASCII码;varchar将空字符识别为空字符串，其它类型将其识别为空值。	缺省为空字符。
–column-mapping *“[map_size:][col_0[,file_col_id][,substr(sta,len)][,default_val]]; …”*	指定数据表字段与数据文件中列的对应关系，数据文件中的列从0开始编号。	缺省为按序一一对应。
-timeout *timeout*	指定数据库超时时间。	缺省为10秒，不建议修改。

3 使用Ob_import注意事项

> **1. 使用ob_import_v2导入数据时会占用MS和UPS的大量资源，应避开业务高峰期使用，否则会对业务响应时间产生影响。**
>
> **2. 参数—concurrency用于指定导入并发线程数，需要按照实际集群中的MS数量来确定。**
>
> **3. 参数–record-persql用于指定一条SQL语句导入的记录数，这个参数过大会导致SQL语句长度超出数据库能解析的长度上限。**
>
> **4. 参数—buffersize用于指定一条SQL中包含记录的总长度，其作用与参数–record-persql类似，一条导入SQL语句的长度由这两个参数共同限定，即记录数不会超过–record-persql且记录长度不会超过–buffersize。**
>
> **5. 数据文件末尾的空行会导致导入报错。**

Cedar 0.2 运维工具操作指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 运维工具操作指南	李捷荧	

rs_admin是RootServer的辅助运维工具，提供各种查询和设置的功能。

小窍门：您可以执行“bin/rs_admin -help”命令查询rs_admin的命令和参数。

1.**set_obi_master_first**

> 多集群启动之后，手动设定主集群。

```
bin/rs_admin -r New_Master_ip -p New_Master_Port set_obi_master_first
```

- New_Master_ip：主集群的RootServer所在服务器的IP地址。
- New_Master_Port：主集群的RootServer的端口号。

2.**reelect**

> 重新选举主集群。

```
bin/rs_admin -r Master_RootServer_ip -p Master_RootServer_Port reelect
```

- Master_RootServer_ip：现任主集群RootServer所在服务器的IP地址。
- Master_RootServer_Port：现任主集群RootServer的端口号。

3.**boot_recover**

> 当RootServer端的“first_tablet_meta”丢失时，需要通过该命令来恢复。

```
bin/rs_admin -r RootServer_ip -p RootServer_port boot_recover
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

4.**boot_strap**

> 集群首次启动时初始化操作。包括新建内部表，初始化Schema/权限信息等。

```
bin/rs_admin -r RootServer_ip -p RootServer_port boot_strap
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

5.**change_log_level**

> 调整日志输出级别，设置成DEBUG可以查看更多运行细节，一般用于查问题时使用，默认级别为INFO。

```
bin/rs_admin -r RootServer_ip -p RootServer_port change_log_level -o Log_Level
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- Log_Level：日志级别，有ERROR，WARN，INFO，DEBUG四种。

6.**change_table_id**

> 旁路导入时使用。修改table的table_id。新的table_id也是由系统自动生成的。比如修改3001号表的table_id，新的table_id则为max_table_id。

```
bin/rs_admin -r RootServer_ip -p RootServer_port change_table_id -o table_id=new_id
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- new_id：新的table_id。

7.**change_ups_master**

> 修改集群内的主UpdateServer为指定的UpdateServer。当指定的UPS状态为notsync时，需要加上force参数。

```
bin/rs_admin -r RootServer_ip -p RootServer_port change_ups_master -o ups_ip=UpdateServer_i
p -o ups_port=UpdateServer_port
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- UpdateServer_ip：指定的UpdateServer的ip地址。
- UpdateServer_port：指定的UpdateServer的端口。

8.**check_root_table**

> 当需要下线ChunkServer时，可以通过该命令来检查，如果直接下线该ChunkServer是否会造成数据丢失。

```
bin/rs_admin -r RootServer_ip -p RootServer_port check_root_table -o cs_ip=ChunkServer_ip -
o cs_port=ChunkServer_port
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- ChunkServer_ip：下线的ChunkServer的ip地址。
- ChunkServer_port：下线的ChunkServer的端口。

9.**check_schema**

> 用户在用sql语句更新schema以后，可能会存在schema信息错误，比如修改压缩算法时，压缩算法的名字写错了，写成lze，这种错误在执行sql的时候是不会报错的，只有在每日合并的时候才会报错。所以用户需要主动调用check_schema函数，来检查系统的schema是否合理。

```
bin/rs_admin -r RootServer_ip -p RootServer_port check_schema
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

10.**check_tablet_version**

> 检查所有的在线Tablet是否都已经合并到该版本。

```
bin/rs_admin -r RootServer_ip -p RootServer_port check_tablet_version -o tablet_version=tab
let_version_num
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- tablet_version_num：Tablet的版本。

11.**clean_error_msg**

> RootServer在每日合并过程中如果出现错误时，RootServer会定期报警，提醒DBA来处理，当问题处理完以后，需要将这个报警任务给删除掉。

```
bin/rs_admin -r RootServer_ip -p RootServer_port clean_error_msg
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

12.**clean_root_table**

> 重新生成一份新的RootTable，在以下情况中使用：
>
> - RootTable中存在错误的range或者index。
> - RootTable中Tablet的数目太多，需要清除掉失效的Tablet。

```
bin/rs_admin -r RootServer_ip -p RootServer_port clean_root_table
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

13.**cs_create_table**

> 命令发给ChunkServer，强制在ChunkServer上创建空tablet。一般只有create table失败的情况下，才会使用该手段，作为一种运维补救方法。

```
bin/rs_admin -r ChunkServer_ip -p ChunkServer_port cs_create_table -o table_id=table_id_num
-o table_version=table_version_num
```

- ChunkServer_ip：ChunkServer所在服务器的IP地址。
- ChunkServer_port：ChunkServer的端口号。
- table_id_num：创建的表的id。
- table_version_num：创建的表的version。

14.**do_check_point**

> 执行“check_point”，防止RootServer重启时，需要回放过多的日志。

```
bin/rs_admin -r RootServer_ip -p RootServer_port do_check_point
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

15.**dump_cs_tablet_info**

> 统计ChunkServer上Tablet的个数。

```
bin/rs_admin -r RootServer_ip -p RootServer_port dump_cs_tablet_info -o cs_ip=ChunkServer_i
p -o cs_port=ChunkServer_port
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- ChunkServer_ip：被统计的ChunkServer的IP地址。
- ChunkServer_port：被统计的ChunkServer的端口。

16.**dump_migrate_info**

> 统计ChunkServer的迁移信息，需要在“bin/rs_admin.log”中查看。

```
bin/rs_admin -r RootServer_ip -p RootServer_port dump_migrate_info
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

17.**dump_root_table**

> 按序打印出RootTable的内容，需要在“bin/rs_admin.log”中查看。

```
bin/rs_admin -r RootServer_ip -p RootServer_port dump_root_table
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

18.**dump_server_info**

> 打印ChunkServer的信息，需要在“bin/rs_admin.log”中查看。

```
bin/rs_admin -r RootServer_ip -p RootServer_port dump_server_info
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

19.**dump_unusual_tablets**

> 打印副本或者版本不满足要求的tablet信息，需要在“bin/rs_admin.log”中查看。

```
bin/rs_admin -r RootServer_ip -p RootServer_port dump_unusual_tablets
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

20.**enable_balance | disable_balance**

> 开启或者关闭负载均衡。

```
bin/rs_admin -r RootServer_ip -p RootServer_port enable_balance | disable_balance
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- enable_balance：开启负载均衡。
- disable_balance：关闭负载均衡。

21.**enable_rereplication | disable_rereplication**

> 开启或者关闭复制过程。

```
bin/rs_admin -r RootServer_ip -p RootServer_port enable_rereplication | disable_rereplicati
on
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- enable_rereplication：开启复制过程。
- disable_rereplication：关闭复制过程。

22.**force_create_table**

> 强制新建空tablet。该命令只能在备集群上执行成功，这是为了防止trigger_create_table消息丢失而增加的运维手段。

```
bin/rs_admin -r RootServer_ip -p RootServer_port force_create_table -o table_id=table_id_nu
m
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_id_num：强制建表的id。

23.**force_drop_table**

> 强制drop某张表。该命令只能在备集群上执行成功，这是为了防止trigger_drop_table丢失而增加的运维手段。

```
bin/rs_admin -r RootServer_ip -p RootServer_port force_drop_table -o table_id=table_id_num
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_id_num：强制删表的id。

24.**get_config**

> 打印配置项到屏幕上。

```
bin/rs_admin -r RootServer_ip -p RootServer_port get_config
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

25.**get_obi_config**

> 暂时未使用。

26.**get_obi_role**

> 获取集群的主备角色。

```
bin/rs_admin -r RootServer_ip -p RootServer_port get_obi_role
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

27.**get_row_checksum**

> 暂时未使用。

28.**import**

> 旁路导入命令。增加旁路导入任务。

```
bin/rs_admin -r RootServer_ip -p RootServer_port import table_name table_id uri
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_name：表名。
- table_id：表的ID。
- uri：数据源地址，可是是ip:port格式也可以是某个目录。

29.**import_tablets**

> 旁路导入命令。由旁路导入的脚本来调用，RootServer收到请求后，会要求ChunkServer加载某个table的Tablet。

```
bin/rs_admin -r RootServer_ip -p RootServer_port import_tablets -o table_id=table_id_num
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_id_num：导入的表的id。

30.**kill_import**

> 旁路导入命令，删除一个旁路导入任务。

```
bin/rs_admin -r RootServer_ip -p RootServer_port kill_import table_name table_id
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_name：导入的表的名字。
- table_id：导入的表的id。

31.**log_move_to_debug**

> 将RootServer的日志设置成Debug级别。

```
bin/rs_admin -r RootServer_ip -p RootServer_port log_move_to_debug
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

32.**log_move_to_error**

> 将RootServer的日志设置成Error级别。

```
bin/rs_admin -r RootServer_ip -p RootServer_port log_move_to_error
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

33.**print_root_table**

> 按table_id打印RootTable的内容。

```
bin/rs_admin -r RootServer_ip -p RootServer_port print_root_table -o table_id=table_id_num
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- table_id_num：要打印的表的id。

34.**print_schema**

> 生成schema的可读格式到指定的文件中。

```
bin/rs_admin -r RootServer_ip -p RootServer_port print_schema -o schema_location=output_sch
ema_location
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- output_schema_location：指定文件的位置。

35.**refresh_schema**

> 更新schema。在用户用sql语句的形式修改三张核心表的内容时，需要主动调用一次refresh_schema。RootServer收到命令后，会从内部表中读出最新的schema并缓存起来。

```
bin/rs_admin -r RootServer_ip -p RootServer_port refresh_schema
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

36.**reload_config**

> 修改配置文件后，需要重新加载配置文件。

```
bin/rs_admin -r RootServer_ip -p RootServer_port reload_config
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

37.**restart_cs**

> 重启ChunkServer，命令发出后，ChunkServer会直接退出。

```
bin/rs_admin -r RootServer_ip -p RootServer_port restart_cs -o server_list=<ip1+ip2+...+ip
n>[,cancel]
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- ip1+ip2+…+ipn：重启的ChunkServer的IP地址。

38.**set_config**

> 临时设置配置项的值，只修改了RootServer内存中值，如果RootServer重启的话，则修改会丢失。

```
bin/rs_admin -r RootServer_ip -p RootServer_port set_config -o configuration_parameter=valu
e
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- value：配置项的值。

39.**set_master_ups_config**

> 设置集群内主UpdateServer的流量分配。如果本集群为主集群的话，则集群内主UPS的流量按照master_master_ups_read_percentage的值来定，如果是备集群的话，则集群内主UPS的流量按照read_salve_master_ups_percentage来算，其他UPS则平分剩余的流量。

```
bin/rs_admin -r RootServer_ip -p RootServer_port set_master_ups_config -o master_master_up
s_read_percentage=master_master_ups_read_percentage_value -o slave_master_ups_read_percenta
ge=slave_master_ups_read_percentage_value
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- master_master_ups_read_percentage_value：设置主集群内主UPS的流量
- slave_master_ups_read_percentage_value：设置备集群内主UPS的流量

40.**set_obi_config**

> 暂时未使用。

41.**set_obi_role**

> 设置集群的主备角色，可以设置为“OBI_MASTER”或者“OBI_SLAVE”。

```
bin/rs_admin -r RootServer_ip -p RootServer_port set_obi_role -o OBI_MASTER
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- OBI_MASTER：集群角色。

42.**set_ups_config**

> 设置单个UPS的流量分配。比如设置集群server访问10.232.38.7:5814这台UPS的流量，正常读请求的流量设置为30%，每日合并过程的流量设置成40%。

```
bin/rs_admin -r RootServer_ip -p RootServer_port set_ups_config -o ups_ip=UpdateServer_ip -
o ups_port=UpdateServer_port -o configuration_parameter=value -o configuration_parameter=va
lue ...
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- UpdateServer_ip：UpdateServer的IP地址。
- UpdateServer_port：UpdateServer的端口。
- configuration_parameter：配置项参数的名字。
- value：配置项的值。

43.**shutdown**

> 下线ChunkServer的功能。RootServer会先将该ChunkServer上的tablet全部迁走，迁完以后RootServer会通知该ChunkServer可以下线了。

```
bin/rs_admin -r RootServer_ip -p RootServer_port shutdown -o server_list=<cs_ip1+cs_ip
2+...+cs_ipn>[,cancel]
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。
- cs_ip：ChunkServer的IP地址。

44.**split_tablet**

> 暂时未使用。

45.**stat**

> 查询RootServer的状态值。该命令需要带参数表达式，如表所示。

```
bin/rs_admin -r RootServer_ip -p RootServer_port stat -t request_timeout_us command -o subo
ptions
```

- RootServer_ip：RootServer所在服务器的IP地址。
- RootServer_port：RootServer的端口号。

参数	含义
all_server	打印所有Server的状态信息。
client_conf	暂时未使用
copy_count	统计复制的次数。
cs	打印所有ChunkServer的IP地址和端口。
cs_num	打印ChunkServer的个数。
data_size	暂时未使用。
fail_get_count	打印get请求失败的次数。
fail_scan_count	打印scan请求失败的次数。
frozen_time	打印执行版本冻结的次数和最近一次冻结的时间。
frozen_version	打印UpdateServer汇报的最新版本。
get_obi_role_count	统计执行“get_obi_role”的次数。
local_time	打印RootServer服务器的当前时间。
log_file_id	打印当前记录的RootServer的CommitLog日志文件名。
log_id	打印当前RootServer的CommitLog日志序列号。
mem	打印RootServer服务器的内存使用情况。
merge	打印每日合并的状态。
migrate_count	统计迁移的次数。
ms	打印所有MergeServer的IP地址和端口，包括LMS。

ms_num	打印MergeServer的个数。
ops_get	打印get请求的次数。
ops_scan	打印scan请求的次数。
pid	打印RootServer的进程号。
prog_version	打印RootServer的版本号。
reserve	保留项，暂时未使用。
row_count	暂时未使用。
rs_slave_num	打印备RootServer的个数。
rs_slave	打印备RootServer的信息。如果RootServer为单机时，显示为空。
rs_status	打印RootServer的状态。
schema_version	暂时未使用。
shutdown_cs	下线ChunkServer。RootServer会将该ChunkServer上的tablet全部迁移到其他ChunkServer，迁移完成后，日志通知用户迁移已经完成，可以直接将该ChunkServer下线掉。
sstable_dist	按表统计SSTable在每个ChunkServer上的分布情况。
start_time	打印RootServer的启动时间。
table_count	打印table的个数。
table_num	暂时未使用。
tablet_count	打印Tablet的个数。
tablet_num	打印RootTable的大小，并不表示Tablet的个数。如果需要查看Tablet个数请使用“dump_root_table”命令，然后查看日志。
unusual_tablets_num	打印不正常的Tablet的个数。说明：不正常是指“副本不够或者没有合并到最新版本”。该命令一般用于检查集群Tablet状态。
ups	打印主UpdateServer的租约剩余时间、主备UpdateServer的IP地址、端口、状态等信息。

46.**switch_schema**

> 暂时未使用。

Cedar 0.2 客户端连接指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 客户端连接指南	尹卓超	

> **Cedar客户端与MergeServer通信，目前支持MySQL客户端、Java客户端和C客户端。**

连接类型	说明
Mysql客户端	MergeServer兼容Mysql协议，MySQL客户端及相关工具（如Java数据库访问方式JDBC）只需要将服务器的地址设置为任意一台MergeServer的地址就可以直接使用。
Cedar Java客户端	Cedar内部部署多台MergeServer，Cedar Java客户端提供对MySQL标准JDBC Driver的封装，并提供流量分配、负载均衡、MergeServer异常处理等功能。简单来讲，Cedar Java客户端首先按照一定的策略选择一台MergeServer，接着调用Mysql JDBC Driver往这台MergeServer发送读写请求。Cedar Java客户端实现符合JDBC标准，能够支持Spring、iBatis等Java编程框架。
Cedar C客户端	Cedar C客户端的功能和Cedar Java客户端类似。它首先按照一定的策略选择一台MergeServer，接着调用MySQL标准C客户端往这台MergeServer发送读写请求。Cedar C客户端的接口和Mysql标准C客户端接口完全相同，因此，能够通过LD_PRELOAD的方式将应用程序依赖的Mysql标准C客户端替换为Cedar C客户端，而无需修改应用程序的代码。

1 Cedar Java客户端

连接 Cedar 主要有“使用 OB Configure + oceanbase.jar”、“使用 oceanbase-core.jar”和“直接通过 MergeServer 连接”三种。目前在 taobao、以及 alipay 的 maven 仓库均可搜索获得发布的客户端。如果使用的是 taobao 的仓库，只需要添加相关pom信息。pom 坐标如下：

```
<dependency>
<groupId>com.alipay.oceanbase</groupId>
<artifactId>oceanbase</artifactId>
<version>1.1.1</version>
</dependency>
```

目前 druid 的最新版本为 0.2.25。 pom 坐标如下：

```
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>druid</artifactId>
<version>0.2.25</version>
</dependency>
```

说明：如果使用的是 alipay 的仓库，需要再添加 druid 的 pom 信息。MYSQL 的驱动版本为 5.1.14。 pom 坐标如下：

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.14</version>
</dependency>
```

添加完 pom 信息之后，那么下面可以进行编码工作了。现在，编码工作主要分为以下两种方式：

- 配置编码
  通过调用方法 getConnection()来使用Cedar Java客户端。

```
<bean id="dataSource" class="com.alipay.oceanbase.OceanbaseDataSourceProxy"
init-method="init" destroy-method="destroy" />
<property name="configURL">http://obconsole.test.alibaba-inc.com/ob-config/config.co?dataI
d=financial_history</property>
</bean>
```

- 硬编码

通过直接在程序中添加以下语句使用Cedar Java客户端。

```
OceanbaseDataSourceProxy obDS = new OceanbaseDataSourceProxy();
obDS.setConfigURL("http://obconsole.test.alibaba-inc.com/ob-config/config.co?dataId=financi
al_history");
obDS.init();
Connection conn = obDS.getConnection();
```

使用“OB Configure + oceanbase.jar” 场景访问Cedar

> 当Cedar主备集群已经安装且正常运行，Java开发环境已经部署完成，已经获取Cedar Java客户端Jar 包：“oceanbase-core-1.2.3.jar” 和“druid-0.2.12.jar”。 已经获取依赖包：“commons-lang-2.3.jar”、 “commons-logging-1.1.jar” 和“mysql-connector-java-5.1.14.jar”时即可连接并使用数据库。

使用“oceanbase-core.jar”访问Cedar

> 当Cedar主备集群已经安装且正常运行。OB Config 搭建、配置完成且正常运行。Java 开发环境以及部署完成，以添加“ oceanbase-1.1.1.jar”、“ druid-0.2.25.jar”、 “ mysql-connector-java-5.1.14.jar”的 pom 坐标，或者已获取这三个 jar 包且已经获取“ commons-logging-1.1.jar”和“ commons-lang-2.3.jar” 时即可连接并使用数据库。

使用 MergeServer直接访问Cedar

> 当Cedar主备集群已经安装且正常运行，Java 开发环境已经部署完成（已经安装 JDK、 Eclipse 等），并且已经获取依赖包（ “commons-lang-2.3.jar”、 “commons-logging-1.1.jar” 和“mysql-connector-java-5.1.14.jar”）时即可连接并使用数据库。

2 Cedar C 客户端

Cedar C 客户端主要用于开发人员编写 C 程序连接 Cedar 数据库。

安装

假设本地计算机的用户为 sqluser。安装Cedar C 客户端的操作步骤如下：

1. 以 sqluser 用户登录本地计算机。
2. 执行以下命令， 下载 Cedar C 客户端安装包。

```
wget https://github.com/alibaba/oceanbase_client/blob/master/C/curl-7.29.0-1.el6.x86_64.rpm
wget https://github.com/alibaba/oceanbase_client/blob/master/C/ocean base-devel-0.4.2.1-119
3.el6.x86_64.rpm
```

3. 执行以下命令，安装依赖库 curl。

```
sudo rpm -Uvh curl-7.29.0-1.el6.x86_64.rpm
```

4. 执行以下命令，安装 Cedar C 客户端。

```
sudo rpm -Uvh oceanbase-devel-0.4.2.1-1193.el6.x86_64.rpm
```

5. 使用 vi 编辑器在“~/.bashrc”文件中添加以下内容。

```
export LD_PRELOAD=/home/admin/cedar/lib/libobsql.so.0.0.0
export OB_SQL_CONFIG_DIR=/home/admin/cedar/etc/
```

6. 执行以下命令，使环境变量生效。

```
source ~/.bashrc
```

7. 使用 vi 编辑器修改“~/cedar/etc/libobsql.conf”文件，

```
logfile=/tmp/obsql.log
#initurl=http://10.232.102.182:8080/diamond-server/config.co?dataId=197 loglevel=DEBUG
minconn=2
maxconn=50
ip=10.10.10.2
port=2828
username=admin
passwd=admin
```

访问Cedar

当 Cedar 主备集群已经安装且正常运行，配置“__all_cluster”中主备集群的流量分配“cluster_flow_percent”，主集群为“0”，备集群为“100”，C 开发环境已经部署完成，且成功安装C客户端后，可连接并使用数据库。

Java/C客户端访问Cedar的流程如下：

> 1. Cedar 客户端请求RootServer获取集群中MergeServer的地址列表。
> 2. 按照一定的策略选择一台MergeServer发送读写请求。 说明：客户端支持的策略主要有两种：随机以及一致性哈希。一致性哈希的主要目的是将相同的SQL请求发送到同一台MergeServer，方便MergeServer对查询结果进行缓存。
> 3. 如果请求MergeServer失败，则从MergeServer列表中重新选择一台MergeServer。当请求某台MergeServer失败超过一定的次数时，则将这台MergeServer加入黑名单并从MergeServer列表中删除。另外，客户端会定期请求RootServer更新MergeServer地址列表。

Cedar 0.2 日志参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 日志参考指南	王苏翔	

1 日志概述

> 主要介绍日志清单、日志级别、查看日志的方法和收集日志的方法等。

1.1 日志清单

表 1 列出了 Cedar 运行日志和操作日志的清单。

表 1 日志清单：

服务类型	日志名称	日志路径	说明
RootServer	rootserver.log	RootServer所在Cedar服务器的“~/cedar/log”目录下。	记录RootServer启动后的运行情况。
RootServer	1、 2、 3、…	RootServer所在Cedar服务器的“~/cedar/data/rs_commitlog”目录下。	RootServer的commit 日志，记录RootServer的Tablet信息和schema信息。
UpdateServer	updateserver.log	UpdateServer所在Cedar服务器的“~/cedar/log”目录下。	记录UpdateServer启动过程以及启动后的运行情况。
MergeServer	mergeserver.log	MergeServer所在Cedar服务器的“~/cedar/log”目录下。	记录MergeServer启动过程以及启动后的运行情况。
ChunkServer	chunkserver.log	所在Cedar服务器的“~/cedar/log”目录下。	记录ChunkServer启动过程以及启动后的运行情况。
/	rs_admin.log	执行 rs_admin 命令的Cedar服务器的“~/cedar”目录下。	记录bin/rs_admin 的执行日志。

1.2 日志级别

介绍 Cedar日志的级别及每类级别的含义。当启动或运行异常时，运维人员可以通过日志级别和内容分析和定位异常原因。
Cedar运行日志中的级别及含义如表 2 所示。日志级别从高到底排列。

表 2 日志级别及含义：

日志级别	含义
ERROR	严重错误，用于记录系统的故障信息，必须进行故障排除，否则系统不可用。
WARN	警告，用于记录可能会出现的潜在错误。
INFO	提示，用于记录系统运行的当前状态，该信息为正常信息。
DEBUG	调试信息，用于调试时更详细的了解系统运行状态，包括当前调用的函数名、参数、变量、函数调用返回值等。

1.3 查看日志

介绍了查看Cedar日志的方法。本文以查看 RootServer 的系统日志为例，介绍查看Cedar日志的方法。假设 RootServer 所在的Cedar服务器 IP 为“10.10.10.2”，查看日志的步骤如下：

1. 登录 Cedar服务器（ 10.10.10.2 ）。
2. 执行以下命令，进入日志文件所在的目录。

```
cd ~/cedar/log
```

3. 执行以下命令，查看已写入的日志记录。

```
more rootserver.log
```

或者执行以下命令，查看正在写入的日志记录。

```
tail rootserver.log
```

1.4 收集日志

介绍了收集 Cedar日志的方法。建议日志收集命名格式为：“日志名称_收集日期.tar”。本文以收集 RootServer 和 UpdateServer 日志为例，介绍收集 Cedar日志的方法。假设 RootServer 和 UpdateServer 所在的 Cedar 服务器 IP 为“10.10.10.2”，收集Cedar日志的操作步骤如下：

1. 登录 Cedar服务器（ 10.10.10.2 ）。
2. 执行以下命令，进入日志文件所在的目录。

```
cd ~/cedar/log
```

3. 执行以下命令，收集日志。

```
tar cvf rs_and_ups_130809.tar rootserver.log updateserver.log
```

说明： 如果需要收集当前目录下的所有日志，可执行”tar cvf 日志名称.tar *命令。

2 启动和运行日志

> 记录 Cedar 各 Server 启动过程以及启动后的运行情况。这是我们在日常工作中主要查看的日志。

2.1 日志说明

Cedar 启动和运行日志的名称、路径以及说明如表 3 所示。

表 3 日志说明：

服务类型	日志名称	日志路径	说明
RootServer	rootserver.log	RootServer所在Cedar服务器的“~/cedar/log”目录下。	记录RootServer启动过程以及启动后的运行情况。
UpdateServer	updateserver.log	UpdateServer所在Cedar服务器的“~/cedar/log”目录下。	记录UpdateServer启动过程以及启动后的运行情况。
MergeServer	mergeserver.log	MergeServer所在Cedar服务器的“~/cedar/log”目录下。	记录MergeServer启动过程以及启动后的运行情况。
ChunkServer	chunkserver.log	ChunkServer所在Cedar服务器的“~/cedar/log”目录下。	记录ChunkServer启动过程以及启动后的运行情况。

2.2 日志格式

> 日志记录主要由五部分组成：记录时间、日志级别、文件名:行号、线程 ID 和日志内容。

日志样例

```
[2013-08-09 10:02:18.339314] INFO ob_root_worker.cpp:3159 [139702581581568] start to boot s
trap
解释：Cedar集群进行“boot srtap”初始化。
```

3 commit 日志

记录 RootServer 的 Tablet 信息和 schema 信息和 UpdateServer 的 mutator 信息。UpdateServer 每次进行更新操作时，均会产生一条 commit log，并追加到已有的 commit log 序列之后。 UpdateServer 通过同步 commit log 实现主备复制。Cedar的 commit 日志的名称、路径和说明如表 4 所示。

表 4 日志说明：

服务类型	日志名称	日志路径	说明
RootServer	1、2、3、…	RootServer所在Cedar服务器的“~/cedar/data/rs_commitlog”目录下。	RootServer的commit日志，记录RootServer的Tablet信息和schema信息。
UpdateServer	1、2、3、…	UpdateServer所在Cedar服务器的“~/cedar/data/ups_commitlog”目录下。	UpdateServer的commit日志，记录UpdateServer的mutator信息。

4 rs_admin 日志

记录在初始化Cedar过程中运行rs_admin的日志。Cedar的rs_admin日志的名称、路径和说明如表 5 所示。

表 5 日志说明：

日志名称	日志路径	说明
rs_admin.log	执行 rs_admin 命令的Cedar服务器的“~/cedar”目录下。	记录 bin/rs_admin 的执行日志。日志记录主要由五部分组成：记录时间、日志级别、文件名:行号、线程 ID 和日志内容。

日志样例

```
[2013-08-09 10:02:18.338467] INFO ob_base_client.cpp:70 [140524464404096] start io thread
解释：启动 io 线程。
```

系统参数及配置项参考

Cedar 0.2 内部表参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 内部表参考指南	尹卓超 隆飞	

1. **__first_tablet_entry**

> 记录了集群中所有table的基本属性信息。
> Rowkey: (table_name)。
> “_first_tablet_entry”参数说明如表 1 所示。

表 1 _first_tablet_entry表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
table_name	varchar(256)	表名。
creat_time_column_id create_time	int	列的列 id。
modify_time_column_id	int	modify_time 列的列 id。
table_id	int	表ID。
table_type	int	1：普通表。2：索引。3：元数据表。4：view。5：临时表。
load_type	int	1：保存到磁盘。2：保存到内存。
table_def_type	int	1：内部表。2：用户定义表。
rowkey_column_num	int	主键的列数，后续endrowkeyobj1, endrowkeyobj2…等来依次表示主键的列。
column_num	int	全部的列数(包括主键)。
max_used_column_id	int	该表使用过的最大列 ID(列 ID 不重用)。
replica_num	int	单个集群的 Tablet 的 replica 的个数(1~6)。
create_mem_version	int	新建该表时候系统的mem_version，暂时保留。
tablet_max_size	int	该表每个 Tablet 的 SSTable 文件最大允许大小。
max_rowkey_length	int	Rowkey的最大长度限制。
compress_func_name	varchar(256)	存储 SSTable 所使用的压缩方法名称。
is_use_bloomfilter	int	指定是否使用 bloomfilter。
merge_write_sstable_versio	int	合并的时候写哪个版本的 SSTable
is_pure_update_table	int	指定是否属于内存更新表。
rowkey_split	int	用于指定每日合并中 Tablet 的分裂点为 rowkey 的第几个 obj。
expire_condition	varchar(512)	使用表达式定义的此表的数据自动过期删除条件。
tablet_block_size	int	Tablet_block 的大小。
is_read_static	int	是否要读静态数据。
schema_version	int	schema 版本。

2. **__all_all_column**

> **“all_all_column”存储了每个表的所有列、 column_id、列类型等，包括内部表(不包括核心表)和用户定义表。与内部表“**all_join_info”共同定义了各个表的 schema 信息。
> Rowkey： (table_id， column_name)。
> “__all_all_column”参数说明如表 2 所示。

表 2 __all_all_column表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
table_id	int	表 ID。
column_name	varchar(128)	列名。
table_name	varchar(256)	表名。
column_id	int	列 ID。
column_group_id	int	列隶属的 column group id。
rowkey_id	int	0：非 rowkey。 正整数： rowkey 的序号，必须是从 1开始的连续正整。说明：“__all_table_table”中的“ rowkey_column_num”定义了该表的 rowkey 的列数量。
length_in_rowkey	int	如果是 rowkey 列，表示在二进制 rowkey 串中占用的字节数。
order_in_rowkey	int	表示该列的升降序。
join_table_id	int	-1：没有 join。 正整数：连接表的表 ID。
join_column_id	int	-1：没有 join。 正整数：连接表中的列 ID。
data_type	int	数据类型。
data_length	int	整数的字节数或字符串的最大长度。
data_precision	int	整数的十进制位数或 decimal 的有效位数(小数点前和小数点后)。
data_scale	int	decimal 小数点后的位数。
nullable	int	1：不可以为空；2：可以为空。

3. **__all_join_info**

> “__all_join_info”存储了表之间的内部 join 关系，即左表通过其某些列对应到右表的 rowkey。
> 说明： 左表及右表的对应列的类型必须一致。
> Rowkey： (left_table_id, left_column_id, right_table_id, right_column_id)“__all_join_info”参数说明如表 3-3 所示。
> “__all_join_info”参数说明如表 3 所示。

表 3 __all_join_info表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。

left_table_id	int	左表的表 ID。
left_column_id	int	左表的列 ID。
right_table_id	int	右表的表 ID。
right_column_id	int	右表的列 ID。
left_table_name	varchar(256)	左表的表名。
left_column_name	varchar(256)	左表的列名。
right_table_name	varchar(256)	右表的表名。
right_column_name	varchar(256)	右表的列名。

4. **__all_client**

> “__all_client”用来保存 JAVA 客户端的版本信息。
> Rowkey：(client_ip, version)。
> “__all_client”参数说明如表 4 所示。

表 4 __all_client表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
client_ip	varchar(32)	0：与特定集群无关；正整数：指定集群。
version	varchar(16)	客户端版本。
status	varchar(32)	客户端状态。
extra1	varchar(32)	预留。
extra2	int	预留。

5. **__all_cluster**

> “__all_cluster”记录了系统中所有的集群，这个表由每个集群的主 RootServer更新。
> Rowkey：(cluster_id)
> “__all_cluster”参数说明如表 5 所示。

表 5 __all_cluster表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	集群 ID，正整数。
cluster_vip	varchar(32)	Cluster 的 IP 地址。
cluster_port	int	Cluster 端口号。

cluster_role	int	1： Master 2： Slave
cluster_name	varchar(128)	集群名称。
cluster_info	varchar(128)	集群说明信息。
cluster_flow_percent	int	流量配比。
read_strategy	int	客户端使用的负载均衡策略： 0：随机轮转策略。1：一致性哈希。 rootserver_port int RootServer 服务端口。

6. **__all_server**

> “__all_server”记录了系统中所有的服务器，这个表仅仅由主集群的主RootServer 更新。
> Rowkey： (cluster_id， svr_type, svr_ip, svr_port)
> “__all_server”参数说明如表 6 所示。

表 6 __all_server表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	0：与特定集群无关。正整数：指定的集群 ID。
svr_type	varchar(16)	RootServer ChunkServer MergeServerUpdateServer Other
svr_ip	varchar(32)	Server IP 地址。
svr_port	int	Server 端口。
inner_port	int	内部交互端口。
svr_role	int	0：与服务器角色(RS 或 UPS 的主或备)无关。1： slave2： master
svr_version	varchar(64)	程序版本信息。

7. **__all_server_session**

> “__all_server_session”用于记录当前连接 Cedar 的信息。
> Rowkey： (id)
> “__all_server_session”参数说明如表 7 所示。

表 7 __all_server_session表参数：

参数	类型	说明
id	int	连接 Cedar 的 ID。
username	varchar(512)	连接 Cedar 的用户名。
host	varchar(128)	连接到 Cedar 的客户端的 IP 和 Port。
db	varchar(128)	数据库名，目前只有 Cedar。
command	varchar(1024)	执行的命令。
timeelapse	int	正在执行的 SQL 的耗时。单位：微秒。

state	varchar(128)	是否正在使用： ACTIVE：正在使用。 SLEEP：暂时未使用。
info	varchar(128)	连接说明信息。
mergeserver	varchar(128)	连接的 MergeServer。
index	int	连接 Cedar 的 thread id。

8. **__all_server_stat**

> “__all_server_stat”用于记录本集群内所有服务器的监控信息，属于虚拟的内存表，不对监控数据进行存储。
> Rowkey： (svr_type, svt_ip, svr_port, name)
> “__all_server_stat”参数说明如表 8 所示。

表 8 __all_server_stat表参数：

参数	类型	说明
svr_type	varchar(16)	RootServer ChunkServer MergeServer UpdateServer Other
svr_ip	varchar(32)	Server IP 地址。
svr_port	int	Server 端口。
name	varchar(64)	监控项的名称。
value	int	监控项的值。

9. **__all_statement**

> “__all_statement”用于记录全局缓存的 PREPARE 语句。
> Rowkey： (svr_ip, svr_port, statement)
> “__all_statement”参数说明如表 9 所示。

表 9 __all_statement表参数：

参数	类型	说明
svr_ip	varchar(32)	创建语句的 IP。
svr_port	int	创建语句的端口号。
statement	varchar(1024)	语句文本。
id	int	内部标识的唯一 id。
prepare_count	int	执行 PREPARE 的次数。
execute_count	int	执行 EXECUTE 的次数。
avg_execute_usec	int	平均执行时间，单位：微秒。
slow_count	int	慢查询的次数。
create_time	timestamp	创建的时间。
last_active_time	timestamp	最后一次使用的时间。

10. **_all_sys_config**

> “__all_sys_config”存储了 Server 所需的配置项参数。
> Rowkey：（cluster_id， svr_type， svr_ip, svr_ port, name)
> “__all_sys_config”参数说明如表 10 所示。

表 10 __all_sys_config表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	0：与特定集群无关。 正整数：指定集群。
svr_type	varchar(16)	RootServer ChunkServer MergeServer UpdateServer Other
svr_ip	varchar(32)	Server IP 地址。
svr_port	int	Server 端口。
name	varchar(256)	参数名称。
section	varchar(256)	参数所属的段。
data_type	varchar(256)	参数值的数据类型。
value	varchar(256)	参数值。
value_strict	varchar(256)	参数值的约束。
info	varchar(256)	对该项的说明。

11. **__all_sys_config_stat**

> “**all_sys_config_stat”用于显示当前各个 Server 已经生效的配置项参数值，它的 Schema 与**“all_sys_config”的相同。
> Rowkey：（cluster_id， svr_type， svr_ip, svr_ port, name)
> “__all_sys_config_stat”参数说明如表 11 所示。

表 11 __all_sys_config_stat表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	0：与特定集群无关。 正整数：指定集群。
svr_type	varchar(16)	RootServer ChunkServer MergeServer UpdateServer Other
svr_ip	varchar(32)	Server IP 地址。
svr_port	int	Server 端口。
name	varchar(256)	参数名称。
section	varchar(256)	参数所属的段。
data_type	varchar(256)	参数值的数据类型。

value	varchar(256)	参数值。
value_strict	varchar(256)	参数值的约束。
info	varchar(256)	对该项的说明。

12. **__all_sys_param**

> “__all_sys_param”存储了系统所需的诸多参数，如环境变量等，不同的参数保存在不同行。
> Rowkey：(cluster_id,name)
> “__all_sys_param”参数说明如表 12 所示。

表 12 __all_sys_param表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	0：与特定集群无关。 正整数：指定的集群 ID。
name	varchar(256)	参数名称。
data_type	int	参数值的数据类型。 0： null 1： int 2： float 3： double 4： datetime 5： precisedate，与 datetime 相同，精确到毫秒。 6： varchar 8： createtime 9： modifytime 11： bool 12： decimal。
value	varchar(256)	参数值。
info	varchar(256)	对该项的说明。

13. **__all_sys_stat**

> “__all_sys_stat”存储了系统各种状态值，不同的项保存在不同行。
> Rowkey：(cluster_id， name)
> “__all_sys_stat”参数说明如表 13 所示。

表 13 __all_sys_stat表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
cluster_id	int	0：与特定集群无关。 正整数：指定集群。
name	varchar(256)	参数名称。
data_type	int	数据类型。 0： null 1： int 2： float 3： double 4： datetime 5： precisedate，与 datetime 相同，精确到毫秒。 6： vchar 8： createtime 9： modifytime 11： bool 12： decimal
value	varchar(256)	参数值。
info	varchar(256)	对参数的说明。

14. **__all_table_privilege**

> “__all_table_privilege”记录了系统中用户在每个表的读写等权限。
> Rowkey：(user_id， table_id)
> “__all_table_privilege”参数说明如表 14 所示。

表 14 __all_table_privilege表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
user_id	int	用户内部 ID。
table_id	int	表 ID， table_id = 0 时，表示 all_table。
priv_all	int	是否有所有权限。
priv_alter	int	是否有 alter table 权限。
priv_create	int	是否有 create table 权限。
priv_create_user	int	是否有 create user 权限。
priv_delete	int	是否有 delete table 权限。
priv_drop	int	是否有 drop table 权限。
priv_grant_option	int	是否有 grant 授权权限。
priv_insert	int	是否有 insert 权限。
priv_update	int	是否有 update 权限。
priv_select	int	是否有 select 权限。
priv_replace	int	是否有 replace 权限。

15. **__all_trigger_event**

> “__all_trigger_event”用于记录内部通知事件。
> Rowkey：(event_ts)
> “__all_trigger_event”参数说明如表 15 所示。

表 15 __all_trigger_event表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
event_ts	PrecisDateTime	事件发生时间戳。
src_ip	varchar	事件发生源机器 ip。
event_type	int	事件类型。
event_param	int	消息参数。
extra	varchar	预留。

16. **__all_user**

> “__all_user”记录了系统中所有的可以登录 Cedar 的用户，每个用户一行记录。
> Rowkey：(user_name)
> “__all_user”参数说如表 16 所示。

表 16__all_user表参数：

参数	类型	说明
gm_create	createtime	创建时间。
gm_modify	modifytime	修改时间。
user_name	varchar	用户名。
user_id	int	用户内部 ID。
pass_word varchar 用户密码（密文存储）。		
info	varchar	注释。
priv_all	int	是否拥有所有的权限。
priv_alter	int	是否有 alter 权限。
priv_create	int	是否有 create table 权限。
priv_create_user	int	是否有 create user 权限。
priv_delete	int	是否有 delete table 权限。
priv_drop	int	是否有 drop table 权限。
priv_grant_option	int	是否有 grant 授权权限。
priv_insert	int	是否有 insert 权限。
priv_update	int	是否有 update 权限。
priv_select	int	是否有 select 权限。
priv_replace	int	是否 replace 权限。
is_locked	int	是否被锁。

17. **__all_secondary_index**

> 记录了集群中创建的二级索引表的信息。
> Rowkey: (table_name)。
> 此内部表中字段信息与__first_tablet_entry大致相同， 额外有两个字段用于记录索引表信息，如表 17 所示。

表 17 __all_secondary_index表参数：

参数	类型	说明
original_table_id	int	索引表对应数据表的表id。
index_status	int	索引表的状态信息。

Cedar 0.2 配置项参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 配置项参考指南	徐石磊	

1 RootServer配置参数

参数	默认值	说明
balance_max_concurrent_migrate_num	2	单个ChunkServer上迁移SSTable时，允许的最大的迁移数。取值范围：[1,10]
balance_max_migrate_in_per_cs	20	一批迁移SSTable组成一个任务，该值表示单个ChunkServer上允许迁入的最大的迁移任务数。取值范围：[1,100]
balance_max_migrate_out_per_cs	20	一批迁移SSTable组成一个任务，该值表示单个ChunkServer上允许的最大的迁移任务数。取值范围：[1,100]
balance_max_timeout	5m	迁移任务的超时时间。不建议修改。
balance_timeout_delta	10s	迁移任务超时时间允许的容错时间，即超时时间达到“balance_max_timeout + balance_timeout_delta”后，才判定任务失败。不建议修改。
balance_tolerance_count	10	对单个表来说，可能保存有一个或多个SSTable，这些SSTable分布在各个ChunkServer数量在以下区间内：[SSTable数/ChunkServer总数 - balance_tolerance_count, SSTable数/ChunkServer总数 + alance_tolerance_count]取值范围：[1,1000]推荐值：10
balance_worker_idl_time	30s	检查所有表的SSTable是否均衡分布在ChunkServer上的时间间隔。不建议修改。
build_root_table_time	5m	当ChunkServer和RootServer中的数据不一致时，需要Clean RootTable进行重新构建。该参数表示重新构建RootTable的时间。
cluster_id	-	Cedar集群ID。
commit_log_dir	data/rs_commitlog	Cedar安装目录下RootServer的CommitLog目录。
commit_log_sync_type	1	RootServer的CommitLog同步类型。0：用于CommitLog的内存缓冲区写满后再刷入CommitLog文件内。1：在用于CommitLog的内存缓冲区内，每写一条日志就刷入CommitLog文件里。
cs_lease_duration_time	10s	ChunkServer的租约有效期。RootServer给

		ChunkServer发送一个租约，如果在该有效期时间内ChunkServer无应答，则判定ChunkServer不在线。不建议修改。
cs_probation_period	5s	新ChunkServer上线后再该时间段内不允许tablet迁入，用于防止ChunkServer上线后马上下线的情况。
ddl_system_table_switch	False	是否允许操作内部表的开关。True：开启。False：不开启。
devname	eth0	启动RootServer服务的网卡名称。
enable_balance	True	是否开启ChunkServer上的迁移SSTable任务均衡。True：开启。False：不开启。
enable_cache_schema	True	RootServer会从UpdateServer和ChunkServer中读取Schema，该参数表示在RootServer内是否保存Schema的缓存。True：是。False：否。
enable_load_data	False	是否开启允许外部数据进行旁路导入。True：开启。False：不开启。
enable_rereplication	True	是否开启Tablet的副本复制。如果不开启，则即使Tablet的副本数小于设置的“tablet_replicas_num”，也不会进行复制。True：开启。False：不开启。
expected_request_process_time	10ms	在RootServer和其他Server进行网络交互时，其他Server会给RootServer发送包，如果“RootServer收到包到处理包的时间间隔 + 该时间 > 其他Server要求响应的时间”，则RootServer放弃该包。
first_meta_filename	first_tablet_meta	用来标示是否已经执行“boot_strap”命令。
inner_table_network_timeout	50s	访问内部表的超时时间。
io_thread_count	4	用于Libeay的I/O线程数。需要重新启动RootServer服务才能生效。取值范围：[1,100]
is_ups_flow_control	False	主备UpdateServer是否按照流量分配进行读服务。True：主备UpdateServer的读服务按流量分配。False：主备UpdateServer的读服务按主备各50%分配。
lease_interval_time	15s	主RootServer给备RootServer发送一个租约的有效期。不建议修改。
lease_on	True	主备RootServer间是否需要开启租约模式。True：开启。False：不开启。
lease_reserved_time	10s	主RootServer给备RootServer发送一个租约后，如果在该时间内，备RootServer对主RootServer无应答，则判定备RootServer不在线。不建议修改。
load_data_max_timeout_per_range	3m	RootServer发起Tablet迁移时，Tablet按range从ChunkServer1迁移到ChunkServer2。该参数表示ChunkServer2向RootServer汇报的超时时间。
load_sstable_time	20m	ChunkServer加载SSTable的超时时间。
log_queue_size	100	主RootServer向备RootServer同步CommitLog时，同步任务会先进入备RootServer的日志任务队列，然后

		在备RootServer空闲时进行处理。该参数表示备RootServer中的日志任务队列存放的对大任务数 。取值范围：[10,100000]
log_replay_wait_time	100ms	备RootServer中，日志回放线程读取CommitLog的时间间隔。不建议修改。
log_sync_limit	40MB	备RootServer启动时，先取主RootServer中的日志。该参数表示取CommitLog的带宽上限。
log_sync_timeout	500ms	主往备同步日志的超时时间。在该时间内备RootServer需对主RootServer进行应答，否侧日志同步失败。不建议修改。
master_root_server_ip	10.10.10.2	Cedar主集群RootServer的VIP。
master_root_server_port	-	Cedar主集群的RootServer端口。
max_commit_log_size	64MB	RootServer中每个CommitLog文件大小的最大值。当文件大小达到该值之后，则生成下一个CommitLog文件。文件名按“1、2、3、4…”依次生成。不建议修改。
max_merge_duration_time	2h	数据合并开始到数据合并结束的最大允许时间，超过该时间则合并失败。
migrate_wait_time	60s	负载均衡线程启动在且等待该时间后才开始工作。不建议修改。
monitor_drop_table_interval	600s	用户在DROP TABLE时，Cedar内部不会立即删除该表，而是在每日合并时删除。该参数表示删除表的时间间隔。
monitor_row_checksum	True	备集群开启行一致性校验的开关。
monitor_row_checksum_interval	1800s	备集群进行行一致性校验的周期。
monitor_row_checksum_timeout	3s	备集群向主集群取校验码的超时时间。
network_timeout	50s	RootServer与其他Server进行网络互交的超时时间。
port	2500	RootServer的服务端口。取值范围：(1024, 65535)
read_master_master_ups_percent	100	Cedar主集群中主UpdateServer的读服务百分比，而主集群中备UpdateServer的读服务百分比为“（1-主UpdateServer的百分比）/备UpdateServer数量”。取值范围：[0,100]推荐值：40
read_queue_size	500	RootServer处理读请求任务的队列大小。取值范围：[10,100000]推荐值：10000
read_slave_master_ups_percent	100	Cedar备集群中主UpdateServer的读服务百分比，而备集群中备UpdateServer的读服务百分比为“（1-主UpdateServer的百分比）/备UpdateServer数量”。取值范围：[0,100]推荐值：50
read_thread_count	20	RootServer处理读任务的线程数。取值范围：[10,100]
report_tablet_time	5m	Clean RootTable时，ChunkServer向RootServer汇报的超时时间。
retry_times	3	RootServer与其他Server网络交互失败时的重试次数。

root_server_ip	-	RootServer的IP。
rs_data_dir	data/rs	Cedar安装目录下RootServer的数据目录。
safe_lost_one_time	3600s	当ChunkServer下线，造成缺少一个Tablet副本时，进行重新复制的等待时间。不建议修改。
safe_wait_init_time	60s	Clean RootTable时，有一个线程检查新的RootTable是否构建完成的时间间隔。
schema_filename	etc/schema.ini	Cedar安装目录下Schame文件的路径和名称。
slave_register_timeout	3s	备RootServer启动后，向主RootServer进行注册的超时时间。不建议修改。
tablet_migrate_disabling_period	60s	刚经过迁移的Tablet或者刚上线的ChunkServer中的Tablet时，需要经过这段时间才可以被迁移。不建议修改。
tablet_replicas_num	3	Tablet副本数。
ups_lease_reserved_time	8500ms	更新租约的保留时间。在“ups_lease_time - ups_lease_reserved_time”时间内，RootServer向主UpdateServer重新发送租约。不建议修改。
ups_lease_time	9s	RootServer给主UpdateServer发送的租约的有效时长。
ups_renew_reserved_time	7770ms	“ups_lease_time - ups_renew_reserved_time”时间内，主UpdateServer未收到RootServer发送的租约时，主UpdateServer会主动向RootServer发送更新租约请求。不建议修改。
ups_waiting_register_time	15s	RootServer需要在第一个UpdateServer进行注册，并再等待该参数设置的时间后，才进行选主UpdateServer。
vip_check_period	500ms	主备RootServer检查VIP是否在自己所在服务器上的时间间隔。不建议修改。
write_queue_size	100	RootServer处理写请求任务的队列大小。取值范围：[10,100000]

2 UpdateServer配置参数

参数	默认值	说明
active_mem_limit	系统自动生成	用户的更新操作写入Active MemTable。当Active MemTable大小到达该值后，则冻结Active MemTable，同时开启新的Active MemTable接受更新操作。推荐值和计算方法如下：当app_mod=import时的计算方法：table_memory_limit/minor_num_limit*0.7；当app_mod=oltp时的计算方法：“table_mem_limit”减去为冻结表预留内存的大小。预留大小为table_mem_limit的10%，但最大为10G。
blockcache_size	系统自动生成	Block缓存大小。该参数不可以动态改小，但是可以动态改大。需要重新启动UpdateServer服务才能生效。

		推荐值：系统自动生成
blockindex_cache_size	系统自动生成	Block索引缓存大小。该参数不可以动态改小，但是可以动态改大。需要重新启动UpdateServer服务才能生效。推荐值：系统自动生成
commit_bind_core_id	-1	并发事务提交线程所绑定的CPU id。
commit_end_thread_num	4	CommitEndHandlePool线程池中线程数量，用于给客户端回包以及销毁一次请求所用的数据结构。
commit_log_dir	data/ups_commitlog	Cedar安装目录下UpdateServer的CommitLog目录。需要重新启动UpdateServer服务才能生效。
commit_log_size	64MB	UpdateServer中每个CommitLog文件大小的最大值。当文件大小达到该值之后，则生成下一个CommitLog文件。文件名按“1、2、3、4…”依次生成。需要重新启动UpdateServer服务才能生效。不建议修改。
consistency_type	2	一致性SQL请求只能读取主UpdateServer的CommitLog；弱一致性SQL请求读取UpdateServer中CommitLog时，需要根据该参数配置的一致性类型进行读取CommitLog。1：Strong，只能读取主UpdateServer的CommitLog。2：Normal，只有当主备UpdateServer同步时，才允许读取备UpdateServer的CommitLog；否则只能读取主UpdateServer的CommitLog。3：Weak，可以读取主或备UpdateServer的CommitLog。推荐值：3
devname	eth0	启动UpdateServer服务的网卡名称。需要重启UpdateServer服务才能生效。
dir_regex	^store[0-9]+$	raid目录下指向磁盘实际目录的软链接的名称匹配式。需要重新启动UpdateServer服务才能生效。不建议修改。
disk_delay_warn_param	40ms;800us;10;100000	写CommitLog的超过该时间。如果超时，则产生一条Warn日志。
fetch_log_wait_time	500ms	备UpdateServer从主UpdateServer两次读取CommitLog的时间间隔。不建议修改。
fetch_schema_timeout	3s	UpdateServer从RootServer获取schema的超时时间。需要重新启动UpdateServer服务才能生效。不建议修改。
fetch_schema_times	10	UpdateServer从RootServer获取schema失败时的重试次数。不建议修改。
high_prio_quota_percent	50	对于工作线程中高优先级的CPU计算资源配给比例。
inner_port	2701	用于每日合并的端口号。需要重新启动UpdateServer服务才能生效。不建议修改。取值范围：(1024,65536)
io_thread_count	3	用于Libeay的I/O线程数。需要重新启动UpdateServer服务才能生效。
io_thread_end_cpu	-1	与io_thread_start_cpu一起，表示读写线程绑定的CPU范围。例如io_thread_start_cpu = 1，io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于处理读写请求。

io_thread_start_cpu	-1	与io_thread_end_cpu一起，表示读写线程绑定的CPU的范围。例如io_thread_start_cpu = 1，io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于处理读写请求。
keep_alive_interval	500ms	控制UpdateServer向RootServer发送心跳信息的最小间隔时间。
keep_alive_reregister_timeout	800ms	备UpdateServer超过此时间都未收到主UpdateServer日志的情况下，则需要重新向主UpdateServer注册。
keep_alive_timeout	5s	备UpdateServer未收到主UpdateServer的消息超过该值时，则重新向主UpdateServer注册。需要重新启动UpdateServer服务才能生效。不建议修改。
lease_queue_size	100	UpdateServer从RootServer获取租约的任务队列长度。需要重新启动UpdateServer服务才能生效。
lease_timeout_in_advance	500ms	在租约有效期内，且该租约经过“ups_lease_time - lease_timeout_in_advance”的时间后，如果主UpdateServer还没有收到RootServer发送的新租约，则主UpdateServer将不再接受写事务。需要重新启动UpdateServer服务才能生效。不建议修改。
log_cache_block_size	32MB	用于备UpdateServer接收主UpdateServer的CommitLog的缓存块大小。需要重新启动UpdateServer服务才能生效。
log_cache_n_block	4	用于备UpdateServer接收主UpdateServer的CommitLog的缓存块的数量。需要重新启动UpdateServer服务才能生效。
log_queue_size	100	CommitLog同步任务队列长度。需要重新启动UpdateServer服务才能生效。
log_sync_delay_tolerable_time_threshold	5s	在同步CommitLog过程中，如果延迟超过该值，则会设置UpdateServer状态为“NOT_SYNC”。
log_sync_delay_warn_report_interval	10s	备UpdateServer接收主UpdateServer的CommitLog延迟时，两次产生报警的最小时间间隔。
log_sync_delay_warn_time_threshold	500ms	备UpdateServer两次接收主UpdateServer的CommitLog的间隔时间超过了该值，则产生报警。
log_sync_retry_times	2	主UpdateServer向备UpdateServer发送CommitLog失败时的重试次数。不建议修改。
log_sync_timeout	500ms	主UpdateServer向备UpdateServer发送CommitLog的超时时间。
log_sync_type	1	主机发备机CommitLog时，写入磁盘的方式可为0或1，0：用于CommitLog的内存缓冲区写满后再刷入CommitLog文件内。1：在用于CommitLog的内存缓冲区内，每写一条日志就刷入CommitLog文件里。需要重新启动UpdateServer服务才能生效。
low_prio_quota_percent	10	对于工作线程中低优先级的CPU计算资源配给比例。
low_priv_adjust_flag	True	是否调整请求处理优先级概率的标志。True：是。False：否。
low_priv_cur_percent	10	低优先级请求处理概率的初始值。

low_priv_network_lower_limit	30MB	低优先级网络请求的最小带宽, 单位M/s, 若low_priv_adjust_flag设置为1，并且最近一段时间低优先级的网络请求使用网络带宽比下面的值小, 则提高低优先级请求的处理概率, 每次调整概率加1%。
low_priv_network_upper_limit	30MB	低优先级网络请求的最大带宽, 单位M/s, 若low_priv_adjust_flag设置为1，并且最近一段时间低优先级的网络请求使用网络带宽比下面的值大, 则降低低优先级请求的处理概率, 每次调整概率减1%。
lsync_fetch_timeout	5s	备UpdateServer从LsyncServer或主UpdateServer读取CommitLog的超时时间。LsyncServer是一个假主机（实际上是一个服务进程），主要提供CommitLog，用于备UpdateServer读取。需要重新启动UpdateServer服务才能生效。
lsync_ip	0.0.0.0	LsyncServer的IP地址。如果配了这个地址，则备UpdateServer从LsyncServer读取CommitLog；如果未配置，则备UpdateServer从主UpdateServer那读取CommitLog。
lsync_port	3000	从LsyncServer读取CommitLog的同步监听端口。取值范围：(1024,65536)
major_freeze_duty_time	Disable, OB_CONFIG_DYNAMIC	每天定时升级主版本的冻结操作的时间。
max_n_lagged_log_allowed	10000	备UpdateServer与主UpdateServer的日志延迟的条数超过了该值时，则产生报警。
max_row_cell_num	256	在Memtable中，当一行中的Cell数量超过该值时，则执行一次合并。
memtable_hash_buckets_size	396867876B	Hash索引大小。需要重新启动UpdateServer服务才能生效。不建议修改。
min_major_freeze_interval	1s	两次升级主版本的最小时间间隔。 如果小于该值，则执行主版本冻结失败。
minor_num_limit	系统自动生成	小版本的个数大于或等于该值后，如果再次执行冻结，则执行主版本冻结。推荐值如下：当app_mod=import时：3；当app_mod=oltp时：1
net_delay_warn_param	50ms;5ms;5;10000	备UpdateServer向主UpdateServer同步CommitLog的网络超时超过该值时，产生告警。
net_warn_threshold	5ms	备UpdateServer向主UpdateServer同步CommitLog的网络超时超过该值时，产生告警。
packet_max_wait_time	10s	通用网络请求超时。需要重新启动UpdateServer服务才能生效。不建议修改。
port	2700	UpdateServer的服务端口。需要重启UpdateServer服务才能生效。
raid_regex	^raid[0-9]+$	raid目录名的匹配式。需要重新启动UpdateServer服务才能生效。不建议修改。
read_queue_size	1000	处理管理命令线程任务队列大小。需要重新启动UpdateServer服务才能生效。

read_thread_count	4	用于管理命令任务的最大线程数。需要重新启动UpdateServer服务才能生效。
refresh_lsync_addr_interval	60s	两次更新LsyncServer地址的时间间隔。
register_timeout	3s	UpdateServer向RootServer注册的超时时间。需要重新启动UpdateServer服务才能生效。不建议修改。
register_times	10	UpdateServer向RootServer注册失败时的重试次数。需要重新启动UpdateServer服务才能生效。不建议修改。
replay_checksum_flag	True	日志回放时，是否对MemTable进行校验和检查。True：是。False：否。
replay_log_buf_size	10GB	用于CommitLog重放的缓冲区大小。不建议修改。
replay_queue_len	500	CommitLog重放时，会先将任务放入一个队列。该参数表示用于CommitLog重放任务的最大队列长度。不建议修改。
replay_wait_time	100ms	两次CommLog重放的时间间隔。不建议修改。
replay_worker_num	20	CommLog重放线程数。
resp_root_timeout	1s	UpdateServer向RootServer汇报冻结数据版本的超时时间。需要重新启动UpdateServer服务才能生效。不建议修改。
resp_root_times	20	UpdateServer向RootServer汇报冻结数据版本失败时的重试次数。需要重新启动UpdateServer服务才能生效。不建议修改。
retry_times	3	UpdateServer与其他Server进行网络互交失败时的重试次数。
root_server_ip	-	UpdateServer所在集群的主RootServer的IP。
root_server_port	2500	UpdateServer所在集群的主RootServer的服务端口。取值范围：(1024,65535)
sstable_block_size	4K	从内存转储到磁盘的SSTable的Block大小。
sstable_compressor_name	none	SSTable从内存转储到磁盘使用的压缩库。
sstable_time_limit	7d	当SSTable加载的时间达到该值时，将被转入“$OB_INSTALL/data/ups_data/raid2/store0/trash”。
state_check_period	500ms	检查UpdateServer主备、是否可服务等内部状态的周期。不建议修改。
store_queue_size	100	用于SSTable从内存转储到磁盘的线程队列长度。
store_root	data/ups_data	Cedar安装目录下UpdateServer的数据目录。需要重新启动UpdateServer服务才能生效。不建议修改。
store_thread_count	3	用于SSTable从内存转储到磁盘的线程个数。需要重新启动UpdateServer服务才能生效。
table_available_error_size	系统自动生成	MemTable可用内存小于该值时，则打印Error日志。推荐值：系统自动生成
table_available_warn_size	系统自动生成	MemTable可用内存小于该值时，则打印Warn日志。

		推荐值：系统自动生成
table_memory_limit	系统自动生成	MemTable可用内存。计算方法：(tatol_memory_limit - total_reserve) / (1/20 + 1/15 + 1)，一般情况下total_reserve =10G推荐值：系统自动生成说明：UpdateServer全局内存包括MemTable、SSTable Cache、事务Session和其他。“total_reserve_gb”为事务Session与其他预留的内存。
total_memory_limit	系统自动生成	UpdateServer可用内存。推荐值：系统自动生成
trans_proc_time_warn	1s	UpdateServer处理某一事务的时间超过该值时，打印一条告警日志。不建议修改。
trans_thread_end_cpu	-1	与trans_thread_start_cpu一起，表示UpdateServer处理事务的CPU范围。例如trans_thread_start_cpu = 1，trans_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于处理事务。
trans_thread_num	30	处理SQL读写请求的工作线程数。
trans_thread_start_cpu	-1	与trans_thread_end_cpu一起，表示UpdateServer处理事务的CPU范围。例如trans_thread_start_cpu = 1， trans_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于处理事务。
using_hash_index	True	是否使用Hash索引。True：是。False：否。
using_memtable_bloomfilter	False	是否使用Bloomfilter。True：是。False：否
using_static_cm_column_id	False	在UpdateServer中，有一个Cache，缓存了table_id和其create_time和mtime的映射。UpdateServer可以根据table_id查询其create_time和modify_time的列ID。该参数表示是否通过查询缓存获取create_time和modify_time的列ID。True：直接使用2和3。False：在缓存中查询获取。
wait_slave_sync_time	100ms	备UPS接收日志的等待时间，需要在此时间之内回包给主UpdateServer。
wait_slave_sync_type	0	备UpdateServer回放CommitLog时，应答主UpdateServer的时机：0：在CommitLog回放前，应答主UpdateServer。1：在CommitLog回放后，且写入磁盘前，应答主UpdateServer。 2：在CommitLog写入磁盘后，应答主UpdateServer。
warm_up_time	10m	SSTable的预热缓存时间。取值范围：[10s,1800s]
write_queue_size	1000	写线程所对应的任务队列的容量。
write_sstable_use_dio	True	是否使用DIO(Direct IO)方式写SSTable。True：直接写入磁盘。False：先写入缓存，再写入磁盘。

3 MergeServer配置参数

参数	默认值	说明
bloom_filter_cache_size	256MB	INSERT语句用bloomfilter大小。
change_obi_timeout	30s	主备集群切换超时时间。

check_ups_log_interval	1	主备集群切换过程中检查UpdateServer日志是否同步的间隔。
devname	eth0	启动MergeServer服务的网卡名称。
frozen_data_cache_size	256M	静态数据缓存大小。
io_thread_count	12	用于内部任务端口的I/O线程数。取值范围：[1,∞)
io_thread_end_cpu	-1	与io_thread_start_cpu一起，表示MergeServer用于IO请求的CPU范围。例如io_thread_start_cpu = 1， io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于IO请求。
io_thread_start_cpu	-1	与io_thread_end_cpu一起，表示MergeServer用于IO请求的CPU范围。例如io_thread_start_cpu = 1，io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于IO请求。
lease_check_interval	6s	检查与RootServer租约是否失效的时间间隔。不建议修改。
lms	False	是否为Listener MergeServer。
location_cache_size	32MB	MergeServer从RootServer中获取Tablet的位置信息，并缓存到本地。该参数表示最大缓存值。
location_cache_timeout	600s	MergeServer从RootServer中获取Tablet的位置信息，并缓存到本地的超时时间。不建议修改。
max_get_rows_per_subreq	20	MergeServer向ChunkServer发送的每个get请求最多包含的行数。如果设置为“0”，MergeServer向ChunkServer发送的每个get请求最多包含的行数不受限制。该参数设置较小时，可以获得更小的响应时间，但会增加MergeServer向ChunkServer发起RPC次数，从而减少整个Cedar系统的QPS；该参数设置较大时，MergeServer处理get请求的响应时间决定于MergeServer发送给ChunkServer的最大的一个get请求的耗时，响应时间可能比较长。取值范围：[0,∞)
max_parellel_count	16	每个请求同一时刻最多并发执行的子请求数量，即一个请求同时最多可有多少个ChunkServer线程并发执行。取值范围：[1,∞)
memory_size_limit_percentage	40	在MergeServer所在服务器的物理内存中，可用于MergeServer的最大百分比数。取值范围：(0,100] 推荐值：40
monitor_interval	600s	MergeServer执行两次内部定时任务（如打印日志等）的的时间间隔。
network_timeout	2s	MergeServer与其他Server进行网络互交的超时时间。推荐值：3s
obmysql_io_thread_count	8	用于MySQL端口的I/O最大线程数。取值范围：[1,∞)
obmysql_io_thread_end_cpu	-1（表示不生效）	与obmysql_io_thread_start_cpu一起，表示MergeServer中MySQL协议端口的IO线程绑定的CPU范围。例如obmysql_io_thread_start_cpu =

		1，obmysql_io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于MySQL协议端口的IO请求。
obmysql_io_thread_start_cpu	-1	与obmysql_io_thread_end_cpu一起，表示MergeServer中MySQL协议端口的IO线程绑定的CPU范围。例如obmysql_io_thread_start_cpu = 1， obmysql_io_thread_end_cpu = 10，表示多核CPU中，从核1到核10用于MySQL协议端口的IO请求。
obmysql_port	2880	MySQL服务端口。取值范围：(1024,65536)
obmysql_task_queue_size	10000	用于SQL操作的任务队列的最大值。取值范围：[1,∞)
obmysql_work_thread_count	120	用于执行SQL任务的最大线程数。取值范围：[1,∞)
port	2800	MergeServer内部任务的服务端口。
query_cache_consistent_expiration_time	10ms	对于一致性读查询，query cache的失效时间。
query_cache_expiration_time	10s	对于非一致性读查询，query cache的失效时间。
query_cache_max_param_num	512	绑定变量数小于这个值的PREPARED STATEMENT才会使用查询结果缓存。
query_cache_max_result	4KB	只有查询结果大小小于该值的PREPARED STATEMENT才会被缓存。
query_cache_size	0	查询缓存允许使用的内存大小。该值为“0”时，表述不启用查询缓存；大于“0”时，启用。取值范围：[1,∞)
query_cache_type	OFF	查询结果缓存功能开关。ON：开。OFF：关。
read_only	False	如果开启只读模式，除了系统表以外不允许执行任何修改操作。True：开。False：关
recorder_fifo_path	run/mergeserver.fifo	记录查询流量的命名管道路径。
retry_times	3	MergeServer向其他Server进行网络互交失败时的重试次数。
root_server_ip	-	MergeServer所在集群的主RootServer的IP。
root_server_port	3500	RootServer的服务端口。取值范围：(1024,65535)
slow_query_threshold	100ms	SQL语句查询时间超过该值时，则标示此次查询为慢查询。
task_left_time	100ms	请求预留给MergeServer的处理时间。如果“一个请求的超时时间 - 在packet quene中的等待时间 < task_left_time”，则放弃处理该请求。不建议修改。
task_queue_size	10000	用于MergeServer内部任务的任务队列最大值。不建议修改。取值范围：[1,∞)
task_thread_count	10	用于处理内部任务的最大线程数。
timeout_percent	70	MergeServer给ChunkServer发SQL请求时，如果该SQL请求的内部超时时间为100ms，那么MergeServer发送给ChunkServer的超时时间

		为“SQL请求的内部超时时间 * timeout_percent”，即“100ms*70%”；剩余时间预留，用于重试其他ChunkServer。取值范围：[10,80]
vtable_location_cache_timeout	8s	虚拟表位置缓存超时时间。

4 ChunkServer配置参数

参数	默认值	说明
appname	-	Cedar启动时指定的App名称。
block_cache_size	1GB	Block缓存大小。类似于Oracle的db cache。配置值越大越好，但是不可超过MergeServer和ChunkServer总内存大小。取值范围：(0,∞)推荐值：1G
block_index_cache_size	512MB	Block索引缓存大小，主要保存每个Block的索引数据。计算方法：(Disk Size / Block Size) * Block Entry SizeBlock的大小一般为4KB~64KB，每个Block的管理开销是：20~30Byte+一个Rowkey长度，假设Rowkey为50个Byte，则一个Block的管理成本70-80byte，如果ChunkServer存储1T的数据，那么索引的管理成本是“(1T/64k)*80Byte=1.28G”。取值范围：(0,∞)推荐值：4G
bypass_sstable_loader_thread_num	0	旁路导入的线程数。当值为“0”时，表示不启用旁路导入。取值范围：[0,10]
check_compress_lib	snappy_1.0:none:lzo_1.0	ChunkServer启动时检查使用的压缩库。
choose_disk_by_space_threshold	60	数据写磁盘时，如果线程数没有超过该值，则选则并发写的线程数最少的磁盘进行写。如果超过该值，则选磁盘空间最小的写。
datadir	/data	SStable存放路径。配置值为绝对路径，不支持相对路径。
devname	bond0	启动ChunkServer使用的网卡名称。
each_tablet_sync_meta	True	每合并一个Tablet是否都将索引文件写入磁盘。True：是。False：合并所有Tablet后或当ChunkServer退出时写入磁盘。
fetch_ups_interval	5s	ChunkServer从RootServer中读取UpdateServer地址列表的时间间隔
file_info_cache_num	4096	文件句柄缓存个数。取值范围：(0,∞)
groupby_mem_size	8MB	使用“Group By”操作符时，每次允许的最大内存。
io_thread_count	4	用于libeasy的I/O线程数。取值范围：[1,∞)
join_batch_count	3000	使用“JION”操作符时，允许的最大的数据条数。取值范围：(0,∞)
join_cache_size	512MB	使用“JION”操作符缓存大小。
lazy_load_sstable	True	ChunkServer启动时，是否立即装载SSTable。

		True：是。False：读取数据是才装载SSTable。
lease_check_interval	5s	检查与RootServer租约是否失效的时间间隔。不建议修改。取值范围：[5s,5s]
max_merge_thread_num	10	每日合并的最大线程数取值范围：[1,32]
max_migrate_task_count	2	Tablet迁移的最大线程数。取值范围：[1,∞)
max_version_gap	3	如果ChunkServer本地版本与RootServer的最后一次冻结版本相差超过该值，则放弃合并本地数据，等待RootServer复制。取值范围：[1,∞)
merge_adjust_ratio	80	如果“ChunkServer负载 > merge_load_high *（1 + merge_adjust_ratio）”时，则挂起当前合并线程。该值为百分数。
merge_delay_for_lsync	5s	每日合并开始时，如果需要读取备UpdateServer数据，则需要备UpdateServer均冻结SSTable。该值表示主UpdateServer等待备UpdateServer冻结SSTable的时间。取值范围：(0,∞)
merge_delay_interval	600s	当收到新版本数据后，需要等待该时间后才开始合并。取值范围：(0,∞)推荐值：600s
merge_highload_sleep_time	2s	ChunkServer负载线程超过“merge_threashold_load_high”时的sleep时间。
merge_mem_limit	64MB	每个合并线程使用的内存大小。
merge_mem_size	8MB	每个查询可能经过多轮merge操作。每轮merge中，如果存储数据的cell array的内存大于该值，本轮merge操作完成。
merge_migrate_concurrency	False	是否允许同时进行数据合并和数据迁移。True：允许。False：不允许。
merge_pause_row_count	2000	合并减速选项。当merge的数据达到该值的行数后，进行一次merge检查。
merge_pause_sleep_time	0	每日合并数据达到“merge_pause_row_count”行后，进行sleep的微秒数。单位：微秒
merge_scan_use_preread	True	当进行每日合并时，是否采用异步I/O机制。True：是。False：否。
merge_thread_per_disk	2	每个disk的合并线程。线程数越多，每日合并速度越快，但查询响应越慢。不建议配置这个选项。取值范围：[1,∞)
merge_threshold_load_high	16	每日合并时，当ChunkServer负载线程超过该值，且每秒get或scan请求的次数超过“merge_threshold_request_high”时，则暂停部分合并线程。取值范围：[1,∞)推荐值：10
merge_threshold_request_high	3000	每日合并时，每秒get或scan请求的最大值次数。如果每秒get或scan请求数超过该值，且ChunkServer负载线程超过“merge_threshold_load_high”，则暂停部分合并线程。取值范围：[1,∞)

merge_timeout	10s	在数据合并时，读取UpdateServer数据的超时时间。取值范围：(0,∞)推荐值：30s
merge_write_sstable_version	2	数据合并后，新SSTable的版本。取值范围：[1,∞)
migrate_band_limit_per_second	50MB	SStable迁移的最大带宽。
min_drop_cache_wait_time	300s	数据合并完成后，原版本数据的保留时间。
min_merge_interval	10s	两次合并最小时间间隔。单位：秒。
network_timeout	3s	ChunkServer与其他Server进行网络互交的超时时间。
over_size_percent_to_split	50	当前合并的SSTable的大小达到“max_sstable_size * (1 + over_size_percent_to_split) ”时，才允许分裂该SSTable。参数可以防止分裂出太多很小的Tablet。该值为百分数。取值范围：(0,∞)
port	2600	ChunkServer的服务端口。
retry_times	3	ChunkServer向其他Server进行网络互交失败时的重试次数。
root_server_ip	-	ChunkServer所在集群的主RootServer的IP。
root_server_port	2500	ChunkServer所在集群的主RootServer的服务端口。
slow_query_warn_time	500ms	鉴定为慢查询的超时时间。如果查询时间超过本选项设置的阈值，ChunkServer打印一条慢查询日志。
sstable_row_cache_size	2GB	SSTable的行缓存大小。取值范围：(0,∞)推荐值：20G
switch_cache_after_merge	False	每日合并完成后，旧版本Block的缓存是否迁移成新版本Block的缓存。True：是。False：否。
task_left_time	300ms	请求预留给MergeServer的处理时间。如果“一个请求的超时时间 - 在packet quene中的等待时间 < task_left_time”，则放弃处理该请求。
task_queue_size	10000	ChunServer中，读任务队列大小。取值范围：[1000,∞)
task_thread_count	20	单个ChunkServer中允许的处理线程总数，OLAP应用中建议配置为核心的2倍左右。取值范围：[1,∞)推荐值：40
unmerge_if_unchanged	True	未修改的SSTable是否需要参与每日合并。选项需要所有ChunkServer在非合并期间统一修改，严禁合并过程中修改并重新加载。True：不需要合并。False：需要合并。
ups_blacklist_timeout	5s	如果该Update Server在黑名单中的时间超过该值时，则该UpdateServer标示为可用状态，并从黑名单中移除。
ups_fail_count	100	如果连接UpdateServer失败次数超过该值时，将该UpdateServer加入到黑名单。取值范围：[1,∞)

write_sstable_use_dio	True	是否使用DIO进行写SSTable。True：是。False：不是。

Cedar 0.2 监控项参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 监控项参考指南	张二宝	

1 RootServer监控项

参数	含义	类型
succ_get_count	get成功次数。	累计值（自增）
succ_scan_count	scan 成功次数。	累计值（自增）
fail_get_count get	失败次数。	累计值（自增）
fail_scan_count scan	失败次数。	累计值（自增）
get_obi_role_count	获取 OBI ROLE 的次数统计。	累计值（自增）
migrate_count	SSTable 的迁移任务数。	累计值（自增）
copy_count	SSTable 的复制任务数。	累计值（自增）
get_schema_count	获取 schema 的次数统计。	累计值（自增）
report_version_count	UpdateServer 汇报冻结版本的次数。	累计值（自增）
all_table_count	集群中 Table 总数。	状态值（更新）
all_tablet_count	集群中 TABLET 总数。	状态值（更新）
all_row_count	集群数据总行数。	状态值（更新）
all_data_size	集群数据总量。	状态值（更新）
location_cache_hit	本地缓存命中次数。（当前版本的Cedar 暂未使用。）	累计值（自增）
location_cache_miss	本地缓存失效次数。（当前版本的Cedar 暂未使用。）	累计值（自增）
rpc_bytes_in	RPC 接收的网络包字节数。	累计值（累加）

2 UpdateServer监控项

参数	含义	类型
get_count	随机读取次数。	表达式（ hl_get_count +nl_get_count +ll_get_count ）

scan_count	顺序扫描次数。	表达式（ hl_scan_count+ nl_scan_count +ll_scan_count ）
trans_count	事务次数。	表达式（ hl_trans_count+ nl_trans_count +ll_trans_count ）
apply_count	写操作次数包括：insert、 update、delete、replace、以及索引修改操作。	表达式（ hl_apply_count+nl_apply_count +ll_apply_count ）
batch_count	事务批处理次数。	累计值（自增）
merge_count	MemTable 行内合并次数。	累计值（自增）
get_qtime	get 操作排队时间。	累计值（累加）
scan_qtime	scan 操作排队时间。	累计值（累加）
apply_qtime	apply 操作排队时间。	表达式（ hl_apply_qtime+ nl_apply_qtime +ll_apply_qtime ）
get_time	get 操作处理时间。	表达式（ hl_get_time +nl_get_time +ll_get_time ）
scan_time	scan 操作处理时间。	表达式（ hl_scan_time +nl_scan_time +ll_scan_time ）
trans_time	事务开始处理到响应完成时间。	表达式（ hl_trans_time+ nl_trans_time +ll_trans_time ）
trans_wtime	事务等待时间。	累计值（累加）
trans_htime	事务处理时间。	累计值（累加）
trans_ctime	事务提交等待时间。	累计值（累加）
trans_ftime	事务日志生成、刷盘及同步时间。	累计值（累加）
trans_rtime	事务响应时间。	累计值（累加）
apply_time	写操作处理时间。	表达式（ hl_apply_time+ nl_apply_time+ll_apply_time ）
batch_time	批处理时间。	累计值（累加）
merge_time	MemTable 行内合并时间。	累计值（累加）
memory_total	内存使用总量。	状态值（更新）
memory_limit	可用内存总量。	状态值（更新）
memtable_total	MemTable 内存分配总量。	状态值（更新）
memtable_used	MemTable 内存使用总量。	状态值（更新）
total_rows	MemTable 总行数。	状态值（更新）
active_memtable_limit	活跃 MemTable 可用内存总量。	状态值（更新）
active_memtable_total	活跃 MemTable 内存分配总量	状态值（更新）

active_memtable_used	活跃 MemTable 内存使用总量。	状态值（更新）
active_total_rows	活跃 MemTable 总行数。	状态值（更新）
frozen_memtable_limit	冻结 MemTable 可用内存总量。	状态值（更新）
frozen_memtable_total	冻结 MemTable 内存分配总量。	状态值（更新）
frozen_memtable_used	冻结 MemTable 内存使用总量。	状态值（更新）
frozen_total_rows	冻结 MemTable 总行数。	状态值（更新）
apply_fail_count	失败的 apply 操作次数。	累计值（自增）
pakcet_long_wait_count	长时间等待的包数。	累计值（自增）
commit_log_size	CommitLog 大小。	累计值（累加）
commit_log_id	CommitLog ID。	状态值（更新）
clog_sync_count	同步日志数。	累计值（自增）
clog_sync_delay	同步日志延时。	累计值（累加）
slow_clog_sync_count	慢速同步日志数。	累计值（自增）
slow_clog_sync_delay	慢速同步日志延时。	累计值（累加）
last_replay_clog_time	上次回放日志时间。	状态值（更新）
frozen_version	冻结版本号。	状态值（更新）
apply_row_count	apply 行数。	累计值（自增）
apply_row_unmerged_cell_count	apply 行内未合并的单元数。	累计值（自增）
ll_get_count	低优先级随机读取次数。	累计值（自增）
ll_scan_count	低优先级顺序扫描次数。	累计值（自增）
ll_apply_count	低优先级写操作次数。	累计值（自增）
ll_trans_count	低优先级事务次数。	累计值（自增）
ll_apply_qtime	低优先级写操作排队时间。	累计值（累加）
nl_get_count	中优先级随机读取次数。	累计值（自增）
nl_scan_count	中优先级顺序扫描次数。	累计值（自增）
nl_apply_count	中优先级写操作次数。	累计值（自增）
nl_trans_count	中优先级事务次数。	累计值（自增）
nl_apply_qtime	中优先级写操作排队时间。	累计值（累加）
hl_get_count	高优先级随机读取次数。	累计值（自增）
hl_scan_count	高优先级顺序扫描次数。	累计值（自增）
hl_apply_count	高优先级写操作次数。	累计值（自增）
hl_trans_count	高优先级事务次数。	累计值（自增）
hl_apply_qtime	高优先级写操作排队时间。	累计值（累加）
ll_get_time	低优先级随机读取处理时间。	累计值（累加）

ll_scan_time	低优先级顺序扫描处理时间。	累计值（累加）
ll_apply_time	低优先级写操作处理时间。	累计值（累加）
ll_trans_time	低优先级事务处理到响应完成时间。	累计值（累加）
nl_get_time	中优先级随机读取处理时间。	累计值（累加）
nl_scan_time	中优先级顺序扫描处理时间。	累计值（累加）
nl_apply_time	中优先级写操作处理时间。	累计值（累加）
nl_trans_time	中优先级事务处理到响应完成时间。	累计值（累加）
hl_get_time	高优先级随机读取处理时间。	累计值（累加）
hl_scan_time	高优先级顺序扫描处理时间。	累计值（累加）
hl_apply_time	高优先级写操作处理时间。	累计值（累加）
hl_trans_time	高优先级事务处理到响应完成时间。	累计值（累加）
lock_wait_time	行锁等待时间。	累计值（累加）
dml_replace_count	REPLACE 语句数。	累计值（自增）
dml_insert_count	INSERT 语句数。	累计值（自增）
dml_update_count	UPDATE 语句数。	累计值（自增）
dml_delete_count	DELETE 语句数。	累计值（自增）
sql_grant_privilege_count	执行用户授权语句次数。	累计值（自增）
sql_revoke_privilege_count	执行撤销权限语句次数。	累计值（自增）
sql_show_grants_count	执行查看权限语句次数。	累计值（自增）
sql_create_user_count	执行新建用户语句次数。	累计值（自增）
sql_drop_user_count	执行删除用户语句次数。	累计值（自增）
sql_lock_user_count	执行锁定用户语句次数。	累计值（自增）
sql_set_password_count	执行修改密码语句次数。	累计值（自增）
sql_rename_user_count	执行修改用户名语句次数。	累计值（自增）
sql_create_table_count	执行创建表语句次数。	累计值（自增）
sql_create_index_count	执行创建索引语句次数。	累计值（自增）
sql_drop_table_count	执行删除表语句次数。	累计值（自增）
sql_ps_allocator_count	PrepareStatementAllocator 的个数，它是一个动态值，新创建一个PrepareStatement 语句，就增加一个PrepareStatementAllocator，关闭 PrepareStatement 语句就减少一个 PSAllocator。	状态值（更新）
sql_insert_cache_hit	INSERT 操作缓存命中次数。	累计值（自增）
sql_insert_cache_miss	INSER 操作缓存失效次数。	累计值（自增）
sql_update_cache_hit	UPDATE 操作缓存命中次数。	累计值（自增）

sql_update_cache_miss	UPDATE 操作缓存失效次数。	累计值（自增）
sql_query_cache_hit	query cache 命中次数。	累计值（自增）
sql_query_cache_miss	query cache 失效次数。	累计值（自增）
distinct_stmt_count	执行不同 SQL 语句的种类。	累计值（自增）
ps_count	活跃的 PREPAREStatement 语句数。	状态值（更新）
location_cache_hit	本地缓存命中次数。（当前版本的Cedar 暂未使用。）	累计值（自增）
location_cache_miss	本地缓存失效次数。（当前版本的Cedar 暂未使用。）	累计值（自增）
rpc_bytes_in	RPC 接收的网络包字节数。	累计值（累加）
rpc_bytes_out	RPC 发送的网络包字节数。	累计值（累加）
block_index_cache_hit	Block 索引缓存命中次数。	累计值（自增）
block_index_cache_miss	Block 索引缓存失效次数。	累计值（自增）
block_cache_hit	Block 缓存命中次数。	累计值（自增）
block_cache_miss	Block 缓存失效次数。	累计值（自增）
sstable_disk_io_num	SSTable 读操作次数。	累计值（自增）
sstable_disk_io_bytes	SSTable 读操作数据量。	累计值（累加）
sstable_disk_io_write_num	SSTable 写操作次数。	累计值（自增）
sstable_disk_io_write_bytes	SSTable 写操作数据量。	累计值（累加）
sstable_row_cache_hit	SSTable rowcache 命中次数。	累计值（自增）
sstable_row_cache_miss	SSTable rowcache 失效次数。	累计值（自增）
sstable_get_rows	SSTable “get_row”操作次数。	累计值（自增）
sstable_scan_rows	SSTable“scan_row”操作次数。	累计值（自增）

3 MergeServer监控项

参数	含义	类型
nb_get_count	服务 get 请求的次数。	累计值（自增）
nb_get_time	服务 get 请求的总时间。	累计值（累加）
nb_scan_count	服务 scan 请求的次数。	累计值（自增）
nb_scan_time	服务 scan 请求的总时间。	累计值（累加）
get_event_count	get 事件次数。	累计值（自增）
get_event_time	get 事件处理总时间。	累计值（累加）
scan_event_count	can 事件次数。	累计值（自增）
scan_event_time	scan 事件处理总时间。	累计值（累加）

ms_memory_limit	最大内存限制。	状态值（更新）
ms_memory_total	内存使用总量。	状态值（更新）
ms_memory_parser	parser 内存使用量。	状态值（更新）
ms_memory_transformer	transformer 内存使用量。	状态值（更新）
ms_memory_ps_plan	ps plan 内存使用量。	状态值（更新）
ms_memory_rpc_request	rpc request 内存使用量。	状态值（更新）
ms_memory_sql_array	sql array 内存使用量。	状态值（更新）
ms_memory_expression	expression 内存使用量。	状态值（更新）
ms_memory_row_store	row store 内存使用量。	状态值（更新）
ms_memory_session	session 内存使用量。	状态值（更新）
sql_grant_privilege_count	执行用户授权语句次数。	累计值（自增）
sql_revoke_privilege_count	执行撤销权限语句次数。	累计值（自增）
sql_show_grants_count	执行查看权限语句次数。	累计值（自增）
sql_create_user_count	执行新建用户语句次数。	累计值（自增）
sql_drop_user_count	执行删除用户语句次数。	累计值（自增）
sql_lock_user_count	执行锁定用户语句次数。	累计值（自增）
sql_set_password_count	执行修改密码语句次数。	累计值（自增）
sql_rename_user_count	执行修改用户名语句次数。	累计值（自增）
sql_create_table_count	执行创建表语句次数。	累计值（自增）
sql_drop_table_count	执行删除表语句次数。	累计值（自增）
sql_ps_allocator_count	Prepare StatementAllocator 的个数，它是一个动态值，新创建一个PrepareStatement 语句，就增加一个 PrepareStatementAllocator，关闭 Prepare Statement语句就减少一个 PSAllocator。	状态值（更新）
sql_insert_cache_hit	INSERT 操作缓存命中次数。	累计值（自增）
sql_insert_cache_miss	INSERT 操作缓存未命中次数。	累计值（自增）
sql_update_cache_hit	UPDATE 操作缓存命中次数。	累计值（自增）
sql_update_cache_miss	UPDATE 操作缓存未命中次数。	累计值（自增）
sql_query_cache_hit	SQL 请求缓存命中次数。	累计值（自增）
sql_query_cache_miss	SQL 请求缓存失效次数。	累计值（自增）
distinct_stmt_count	执行不同 SQL 语句的种类。	累计值（自增）
ps_count	活跃的 PREPAREStatement 语句数。	状态值（更新）
sql_succ_query_count	执行 SQL 请求成功次数。	累计值（自增）
sql_fail_query_count	执行 SQL 请求失败次数。	累计值（自增）

sql_succ_prepare_count	执行 PREPARE 成功次数。	累计值（自增）
sql_fail_prepare_count	执行 PREPARE 失败次数。	累计值（自增）
sql_succ_exec_count	执行 EXECUTE 成功次数。	累计值（自增）
sql_fail_exec_count	执行 EXECUTE 失败次数。	累计值（自增）
sql_succ_close_count	关闭 SQL 请求成功的次数。	累计值（自增）
sql_fail_close_count	关闭 SQL 请求失败的次数。	累计值（自增）
sql_succ_login_count	登录 Cedar 的成功次数。	累计值（自增）
sql_fail_login_count	登录 Cedar 的失败次数。	累计值（自增）
sql_logout_count	退出 Cedar 的次数。	累计值（自增）
sql_compound_count	compound 语句个数。	累计值（自增）
sql_compound_time	compound 语句执行时间。	累计值（累加）
sql_select_count	执行 SELECT 次数。	累计值（自增）
sql_select_time	SELECT 语句执行时间。	累计值（累加）
sql_insert_count	执行 INSERT 次数。	累计值（自增）
sql_insert_time	INSERT 语句执行时间。	累计值（累加）
sql_replace_count	执行 REPLACE 次数。	累计值（自增）
sql_replace_time	REPLACE 语句执行时间。	累计值（累加）
sql_update_count	执行 UPDATE 次数。	累计值（自增）
sql_update_time	UPDATE 语句执行时间。	累计值（累加）
sql_delete_count	执行 DELETE 次数。	累计值（自增）
sql_delete_time	DELETE 语句执行时间。	累计值（累加）
sql_query_bytes	SQL 请求数据总量。	累计值（累加）
sql_commit_count	执行 COMMIT 次数。	累计值（自增）
sql_rollback_count	执行 ROLLBACK 次数。	累计值（自增）
sql_autocommit_on_count	设置 autocommit on语句个数。	累计值（自增）
sql_autocommit_off_count	设置 autocommit off语句个数。	累计值（自增）
location_cache_hit	SSTable 的本地缓存命中次数。MergeServer 每次向ChunkServer 取数据时，优先从 locationcache 中获取location 信息，如果命中，则location_cache_hit加一。	累计值（自增）
location_cache_miss	SSTable 的本地缓存失效次数。MergeServer 每次向ChunkServer 取数据时，优先从 locationcache 中获取location 信息，如果未命中，则location_cache_miss加一。	累计值（自增）
rpc_bytes_in	RPC 接收的网络包字节数。	累计值（累加）
rpc_bytes_out	RPC 发送的网络包字节数。	累计值（累加）

4 ChunkServer监控项

参数	含义	类型
serving_version	当前服务的数据版本号。	状态值（更新）
old_ver_tablets_num	旧版本的TABLET 个数。	状态值（更新）
old_ver_merged_tablets_num	旧版本完成合并的 tablet 个数。	状态值（更新）
new_ver_tablets_num	新版本的TABLET 个数。	状态值（更新）
memory_used_network	network 内存使用量。	状态值（更新）
memory_used_thread_buffer	thread buffer 内存使用量。	状态值（更新）
memory_used_tablet	tablet 内存使用量。	状态值（更新）
memory_used_bi_cache	block index cache内存使用量。	状态值（更新）
memory_used_block_cache	block cache 内存使用量。	状态值（更新）
memory_used_join_cache	join cache 内存使用量。	状态值（更新）
memory_used_sstable_row_cache	stable rowcache 内存使用量。	状态值（更新）
memory_used_merge_buffer	merge buffer 内存使用量。	状态值（更新）
request_count	请求次数。	累计值（自增）
request_count_per_second	每秒的请求次数。	表达式（计算request_count 变化速率）
queue_wait_time	请求队列等待时间。	累计值（累加）
get_count	get 请求次数。	累计值（自增）
scan_count	scan 请求次数。	累计值（自增）
get_time	get 请求处理时间。	累计值（累加）
scan_time	scan 请求处理时间。	累计值（累加）
get_bytes	get 请求数据量。	累计值（累加）
scan_bytes	scan 请求数据量。	累计值（累加）
location_cache_hit	本地缓存命中次数。（当前版本的 Cedar 暂未使用。）	累计值（自增）
location_cache_miss	本地缓存失效次数。（当前版本的 Cedar 暂未使用。）	累计值（自增）
rpc_bytes_in	RPC 接收的网络包字节数。	累计值（累加）
rpc_bytes_out	RPC 发送的网络包字节数。	累计值（累加）
block_index_cache_hit	Block 索引缓存命中次数。	累计值（自增）
block_index_cache_miss	Block 索引缓存失效次数。	累计值（自增）
block_cache_hit	Block 缓存命中次数。	累计值（自增）
block_cache_miss	Block 缓存失效次数。	累计值（自增）

sstable_disk_io_num	SSTable 读操作次数。	累计值（自增）
sstable_disk_io_bytes	SSTable 读操作数据量。	累计值（累加）
sstable_disk_io_write_num	SSTable 写操作次数。	累计值（自增）
sstable_disk_io_write_bytes	SSTable 写操作数据量。	累计值（累加）
sstable_row_cache_hit	SSTable rowcache 命中次数。	累计值（自增）
sstable_row_cache_miss	SSTable rowcache 失效次数。	累计值（自增）
sstable_get_rows	SSTable“get_row”操作次数。	累计值（自增）
sstable_scan_rows	SSTable“scan_row”操作次数。	累计值（自增）

Cedar 0.2 错误码参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 错误码参考指南	屈兴	

1 用户结果码

> 用于定位和解决Cedar 客户端或者MySQL客户端返回的错误。

结果码	含义	参数
-12	请求超时。	const int OB_RESPONSE_TIME_OUT
-13	分配内存空间失败。	const int OB_ALLOCATE_MEMORY_FAILED
-14	UpdateServer内存溢出。	const int OB_MEM_OVERFLOW
-38	读写集群不是主集群。	const int OB_NOT_MASTER
-42	Cedar用户不存在。	const int OB_USER_NOT_EXIST
-43	连接Cedar的密码错误。	const int OB_PASSWORD_WRONG
-46	没有操作权限。	const int OB_NO_PERMISSION
-49	UpdateServer处理超时。	const int OB_PROCESS_TIMEOUT
-56	建立TCP连接失败。	const int OB_CONN_ERROR
-64	MergeServer将RPC请求拆分成多个SESSION从ChunkServer上读取数据时，SESSION被杀死。	const int OB_SESSION_KILLED
-119	冻结时事务被回滚。	const int OB_TRANS_ROLLBACKED
-4007	主集群在每日合并过程中不允许进行DDL操作。备集群不允许进行DD操作。	const int OB_OP_NOT_ALLOW
-5001	SQL语法错误。	const int OB_ERR_PARSE_SQL

-5009	列不存在。	const int OB_ERR_COLUMN_UNKNOWN
-5019	表格不存在。	const int OB_ERR_TABLE_UNKNOWN
-5035	Cedar用户不存在。	const int OB_ERR_USER_NOT_EXIST
-5036	没有操作权限。	const int OB_ERR_NO_PRIVILEGE
-5038	连接Cedar的密码错误。	const int OB_ERR_WRONG_PASSWORD
-5040	更新RowKey列错误。	const int OB_ERR_UPDATE_ROWKEY_COLUMN
-5041	更新JOIN列错误。	const int OB_ERR_UPDATE_JOIN_COLUMN
-5048	执行REPLACE操作的行锁冲突。	const int OB_ERR_EXCLUSIVE_LOCK_CONFLICT
-5049	执行INSERT/UPDATE/DELETE操作的行锁冲突。	const int OB_ERR_SHARED_LOCK_CONFLICT
-5060	事务已开始，开启新事务失败。	const int OB_ERR_TRANS_ALREADY_STARTED
-5062	不在事务中。	const int OB_ERR_NOT_IN_TRANS

2 系统结果码

> 用于定位和解决Cedar的系统内部错误。

结果码	含义	参数
-1	Object类型错误。	const int OB_OBJ_TYPE_ERROR
-2	传入的参数错误。	const int OB_INVALID_ARGUMENT
-3	数组越界。	const int OB_ARRAY_OUT_OF_RANGE
-4	服务器监听错误。	const int OB_SERVER_LISTEN_ERROR
-5	已经初始化。	const int OB_INIT_TWICE
-6	没有初始化。	const int OB_NOT_INIT
-7	不支持的功能。	const int OB_NOT_SUPPORTED
-8	迭代到最后一行。	const int OB_ITER_END
-9	IO错误。	const int OB_IO_ERROR
-10	版本错误。	const int OB_ERROR_FUNC_VERSION
-11	包未发送。	const int OB_PACKET_NOT_SENT
-15	系统错误。	const int OB_ERR_SYS
-16	未知错误。	const int OB_ERR_UNEXPECTED
-17	项已存在。	const int OB_ENTRY_EXIST

-18	项不存在。	const int OB_ENTRY_NOT_EXIST
-19	大小越界。	const int OB_SIZE_OVERFLOW
-20	引用计数不为零。	const int OB_REF_NUM_NOT_ZERO
-21	值发生冲突。	const int OB_CONFLICT_VALUE
-22	项未设置。	const int OB_ITEM_NOT_SETTED
-23	需要重试。	const int OB_EAGAIN
-24	缓冲不足。	const int OB_BUF_NOT_ENOUGH
-25	同步Slave过程失败。	const int OB_PARTIAL_FAILED
-26	读取内容为空。	const int OB_READ_NOTHING
-27	文件不存在。	const int OB_FILE_NOT_EXIST
-28	日志不连续。	const int OB_DISCONTINUOUS_LOG
-29	Schema错误。	const int OB_SCHEMA_ERROR
-30	不提供该数据服务。	const int OB_DATA_NOT_SERVE
-31	未知的Object。	const int OB_UNKNOWN_OBJ
-32	没有监控数据。	const int OB_NO_MONITOR_DATA
-33	序列化失败。	const int OB_SERIALIZE_ERROR
-34	反序列化失败。	const int OB_DESERIALIZE_ERROR
-35	AIO超时。	const int OB_AIO_TIMEOUT
-36	需要重试。	const int OB_NEED_RETRY
-37	SSTable过多。	const int OB_TOO_MANY_SSTABLE
-39	Token失效。	const int OB_TOKEN_EXPIRED
-40	加密失败。	const int OB_ENCRYPT_FAILED
-41	解密失败。	const int OB_DECRYPT_FAILED
-44	密钥错误。	const int OB_SKEY_VERSION_WRONG
-45	不是Token类型。	const int OB_NOT_A_TOKEN
-47	条件检查失败。	const int OB_COND_CHECK_FAIL
-48	未注册。	const int OB_NOT_REGISTERED
-50	不是这个Object。	const int OB_NOT_THE_OBJECT
-51	重复注册。	const int OB_ALREADY_REGISTERED
-52	最后一个日志损坏。	const int OB_LAST_LOG_RUINNED
-53	没有ChunkServer被选择。	const int OB_NO_CS_SELECTED
-54	没有Tablets被创建。	const int OB_NO_TABLETS_CREATED
-55	无效。	const int OB_INVALID_ERROR

-57	十进制数溢出。	const int OB_DECIMAL_OVERFLOW_WARN
-58	十进制数不合法。	const int OB_DECIMAL_ILLEGAL_ERROR
-59	除零非法。	const int OB_OBJ_DIVIDE_BY_ZERO
-60	除法错误。	const int OB_OBJ_DIVIDE_ERROR
-61	不是十进制数。	const int OB_NOT_A_DECIMAL
-62	十进制数精度不一致。	const int OB_DECIMAL_PRECISION_NOT_EQUAL
-63	获取空取值范围。	const int OB_EMPTY_RANGE
-65	日志不同步。	const int OB_LOG_NOT_SYNC
-66	目录不存在。	const int OB_DIR_NOT_EXIST
-67	结束流式接口调用。	const int OB_NET_SESSION_END
-68	日志无效。	const int OB_INVALID_LOG
-69	日志填充。	const int OB_FOR_PADDING
-70	数据无效。	const int OB_INVALID_DATA
-71	已经完成。	const int OB_ALREADY_DONE
-72	操作被取消。	const int OB_CANCELED
-73	数据源变更。	const int OB_LOG_SRC_CHANGED
-74	日志不对齐。	const int OB_LOG_NOT_ALIGN
-75	日志记录丢失。	const int OB_LOG_MISSING
-76	需要等待。	const int OB_NEED_WAIT
-77	不支持的操作。	const int OB_NOT_IMPLEMENT
-78	被零除。	const int OB_DIVISION_BY_ZERO
-79	值超出范围。	const int OB_VALUE_OUT_OF_RANGE
-80	超出内存限制。	const int OB_EXCEED_MEM_LIMIT
-81	未知的结果。	const int OB_RESULT_UNKNOWN
-82	数据格式错误。	const int OB_ERR_DATA_FORMAT
-83	可能循环。	const int OB_MAYBE_LOOP
-84	没有结果。	const int OB_NO_RESULT
-85	队列溢出。	const int OB_QUEUE_OVERFLOW
-101	死锁。	const int OB_DEAD_LOCK
-102	日志不完整。	const int OB_PARTIAL_LOG
-103	校验和错误。	const int OB_CHECKSUM_ERROR
-104	初始化失败。	const int OB_INIT_FAIL
-110	读出全零日志。	const int OB_READ_ZERO_LOG
-111	切换日志错误。	const int OB_SWITCH_LOG_NOT_MATCH

-112	写日志未开始。	const int OB_LOG_NOT_START
-113	致命错误状态。	const int OB_IN_FATAL_STATE
-114	停止状态。	const int OB_IN_STOP_STATE
-115	主UpdateServer已存在。	const int OB_UPS_MASTER_EXISTS
-116	日志未清除。	const int OB_LOG_NOT_CLEAR
-117	文件已经存在。	const int OB_FILE_ALREADY_EXIST
-118	未知的包。	const int OB_UNKNOWN_PACKET
-120	日志过大。	const int OB_LOG_TOO_LARGE
-121	同步发包失败。	const int OB_RPC_SEND_ERROR
-122	异步发包失败。	const int OB_RPC_POST_ERROR
-123	Libeasy错误。	const int OB_LIBEASY_ERROR
-124	连接错误。	const int OB_CONNECT_ERROR
-125	不处于释放状态。	const int OB_NOT_FREE
-126	初始化SQL语境失败。	const int OB_INIT_SQL_CONTEXT_ERROR
-127	跳过无效的Row。	const int OB_SKIP_INVALID_ROW
-131	系统配置表错误。	const int OB_SYS_CONFIG_TABLE_ERROR
-132	读取配置失败。	const int OB_READ_CONFIG_ERROR
-140	事务不匹配。	const int OB_TRANS_NOT_MATCH
-141	只读事务。	const int OB_TRANS_IS_READONLY
-142	Row被修改。	const int OB_ROW_MODIFIED
-143	版本不匹配。	const int OB_VERSION_NOT_MATCH
-144	无效的地址。	const int OB_BAD_ADDRESS
-145	重复的COLUMN。	const int OB_DUPLICATE_COLUMN
-146	进队列失败。	const int OB_ENQUEUE_FAILED
-147	配置无效。	const int OB_INVALID_CONFIG
-151	存储过程退出。	const int OB_SP_EXIT
-1001	ChunkServer cache未命中。	const int OB_CS_CACHE_NOT_HIT
-1002	ChunkServer超时。	const int OB_CS_TIMEOUT
-1008	Tablet不存在。	const int OB_CS_TABLET_NOT_EXIST
-1010	需要重试。	const int OB_CS_EAGAIN
-1011	扫描下一列。	const int OB_GET_NEXT_COLUMN
-1012	扫描下一行。	const int OB_GET_NEXT_ROW
-1013	反序列化失败。	const int OB_DESERIALIZE_ERROR

-1014	RowKey无效。	const int OB_INVALID_ROW_KEY
-1015	不支持的查询模式。	const int OB_SEARCH_MODE_NOT_IMPLEMENT
-1016	错误的Block索引。	const int OB_INVALID_BLOCK_INDEX
-1017	错误的Block数据。	const int OB_INVALID_BLOCK_DATA
-1018	未找到。	const int OB_SEARCH_NOT_FOUND
-1020	查询的Key或者取值范围不在当前的Tablet中。	const int OB_BEYOND_THE_RANGE
-1021	完成移动块缓存。	const int OB_CS_COMPLETE_TRAVERSAL
-1022	Row的结尾。	const int OB_END_OF_ROW
-1024	ChunkServer合并失败。	const int OB_CS_MERGE_ERROR
-1025	ChunkServer的Schema不兼容。	const int OB_CS_SCHEMA_INCOMPATIBLE
-1026	ChunkServer服务未启动。	const int OB_CS_SERVICE_NOT_STARTED
-1027	Lease过期。	const int OB_CS_LEASE_EXPIRED
-1028	合并超时。	const int OB_CS_MERGE_HAS_TIMEOUT
-1029	表已经被删除。	const int OB_CS_TABLE_HAS_DELETED
-1030	ChunkServer合并取消。	const int OB_CS_MERGE_CANCELED
-1031	压缩依赖库出错。	const int OB_CS_COMPRESS_LIB_ERROR
-1032	超出磁盘空间。	const int OB_CS_OUTOF_DISK_SPACE
-1033	ChunkServer移植已经存在的Tablet。	const int OB_CS_MIGRATE_IN_EXIST
-1037	错误的SSTable数据。	const int OB_WRONG_SSTABLE_DATA
-1039	COLUMN GROUP未找到。	const int OB_COLUMN_GROUP_NOT_FOUND
-1040	没有导入的SSTable。	const int OB_NO_IMPORT_SSTABLE
-1041	导入SSTable不存在。	const int OB_IMPORT_SSTABLE_NOT_EXIST
-1042	索引构建任务超时。	const int OB_CS_STATIC_INDEX_TIMEOUT
-1043	索引表构建全局阶段找不到局部阶段构建的局部sstable。	const int OB_TABLET_HAS_NO_LOCAL_SSTABLE
-1044	构建索引表任务重试后仍失败。	const int OB_TABLET_FOR_INDEX_ALL_FAILED
-1045	构建索引表任务失败，稍后重试。	const int OB_INDEX_BUILD_FAILED
-2001	UpdateServer事务正在进行中。	const int OB_UPS_TRANS_RUNNING
-2002	MemTable已经被冻结。	const int OB_FREEZE_MEMTABLE_TWICE
-2003	MemTable已经被删除。	const int OB_DROP_MEMTABLE_TWICE
-2004	MemTable启动的版本无效。	const int OB_INVALID_START_VERSION
-2005	UpdateServer不存在。	const int OB_UPS_NOT_EXIST
-2006	UpdateServer获取memtable或者sstable失败。	const int OB_UPS_ACQUIRE_TABLE_FAIL
-2007	主版本无效。	const int OB_UPS_INVALID_MAJOR_VERSION

-2008	UpdateServer表未冻结。	const int OB_UPS_TABLE_NOT_FROZEN
-2009	UpdateServer切主超时。	const int OB_UPS_CHANGE_MASTER_TIMEOUT
-2010	强制终止时间。	const int OB_FORCE_TIME_OUT
-2012	备UPS上的未决日志和主UPS中的不同。	const int OB_NOT_EQUAL
-2013	备UPS本地所有的未决日志都和主UPS相同	const int OB_NOTIFY_EQUAL
-3001	时间戳错误。	const int OB_ERROR_TIME_STAMP
-3002	交叉错误。	const int OB_ERROR_INTRESECT
-3003	超出取值范围。	const int OB_ERROR_OUT_OF_RANGE
-3004	RootServer状态初始化。	const int OB_RS_STATUS_INIT
-3005	不提供旁路服务。	const int OB_IMPORT_NOT_IN_SERVER
-3006	发现取值范围。	const int OB_FIND_OUT_OF_RANGE
-3007	转换错误。	const int OB_CONVERT_ERROR
-3008	遍历MergeServer列表介绍。	const int OB_MS_ITER_END
-3036	启动各Server后已经设置过主集群。	const int OB_RS_LEADER_SETTED_WHEN_STARTING
-4001	内部状态错误。	const int OB_INNER_STAT_ERROR
-4002	Schema版本太旧。	const int OB_OLD_SCHEMA_VERSION
-4003	输入参数错误。	const int OB_INPUT_PARAM_ERROR
-4004	找不到空项。	const int OB_NO_EMPTY_ENTRY
-4005	释放Schema失败。	const int OB_RELEASE_SCHEMA_ERROR
-4006	项目数量统计错误。	const int OB_ITEM_COUNT_ERROR
-4008	ChunkServer缓存失败。	const int OB_CHUNK_SERVER_ERROR
-4009	没有新Schema。	const int OB_NO_NEW_SCHEMA
-4010	子扫描请求过多。	const int OB_MS_SUB_REQ_TOO_MANY
-5000	启动SQL失败。	const int OB_ERR_SQL_START
-5000	语法解析初始化失败。	const int OB_ERR_PARSER_INIT
-5002	解析SQL失败。	const int OB_ERR_RESOLVE_SQL
-5003	产生物理执行计划错误。	const int OB_ERR_GEN_PLAN
-5004	未知的系统功能错误。	const int OB_ERR_UNKNOWN_SYS_FUNC
-5005	解析SQL语法时分配内存错误。	const int OB_ERR_PARSER_MALLOC_FAILED
-5006	解析SQL语法错误。	const int OB_ERR_PARSER_SYNTAX
-5007	COLUMN大小错误。	const int OB_ERR_COLUMN_SIZE
-5008	重复列。	const int OB_ERR_COLUMN_DUPLICATE
-5010	未知的操作错误。	const int OB_ERR_OPERATOR_UNKNOWN

-5011	“*”使用错误。	const int OB_ERR_STAR_DUPLICATE
-5012	ID不合法。	const int OB_ERR_ILLEGAL_ID
-5013	位置错误。	const int OB_ERR_WRONG_POS
-5014	值不合法。	const int OB_ERR_ILLEGAL_VALUE
-5015	列描述错误。	const int OB_ERR_COLUMN_AMBIGOUS
-5016	逻辑计划失败。	const int OB_ERR_LOGICAL_PLAN_FAILD
-5017	Schema未设置。	const int OB_ERR_SCHEMA_UNSET
-5018	名称不合法。	const int OB_ERR_ILLEGAL_NAME
-5020	表格重复。	const int OB_ERR_TABLE_DUPLICATE
-5021	字符串被截断。	const int OB_ERR_NAME_TRUNCATE
-5022	表达式错误。	const int OB_ERR_EXPR_UNKNOWN
-5023	类型不合法。	const int OB_ERR_ILLEGAL_TYPE
-5024	主键已经存在。	const int OB_ERR_PRIMARY_KEY_DUPLICATE
-5025	已经存在。	const int OB_ERR_ALREADY_EXISTS
-5026	create_time列已经存在。	const int OB_ERR_CREATETIME_DUPLICATE
-5027	modify_time列已经存在。	const int OB_ERR_MODIFYTIME_DUPLICATE
-5028	索引非法。	const int OB_ERR_ILLEGAL_INDEX
-5029	Schema无效。	const int OB_ERR_INVALID_SCHEMA
-5030	插入空RowKey。	const int OB_ERR_INSERT_NULL_ROWKEY
-5031	COLUMN未找到。	const int OB_ERR_COLUMN_NOT_FOUND
-5032	删除空RowKey。	const int OB_ERR_DELETE_NULL_ROWKEY
-5033	插入Join列错误。	const int OB_ERR_INSERT_INNER_JOIN_COLUMN
-5034	用户为空。	const int OB_ERR_USER_EMPTY
-5037	没有进入权限。	const int OB_ERR_NO_AVAILABLE_PRIVILEGE_ENTRY
-5039	用户被锁。	const int OB_ERR_USER_IS_LOCKED
-5042	无效的列名。	const int OB_ERR_INVALID_COLUMN_NUM
-5043	未知的PREPARE语句。	const int OB_ERR_PREPARE_STMT_UNKNOWN
-5044	未知变量。	const int OB_ERR_VARIABLE_UNKNOWN
-5045	初始化SESSION失败。	const int OB_ERR_SESSION_INIT
-5046	旧权限版本。	const int OB_ERR_OLDER_PRIVILEGE_VERSION
-5047	缺少RowKey列。	const int OB_ERR_LACK_OF_ROWKEY_COL
-5050	用户已存在。	const int OB_ERR_USER_EXIST
-5051	密码为空。	const int OB_ERR_PASSWORD_EMPTY

-5052	授予CREATE TABLE权限错误。	const int OB_ERR_GRANT_PRIVILEGES_TO_CREATE_TABLE
-5053	错误的动态参数。	const int OB_ERR_WRONG_DYNAMIC_PARAM
-5054	参数大小错误。	const int OB_ERR_PARAM_SIZE
-5055	未知的功能错误。	const int OB_ERR_FUNCTION_UNKNOWN
-5056	建立molidfy_time列错误。	const int OB_ERR_CREAT_MODIFY_TIME_COLUMN
-5057	修改Primary Key错误。	const int OB_ERR_MODIFY_PRIMARY_KEY
-5058	重复的参数。	const int OB_ERR_PARAM_DUPLICATE
-5059	SESSION过多。	const int OB_ERR_TOO_MANY_SESSIONS
-5061	PS次数太多。	const int OB_ERR_TOO_MANY_PS
-5063	未知的HINT错误。	const int OB_ERR_HINT_UNKNOWN
-5075	数据表上没有建立索引表。	const int OB_INDEX_NOT_EXIST
-5082	空指针。	const int OB_ERR_NULL_POINTER
-5090	为表添加排它锁失败。	const int OB_ERR_TABLE_EXCLUSIVE_LOCK_CONFLICT
-5092	为表添加意向排它锁失败。	const int OB_ERR_TABLE_INTENTION_LOCK_CONFLICT
-5602	get_tablets_ranges()获取的是tablet，进行局部索引表排序。	const int OB_GET_TABLETS
-5603	get_tablets_ranges()获取的是range，进行全局的索引表构建。	const int OB_GET_RANGES
-5604	get_tablets_ranges()获取不到任何东西，不进行索引构建。	const int OB_GET_NOTHING
-5999	SQL结束错误。	const int OB_ERR_SQL_END
-6001	存储过程不存在。	const int OB_ERR_SP_DOES_NOT_EXIST
-6002	对于存储过程，参数数目不正确。	const int OB_ERR_SP_WRONG_NO_OF_ARGS
-6003	结果有一个以上的行组成。	const int OB_ERR_TOO_MANY_ROWS
-6004	参数重复。	const int OB_ERR_SP_DUP_PARAM
-6005	变量重复。	const int OB_ERR_SP_DUP_PARAM
-6007	存储过程已经存在。	const int OB_ERR_SP_ALREADY_EXISTS
-6008	删除存储过程失败。	const int OB_ERR_SP_DROP_FAILED
-6009	创建存储过程失败。	const int OB_ERR_SP_STORE_FAILED
-6010	存储过程引用了未初始化的变量。	const int OB_ERR_SP_UNINIT_VAR
-6011	存储过程不能再给定的场景下返回结果集。	const int OB_ERR_SP_BADSELECT
-6012	存储过程中有不允许的语句。	const int OB_ERR_SP_BADSTATEMENT
-6013	存储过程中使用了未声明的变量。	const int OB_ERR_SP_UNDECLARED_VAR

-6014	对于CASE语句，未发现case。	const int OB_ERR_SP_CASE_NOT_FOUND
-6015	存储过程的out或inout参数不是变量。	const int OB_ERR_SP_NOT_VAR_ARGS
-6017	未知的存储过程。	const int OB_ERR_UNKNOWN_PROCEDURE
-6018	对于存储过程，参数计数不正确。	const int OB_ERR_WRONG_PARAMCOUNT_TO_PROCEDURE
-6019	对于存储过程，参数不正确。	const int OB_ERR_WRONG_PARAMETERS_TO_PROCEDURE
-6020	不良的SQLSTATE。	const int OB_ERR_SP_BAD_SQLSTAT
-6021	不正确的存储过程名。	const int OB_ERR_SP_WRONG_NAME
-6022	不能在存储过程中间插入声明语句。	const int OB_PROCEDURE_DECLARE_ERROR

Cedar 0.2 系统术语参考指南

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-08-01	Cedar 0.2 系统术语参考指南	钱于引	

术语	含义
RootServer	主控服务器，提供服务器管理和集群管理的功能。一般与UpdateServer部署在同一台服务器中。
UpdateServer	更新服务器，存储Cedar系统的增量更新数据，是Cedar中唯一的写入模块。一般与RootServer部署在同一台服务器中。
ChunkServer	基准数据服务器，存储Cedar系统的基准数据。一般与MergeServer部署在同一台服务器中。
MergeServer	合并服务器，接收并解析用户的SQL请求，经过词法分析、语法分析、查询优化等一系列操作后转发给相应的ChunkServer或者UpdateServer。一般与ChunkServer部署在同一台服务器中。
LMS（Listener MergeServer）	Cedar集群内部特殊的MergeServer进程，只负责从集群的内部表中查询主备集群的流量分布信息和所有的其他MergeServer的地址列表。一般与RootServer部署在同一台服务器中。
表	一个表由若干列（在schema中定义）和任意行组成，有些表的每一行都存储了schema中定义的每一列，这样的表是稠密的，如基准数据的表一般是稠密的；另外一些表行只存储了部分的列，例如UpdateServer中的表的修改增量，这样的表是稀疏的。
Commit Log	操作日志。UPS每次更新操作会产生一条commit log，并追加到已有的commit log之后，并通过同步commit log实现主备复制。完整的commit log序列代表了Cedar的更新历史。
CS	ChunkServer，基准数据服务器。存储Cedar数据库中的基准数据，提供数据读取服务、执行定期合并以及数据分发。
DIO	Direct Input-Output，直接输入输出。
行	一个行由若干列组成，有些时候其中的部分列构成主键（row key）并且整个表按主键顺序存储。有些表（如select指令的结果）可以不包含主键。
基准数据	Cedar存储的某个时间点以前的数据快照，作为静态数据以SSTable的格式存放在ChunkServer上。
LMS	Listener MergeServer。只负责从集群的内部表中查询主备集群的流量分布信息和所有的其他

	MergeServer的地址列表。
列	一个列由列ID(column_id)及其值(column_value)组成。
MS	MergeServer， 合并服务器。主要提供协议解析、SQL解析、请求转发、结果合并和多表操作等功能。
MVCC	Multi-Version Concurrency Control，多版本并发控制。
慢查询	超过指定时间的SQL语句查询。
OB	OceanBase，阿里巴巴集团研发的支持海量数据的关系型数据库系统。
OLAP	On-Line Analytical Processing，联机分析处理。
OLTP	On-Line Transaction Processing，联机事务处理。
Remote Process Call	远程方法调用。
RS	RootServer，主控服务器。主要进行集群管理、数据分布和副本管理，并提供Listener服务。
schema	表的列的类型、值范围等以及该表与其他表的join等关系称为表的schema。
UPS	UpdateServer，更新服务器，是集群中唯一能够接受写入的模块，存储每日更新的增量数据。
UPS Master	UpdateServer Master，主UpdateServer。主要处理实际读写请求。
UPS Slave	UpdateServer Slave，备UpdateServer。主要用于实时备份。
VIP	Virtual IP，虚拟IP地址。RootServer主机由VIP决定。
增量数据	Cedar存储的某个时间点以后的更新数据，存放在UpdateServer上，在内存中以btree和hash table的形式组织。

应用局限性及展望

- Cedar 0.2版本不支持多库，Cedar没有Database的概念，一个Cedar集群只有一个Database。
- Cedar 0.2版本不支持decimal、numeric数据类型。
- Cedar 0.2版本支持游标（cursor），但是在存储过程中，cursor不可用。

期待在Cedar未来的版本中，支持的功能会越来越丰富。