# arkit安装与使用

## 安装

Arkit是一款MySQL的插件，所以安装起来非常简单，只需要拿到arkit.so包，然后将其放置在MySQL参数plugin_dir所指的目录中，然后通过install plugin命令安装即可。arkit.so中包含了审核所需要的所有插件，下面是每个插件的详细信息：

| 包含的所有插件名称 | 插件说明 |
| --- | --- |
| arkit | 审核功能的主体插件，安装之后，可以通过show variables like "%ark%";命令看到所有包含的变量 |
| arkit_order_queue | information schema插件，这个插件主要是用来观察arkit内部的队列状态，包括队列长度，处理的任务数，等待状态的任务数 |
| arkit_osc_status | information schema插件，这个插件用来观察arkit正在执行的osc任务的状态信息 |
| arkit_processlist | information schema插件，这个插件用来观察arkit正在执行的所有任务的状态信息 |

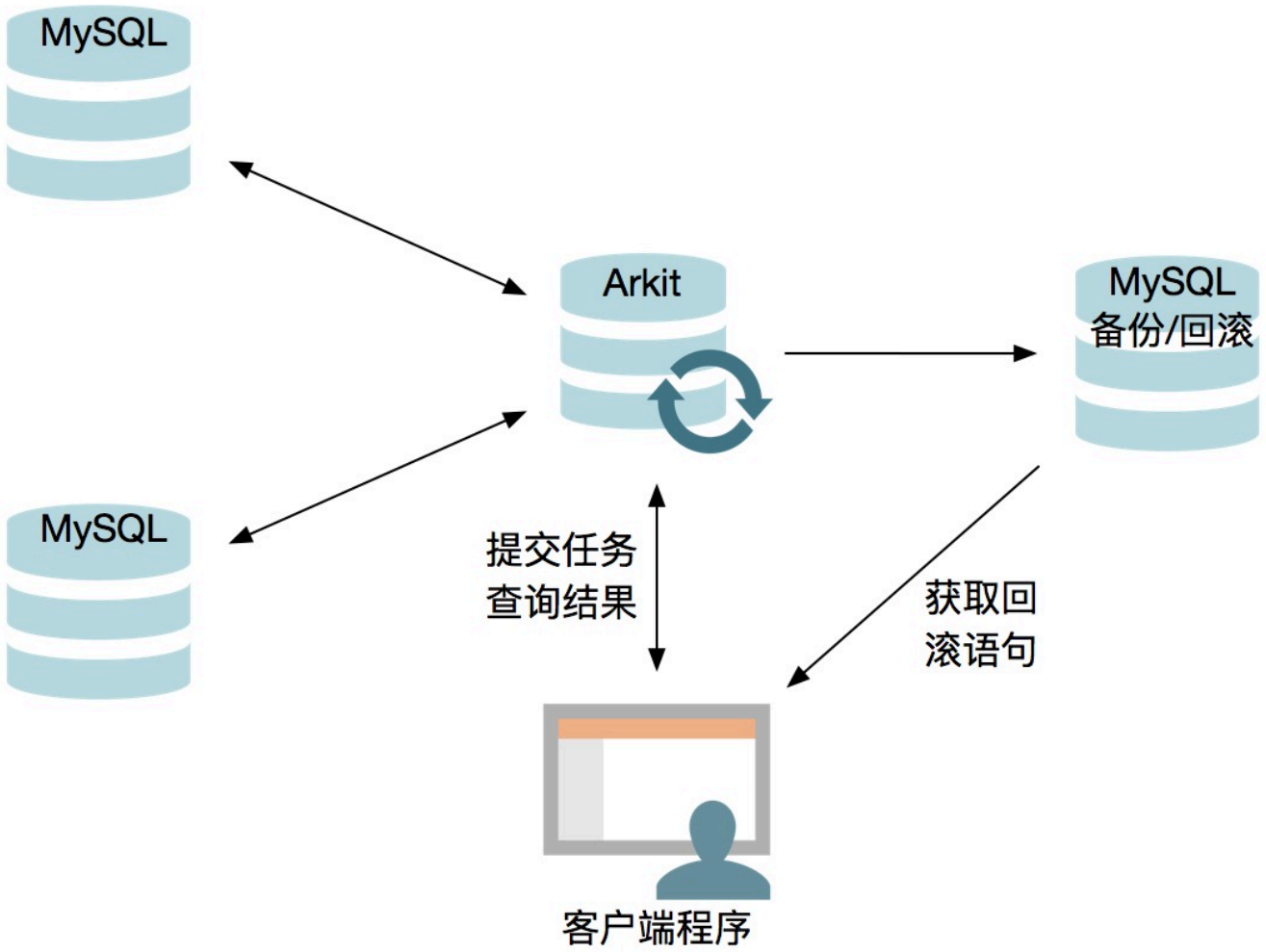
可以在MySQL客户端中通过下面的命令来执行插件的安装：

```
install plugin arkit soname 'arkit.so';
install plugin arkit_order_queue soname 'arkit.so';
install plugin arkit_osc_status soname 'arkit.so';
install plugin arkit_processlist soname 'arkit.so';
```

安装插件arkit之后，就可以通过show variables like "%arkit%";命令，查看所有arkit支持的配置参数。具体每个配置参数的详细功能，后面专门展示出来。
后面四个都是information schema插件，在安装之后，可以在information_schema库中看到这些表。可以直接用SQL语句查询即可，也可以看其表结构。

安装完之后，安装了arkit的MySQL服务器，就变身为一个审核工具了，当然他还可以是一个普通的MySQL服务器。

## 架构说明




从整体上来看，架构非常简单，只需要在一个MySQL Server中安装Arkit插件，这个MySQL Server就变为一个专业的自动化运维程序了，在客户端程序提交了审核/执行任务之后，客户端程序就可以返回了。之后，Arkit会将这个任务分发给审核线程，审核线程通过语法分析、语义分析，找到语句中的所有信息，将相关的库表信息从线上MySQL中获取过来，根据表结构及Arkit的配置参数进行逐个审核，而此时，客户端程序可以不断地查询information schema中的状态进度表，了解审核/执行进度，在页面上展示出来。当任务执行完成之后，他会自动将审核/执行结果存储到本地Arkit所在的数据库中，此时客户端程序可以了解到这个任务已经执行完成了，就可以去相应的结果集表中查询相应任务的结果，从而了解审核详细情况。

当然，如果当前任务是一个执行请求，在任务执行完成之后，相应的回滚语句也已经被存储到了后端MySQL数据库中，这个数据库专门用来存储回滚语句及任务的执行情况，用来记录每一个任务，做到故障之后有踪可寻。

# 使用方法

arkit在使用方面，特别简单，只需要配置相应的参数，就可以使用起来，就可以提交任务了，然后执行任务会在后台运行，可以通过arkit_progress等插件来查看执行进度，执行完成之后，就

可以通过查询本地数据库表查询具体执行结果了。非常简单。

我们首先了解一下提交任务时需要的信息。arkit对语句进行审核时，必须要告诉arkit这些语句对应的数据库地址、数据库端口及arkit连接数据库时使用的用户名、密码等信息，而不能简单地只是执行一条SQL语句，所以必须要通过某种方式将这些信息传达给arkit。而我们选择的方式是，为了不影响语句的意义，将这些必要的信息都以注释的方式放在语句最前面，也就是说所有这些信息都是被“/**/”括起来的，每一个参数都是通过分号来分隔，最终通过设置参数arkit_sql_statement来提交任务。

```
set arkit_sql_statement=
'/*--user=username;--password=xxxx;--host=127.0.0.1;
--port=3306;--enable-check;--sequence=xxx*/
use db1;
select * from table1;
insert into table2 values(1,1);'
```

从设置的参数值来看，前面包括了要提交任务的基本信息，后面紧跟着是每条要执行或者审核的SQL语句，这样组成一个字符串之后，执行之后，就将这个任务提交了。当然，支持的参数不止是这几个，后面还会介绍一些其他的参数。

# License验证

arkit必须要有License验证才可以正常使用。如果没有正确设置License文件的话，会有如下报错信息：

```
mysql> set global arkit_sql_statement = 'xx';
ERROR 6986 (HY000): License is invalid or expire this product is disabled, please c
heck and update the license.
mysql>
```

在每次提交任务的时候，arkit都会去验证License的，所以在使用前必须要设置完成License文件的位置，通过设置参数arkit_license_file来完成。如果所使用的License是非法的，或者已经过期的，在设置的时候就会报错，而如果是正确的，也在有效期内，就会正常设置完成。

# arkit支持选项

前面讲述了关于arkit的基本使用方法，在一些例子中已经看到了部分选项。不同的选项，对应的arkit功能也是不同的，相应的arkit行为与结果都会有所不同。那么，本章将会详细讲述arkit目前

所支持的每一个选项及其各自的意义。

# 选项说明

选项是在SQL语句块前面以注释的方式一起传达给arkit的，它支持的选项很丰富，这里将详细介绍现在所支持的每个选项的使用方法及其意义。

| 参数名称及缺项名称 | 必选 | 可否无值 | 功能描述 |
| --- | --- | --- | --- |
| --host | 是 | 否 | 需要操作的这部分语句块对应的数据库地址。指定方式可以是IP地址、机器名或DNS域名等，只要能唯一解析到这个机器即可 |
| --port | 是 | 否 | 与上面的--host选项对应，指定机器名，必然要再指定一个MySQL实例的端口，那就通过--port来完成 |
| --user | 是 | 否 | 已经确定了MySQL实例，那就需要指定如何连接这个实例，通过--user指定用户名，且这个用户必须要在上面指定的后端MySQL Server上有相应的权限，在前面已经对需要什么权限做过说明 |
| --password | 是 | 否 | 已经确定了MySQL实例，那就需要指定如何连接这个实例，通过上面的--user指定了用户名，那么--password所指定的就是这个用户名对应的密码，必须要以明文的方式来指定 |
| --sleep | 否 | 否 | 这个参数用来指定在执行完每一条语句后，暂停多少毫秒，这样可以适当控制对线上数据库的冲击，特别是针对大量写入的操作，单位为毫秒，最小值为0，也就是不暂停，最大值为100秒，也就是100000毫秒。如果设置得超过100000毫秒，arkit会自动将其设置为100000毫秒。这个参数可以和其他参数一起设置，但是只有在--enable-execute为1的情况下，才起作用 |
| --check | 否 | 否 | 告诉arkit当前要做什么操作，是审核还是执行，这个参数与下面的--enable-execute只能指定一个，功能如其名，--enable-check就是告 |

| | | | |
|---|---|---|---|
| | | | 诉arkit，当前的请求是要做审核操作，审核完成就返回结果集 |
| --execute | 否 | 否 | 告诉arkit当前要做什么操作，是审核还是执行，这个参数与上面的--enable-check只能指定一个。如果指定的是选项--enable-execute，则arkit在执行前还会做一次实时的审核，这个审核和前面指定--enable-check时的审核基本是相同的，只是这次在审核完成之后，还会继续执行。因为相同的语句在不同的时间审核有可能会产生不同的审核结果（环境有可能变了），所以有必要再做一次审核。如果审核发现了错误（而不是警告），就不会被执行，此时会提前返回告知错误，如果审核时发现的是警告，并且没有指定--enable-ignore-warnings（下面会介绍），则有警告也不会执行，而如果想跳过这些警告，则可以通过下面这个选项来设置 |
| --ignore-warnings | 否 | 否 | arkit采取严格的分阶段处理方式，先对所有语句进行审核，审核完成之后，会执行所有语句，之后才会去做所有语句的备份操作。在三个阶段的过渡过程中，如果审核有问题则不会继续执行，此时如果人为确定想要跳过这些警告，也就是需要忽略它们，则可以选择这个参数，告诉arkit跳过这个警告的检查，继续执行 |
| --force | 否 | 否 | 在使用arkit的时候，还经常会遇到一个问题，那就是批量导入数据时，执行某些语句时有可能会报出主键冲突的问题，而DBA可以确定的是，出现主键冲突不是问题，可以继续执行，那么此时就可以通过选项--enable-force告诉arkit，在执行过程中碰到一个错误时，可以先保存错误信息并继续执行下一条语句。这个参数要谨慎使用 |
| --remote-backup | 否 | 否 | arkit支持备份并生成对应的回滚语句，这是默认的，但当有些影响行数很多且明确不需要回滚的时候，为了提高执行效率，可以指定在执行时不做备份，指定方式是通过disable来禁用 |

| | | | |
|---|---|---|---|
| | | | 它。这种方式是MySQL配置文件禁用某个选项时常用的方法，即--disable-remote-backup。关于备份的具体内容，在47章中会有专门介绍 |
| --split | 否 | 否 | 这个参数用来拆分要执行的语句块。如果在语句块中存在对同一个表的DDL操作及DML操作，那么在分析Binlog来生成回滚语句时，由于表结构已经发生改变，会导致arkit没有办法处理，所以使用这个参数将这些语句分成多批，然后再分别执行。这是在执行前必须要做的一个操作，不然可能会产生不可预知的错误。当然在执行前的最后一次审核中，如果检查到这样的混用情况，则会返回报错，而不是警告。这个参数指定之后，除了前面四个参数之外，其他参数都被忽略，也可以不指定 |
| --query-print | 否 | 否 | 这个参数用来打印SQL语句在被arkit分析之后的执行树结构，以JSON的形式提供，目的是为了可以在arkit的基础上，对已经结构化的（JSON）语句做再次分析，可以对arkit内置支持的规则进行扩展，做个性化定制，比如使用到哪些列、哪些语句类型等信息。目前支持的语句类型有插入、删除、更新及查询，详情请参考52章中“打印语法树”一节 |
| --sequence | 是 | 否 | 这个值用来唯一标识一个任务，是一个字符串，上层应用程序最好通过任务信息生成一个HASH值，来保证唯一性，如果不唯一，arkit会检查出来并且报错，导致提交失败，这个值提交之后，会作为这个任务的唯一标识，一直被使用，比如停止任务、查询任务进度状态等等。 |

# DDL与DML语句分离

首先要说明的一点是，这一节是对上面所述选项“--split”的进一步阐述。

在执行时，不能将 DML 语句及 DDL 语句放在一起执行，否则会由于表结构的变化而导致备份解析Binlog时出现不可预知的错误，如果要同时执行 DML 及 DDL，则需要分开多个语句块来执行，如果执行时一个语句块中同时包含了这二者，则arkit会报错，不会去执行。

arkit支持将一段SQL语句按照语句之间相互不影响的原则把DDL和DML语句分开，也就是让相同表的DDL和DML语句不在同一个语句块中执行。

与其说这是一个功能，不如说这是一个必经流程，为什么我们要实现这样一个功能呢？原因是，如果在同一个块中同时执行DDL和DML的混合语句体，那么arkit对表的缓存结构在变化之后（因为表的修改导致的），就会导致不能正确地解析修改之前的Binlog记录，从而导致备份出问题，所以arkit采取了一个简单折中的办法来处理这个问题。

这个功能的用法是通过指定另一个选项来实现的，这个选项为`--split`。在指定这个选项之后，所有其他选项（除了指定线上地址的4个选项之外）都不起作用，只处理split选项。

在提交任务之后，通过查询表arkit_progress来判断当前split任务是不是完成了，如果完成，则可以在本地查询表arkit.arkit_split来获取拆分结果。

经过这样的处理，就可以放心地将每一行单独作为一个arkit任务提交了，此时可以直接以--execute类型的任务提交。

所以从这里可以看出，对于执行一个任务而言，一个完整的流程应该是如下这样。

- 流程一：enable-check。
- 流程二：enable-split。
- 流程三：enable-execute(s)。 流程三中最后的s的意思是，被split处理过之后，有可能需要分多次来执行。

知道这两种语句拆分的用法之后，应该还会想知道在内部是如何拆分开的。其实，在知道拆分的目的之后，拆分原则也应该猜到一二了。

这个拆分原则是，通过分析，按照在语句块中的出现顺序，对库表及操作类型进行统计，在扫描的过程中，从某一个表中第一次出现DDL和DML共存情况的这条语句开始，重新生成一个语句块，而这条语句之前的内容就作为一个完整的语句块。产生新的语句块之后，继续向后扫描，直到处理完所有的语句为止，扫描结束之后，分成多少块，那就是split的结果集中有几行。

很明显，如果一个语句块中的所有语句涉及的表都不相同，或者是涉及的不是同一个库下面的表，又或者涉及了相同的表但都是DDL或DML等，那么就都不会被拆开。经过split处理之后，输出结果集其实还是只有一行。这有点类似一条流水线，可以称之为“合格检查”，同一个表的DDL和DML语句共存，就是不合格，除此之外，都是“合格产品”。

在拆分过程中，还有一点需要注意，因为其中会涉及环境变量的问题，所以拆分过程不能简单地把所有语句机械地从物理上分开，每个语句还是要依赖于环境变量，比如db信息、charset信息等。所以arkit在拆分过程的每一段中，都会将它们的环境变量继承过来，保证逻辑上不会有问题。

# 小技巧

可以看到，上面所说的参数名除了前面四个连接选项之外，都在前面加了enable。实际上，这些参数的写法与MySQL配置参数的写法是相同的，比如--enable-check，这个参数的名字只是简单的check，可以写成--check=1，也可以写成--disable-check，都是比较灵活的，其他的也都是一样的道理。

从上面列出的选项来看，做任何操作时，有五个选项是必须要有的，分别是--host、--port、--user、--password及--sequence。

而有一些选项之间是互斥的，也就是不可以同时出现的。这些选项分别是--enable-check、--enable-execute、--enable-split和--enable-query-print。这也说明，arkit目前从选项上来说支持这四个功能。

还有一些选项，是用来辅助执行的，分别如下。

- --sleep，执行间隔，辅助的选项是--enable-execute。
- --enable-ignore-warnings，忽略警告，辅助的选项是--enable-execute。
- --enable-force，跳过错误，辅助的选项是--enable-execute。
- --enable-remote-backup，备份，辅助的选项是--enable-execute。

这样归类之后，对arkit的功能应该就比较清楚了，也可以知道在什么情况下该用什么选项了。

# 任务状态

arkit的设计，都是采用了提交任务，然后异步查询状态及结果这种方式，所以操作方式就非常明了了，先通过设置参数arkit_sql_statement提交任务，然后查询arkit.arkit_progress表中相应的任务状态。如果发现执行完了，则再去本地数据库表arkit.arkit_feedback中找到相应的执行结果即可。那这里就介绍一下每一个表的详细信息。

## information_schema表

### arkit_osc_status

```
CREATE TEMPORARY TABLE `arkit_osc_status` (
  `Task_Id` longtext NOT NULL,
```

```
  `Db_Name` longtext NOT NULL,
  `Table_Name` longtext NOT NULL,
  `Command` longtext NOT NULL,
  `SQLSHA1` longtext NOT NULL,
  `Percent` int(11) unsigned NOT NULL DEFAULT '0',
  `Remain_Time` longtext NOT NULL,
  `Information` longtext NOT NULL,
  `Execute_Time` longtext NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

在使用osc（包括build_in_osc和pt-online-schema-change）改表时，具体的改表信息可以通过这个表来查询，具体的每个列的详细信息如下：

1. Task_Id，所属任务的Sequence。
2. Db_Name，被修改表所在的数据库的名称。
3. Table_Name，被修改表的名称。
4. Command，改表语句。
5. SQLSHA1，用来唯一标识一条改表语句的SHA1，在审核之后的结果集中获取。
6. Percent，表示当前改表已经执行的百分比。
7. Remain_Time，表示当前改表还需要多少时间才能完成，单位是秒。
8. Information，表示当前改表语句在改表过程中输出的详细信息。
9. Execute_Time，表示当前改表任务已经执行了多少时间，单位是秒。

这条表的功能是输出所有当前正在使用OSC执行的操作，如果在同一个arkit请求中有多条ALTER语句，那么显示出来的就有可能存在执行进度为100%的语句，通过这条语句，可以轻松查看当前每一个OSC的执行进度，如果进度一直不前进（比如存在从库复制延迟，导致OSC长时间waiting），则在这里可以看到具体信息。

## arkit_order_queue

```
CREATE TEMPORARY TABLE `arkit_order_queue` (
  `Id` int(11) unsigned NOT NULL DEFAULT '0',
  `Enqueue_Index` int(11) unsigned NOT NULL DEFAULT '0',
  `Dequeue_Index` int(11) unsigned NOT NULL DEFAULT '0',
  `Task_Count` int(11) unsigned NOT NULL DEFAULT '0',
  `Total_Count` int(11) unsigned NOT NULL DEFAULT '0'
) ENGINE=MEMORY DEFAULT CHARSET=utf8
```

arkit内部有多个执行队列，每一个执行线程对应的一个队列，队列大小通过参数arkit_worker_queue_size来控制，线程个数通过arkit_parallel_workers来控制，有多少个线程，在arkit_order_queue表中就有多少条记录，这个表主要是用来查询队列的执行情况的。每个列的

意义如下：

1. Id，一个序号而已。
2. Enqueue_Index，队列的入队下标。
3. Dequeue_Index，队列的出队下标。
4. Task_Count，表示当前队列中有多少个任务。
5. Total_Count，表示当前线程总共处理过多少个任务。

## arkit_processlist

```
CREATE TEMPORARY TABLE `arkit_processlist` (
  `Id` int(11) unsigned NOT NULL DEFAULT '0',
  `Host_Name` longtext NOT NULL,
  `Port` int(11) unsigned NOT NULL DEFAULT '0',
  `User_Name` longtext NOT NULL,
  `Backup` longtext NOT NULL,
  `Force` longtext NOT NULL,
  `Ignore_Warnings` longtext NOT NULL,
  `Sleep_Nms` bigint(21) unsigned NOT NULL DEFAULT '0',
  `Command` longtext NOT NULL,
  `State` longtext NOT NULL,
  `Time` int(11) unsigned NOT NULL DEFAULT '0',
  `Info` longtext NOT NULL,
  `Current_Execute` longtext NOT NULL,
  `Progress` longtext NOT NULL,
  `Current_DB` longtext NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

arkit_processlist表想要展示的内容是当前所有正在执行的任务的详细信息，具体每一个列的意义如下：

1. Id，一个序号。
2. Host_Name，表示当前任务执行或者审核的线上数据库地址。这个与提交时，--host参数指定的内容一致。
3. Port，与上面列Host_Name对应，这里表示的是数据库端口。
4. User_Name，表示arkit连接Host_Name：Port所表示的数据库时所用的用户名。
5. Backup，对应提交任务时的--remote-backup参数的值。
6. Force，对应提交任务时的--force参数的值。
7. Ignore_Warnings，对应提交任务时的--ignore-warnings参数的值。
8. Sleep_Nms，对应提交任务时的--sleep参数的值。
9. Command，表示当前任务的操作类型，包括Check、Execute、Split、Print等值。
10. State，表示当前任务的执行阶段，包括INIT、CHECKING、EXECUTING、DEINIT、

BACKUP等值。

11. Time，表示当前任务已经执行的时间值。
12. Info，表示当前任务执行的所有语句。
13. Current_Execute，表示当前任务正在执行的语句。
14. Progress，如果Command为Execute，表示当前任务中，正在执行第几条语句，格式是“当前语句序号/总的语句数”，如果Command为Check，则只表示当前已经审核了多少条语句了。
15. Current_DB，表示当前处理的语句所在的数据库名。

# 本地arkit数据库

## arkit_progress

```
CREATE TEMPORARY TABLE `arkit_progress` (
  `Task_Id` longtext NOT NULL,
  `Sequence` int(11) unsigned NOT NULL DEFAULT '0',
  `Total` int(11) unsigned NOT NULL DEFAULT '0',
  `Status` longtext NOT NULL,
  `Update_Time` timestamp(4) NOT NULL DEFAULT '0000-00-00 00:00:00.0000',
  `Error_Code` int(11) unsigned NOT NULL DEFAULT '0',
  `Error_Message` longtext NOT NULL,
  `Db_Name` longtext NOT NULL,
  `Complete` longtext NOT NULL,
  `Warnings` text,
  PRIMARY KEY (`Task_Id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

这个表用来查询当前正在执行或者已经执行完成的任务进度，每个列的意义如下：

1. Task_Id，对应的是任务sequence。全局唯一。
2. Sequence，表示当前任务正在执行或者审核第几条语句，如果为0，则表示已经完成或者还未开始。
3. Total，表示当前任务总共有多少条语句，在审核的过程中，这个值会不断增加，在执行时候，这个值表示的就是真实的语句数目。
4. Status，表示当前正在处理的语句（Sequence所表示的语句）的状态，如果准备执行了，它的值是Prepare，如果已经执行完成，它的值是Done，如果出错了，它的值是Error，如果任务初始化出错，则它的值是Init_Error，如果审核任务还未开始，在等待线程调度时，它的值为Waiting_Enqueue。
5. Update_Time，表示当前任务最新更新的时间。
6. Error_Code，如果Status为Error，则这个值表示出错时的MySQL错误码。

7. Error_Message，如果Status为Error，则这个值表示出错时的MySQL错误信息。
8. Db_Name，表示Sequence对应的语句所在的Db。
9. Complete，表示是不是已经结束。
10. Warnings，表示在执行过程中，产生的警告信息。

arkit中的是持久化的，即使数据库挂了，再启动之后数据也可以查到，这样做的好处是，可以通过这个表查询在挂之前的任务状态，用于判断任务的后续处理方式。

## arkit_feedback

```
CREATE TABLE `arkit_feedback` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `task_id` varchar(128) NOT NULL COMMENT 'task sequence',
  `sequence` int(11) NOT NULL DEFAULT '0' COMMENT 'sequnce in one task',
  `stage` varchar(32) NOT NULL COMMENT 'what stage',
  `error_level` int(11) NOT NULL DEFAULT '0' COMMENT 'err_level',
  `stage_status` varchar(64) NOT NULL COMMENT 'stage detail',
  `error_message` text NOT NULL COMMENT 'errmeg in this sql',
  `sql_statement` longtext NOT NULL COMMENT 'sql statement',
  `affected_rows` bigint(20) NOT NULL DEFAULT '0' COMMENT 'affected_rows',
  `backup_sequence` varchar(64) NOT NULL COMMENT 'backup sequnce',
  `backup_dbnames` varchar(1024) NOT NULL COMMENT 'dbnames in json',
  `execute_time` varchar(64) NOT NULL COMMENT 'execute time',
  `sqlsha1` varchar(128) NOT NULL COMMENT 'osc hash',
  `command` varchar(32) NOT NULL COMMENT 'sql command type',
  `create_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_taskid_sequence` (`task_id`,`sequence`),
  KEY `idx_create_time` (`create_time`)
) ENGINE=InnoDB AUTO_INCREMENT=35 DEFAULT CHARSET=utf8
```

这个表用来存储所有--check和--execute完成之后的结果的，表中每一个列的详细信息如下：

1. **ID**：用来表示结果集中记录序号的，也就是被审核的语句在语句块中的序号，按位置排序，计数从1开始。
2. **task_id**：用来存储当前语句属于哪一个任务。
3. **sequence**：用来存储当前语句在所属任务中的序号，从1开始计数。
4. **stage**：这一列显示当前语句已经进行到哪一步了，总共包括四个值，分别是CHECKED、EXECUTED、RERUN和NONE。NONE表示没有做过任何处理，有可能前面有语法错误直接提前返回了；CHECKED表示这个语句只做过审核，而没有再进行下一步操作；EXECUTED表示已经执行过，如果执行失败，也是用这个状态表示；RERUN表示的是，对于影响上下文的语句，已经执行成功，但为了与EXECUTED区分，用RERUN表示，主要是因为在执行

过程中，如果某一条语句执行失败了，则上层可能需要将没有执行的语句提取出来，再次执行，那么影响上下文的语句就是需要加上的，所以用RERUN来表示。arkit目前支持两种影响上下文的语句，分别是set names charsetname语句和use database语句。

5. **error_level**：该列目前总共有三个值，分别是0、1、2。如果为0，则说明当前语句审核没有任何问题；如果值为1，则说明当前语句审核时发现有些写法不符合arkit定义的内置规则，属于警告；如果值为2，则说明当前语句审核时，发现了严重错误，是无论如何都不能通过的，有这种错误时，如果想要做的是--enable-execute，即使打开--enable-ignore-warnings，也没有用。分等级的设计初衷是，对于规则的设定，arkit的执行是死的。但有时候由于业务的特殊需要，不符合规则也是可以通过的，即警告是可以被忽略的，具体过程可以由DBA和业务开发人员商议决定。所以在实现自动化运维平台时，一个比较好的实现方式是，如果审核结果中没有2，但有1，开发同学可以提交成功，但经过DBA审核之后，如果认为不能通过，那么就可以打回修改；而对于存在2的情况，则可以在平台上直接被限制提交，必须是在开发同学修改过之后，才能提交成功。这种方式增大了平台的容错率，考虑到了业务环境的复杂性及特殊性。
6. **stage_status**：该列用来描述当前语句的阶段结果，与列stage对应。如果是审核阶段，并且完成，则返回 Audit completed。如果是执行阶段，并且执行成功则返回Execute Successfully，否则返回Execute failed。如果是备份阶段，并且备份成功，则在执行描述信息后面追加Backup successfully，否则追加Backup failed。这一列的返回信息是为了将结果集直接输出而设置的，如果在具体的使用过程中，为了更友好地显示，则可以在此基础上再做加工处理。
7. **error_message**：用来表示出错的错误信息，这里包括一条语句中的所有错误信息，用换行符分隔，但有时候如果某一个错误导致不能继续分析了，比如表不存在等问题，在这种情况下，如果语句还有其他错误，就不能被审核出来了。如果当前语句没有任何错误，则这个列的值为None。对于执行及备份操作，因为对于一条语句，这样的错误只会有一次，那么执行错误会在后面追加“execute:具体的执行错误原因”，如果是备份出错，则在后面追加“backup:具体的备份错误原因”。在执行时，有时候还会出现Warnings，比如插入数据时字符串被截断等，此时会在后面输出这些warnings，输出格式是`#1 Execute(Warning, Code errno):warning message`，#号后面的数字表示第几个警告，因为有时候执行一条语句会产生多个警告。
8. **sql_statement**：用来表示当前检查的是哪条SQL语句，这一列所存储的值就是这条SQL语句的文本内容。如果某一条SQL语句在检查时有语法错误，则这里会包括从出错语句开始到后面所有的语句。因为语法出错后，arkit就不能再继续分析了，也就不能将后面的每条语句分开了。
9. **affected_rows**：审核时，用来表示当前语句预计影响的行数，这个行数一般是通过EXPLAIN来获取的，但在MySQL 5.5及以下版本中，只支持SELECT，所以对于增删改语句的影响行数，这一列的值有可能就是0了。在执行时，该列输出的是执行时真实影响的行数。
10. **backup_sequence**：该列与arkit备份功能有关，其实就是与`arkit_execution_backup`表中

的列opid_time一一对应，这就为自动化运维平台针对某一条语句做回滚操作找到了入口，每次执行都会产生一个序号，如果要回滚，就使用这个值从备份表中找到对应的回滚语句执行即可。

11. **backup_dbnames**：该列表示的是当前语句产生的备份信息，存储在备份服务器的哪些数据库中，这是一个JSON串的值，可能包括多个数据库，当然只针对需要备份的语句才有意义，数据库名由IP地址、端口、源数据库名组成，由下划线连接，而如果是不需要备份的语句，则返回字符串None，提供这个信息，主要是为了让运维平台方便，不需要自己拼接这个库名了。
12. **execute_time**：该列表示当前语句的执行时间，单位为秒，精确到小数点后两位。列类型为字符串，使用时可能需要转换成DOUBLE类型的值，如果只是审核而不执行，则该列返回的值为0。
13. **SQLSHA1**：这一列用来存储当前这条语句的一个HASH值，用来标识这个语句是否会使用OSC功能，如果返回信息中有值，则表示这条语句在执行时会使用OSC。因为在执行前，会有一次单独的审核操作，此时上层已经可以拿到了这个值，审核通过之后，语句是不会改变的，当然这个值也不会改变，那么在执行时就可以使用这个值来查看OSC执行的进度等信息。这个HASH值一般的样子如下：
    *__D0210DFF35F0BC0A7C95CD98F5BCD4D9B0CA8154__，其他具体信息，请参考52章中“对OSC的支持”一节的详细讲述。
14. **command**：表示当前语句的语句类型，包括INSERT、DELETE、UPDATE及DROP等，可以通过这个列才做一些语句类型的判断，识别出一些危险的语句类型，可以给出相应提醒。

## arkit_print

```
CREATE TABLE `arkit_print` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `task_id` varchar(128) NOT NULL COMMENT 'task sequence',
  `sequence` int(11) NOT NULL DEFAULT '0' COMMENT 'sequnce in one task',
  `sql_statement` longtext NOT NULL COMMENT 'sql statement',
  `error_level` int(11) NOT NULL DEFAULT '0' COMMENT 'err_level',
  `query_tree` longtext NOT NULL COMMENT 'sql statement',
  `error_message` text NOT NULL COMMENT 'errmeg in this sql',
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_taskid_sequence` (`task_id`,`sequence`)
) ENGINE=InnoDB AUTO_INCREMENT=19 DEFAULT CHARSET=utf8
```

这个表用来存储--query-print执行完成之后的结果，表中每一个列的结果如下：

- ID：用来表示当前语句的一个序列值。
- task_id：用来存储当前语句属于哪一个任务。
- sequence：用来存储当前语句在所属任务中的序号，从1开始计数。

- sql_statement：用来存储当前被分析的SQL语句。
- error_level：用来存储当打印遇到问题时错误的级别，与审核结果集中的ERRLEVEL意义相同。
- query_tree：当前语句的分析结果，格式为JSON字符串。
- error_message：与上面的error_level对应，当出错时，这里存储分析过程中的所有错误信息，与审核结果集中的同名列意义相同。

## arkit_split

```
CREATE TABLE `arkit_split` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Task_Id` varchar(128) NOT NULL COMMENT 'task sequence',
  `Sequence` int(11) NOT NULL DEFAULT '0' COMMENT 'sequnce in one task',
  `sql_statement` longtext NOT NULL COMMENT 'sql statement',
  `DDL_Flag` int(11) NOT NULL DEFAULT '0' COMMENT 'ddl_flag',
  `SQL_Count` int(11) NOT NULL DEFAULT '0' COMMENT 'SQL_Count',
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_taskid_sequence` (`Task_Id`,`Sequence`)
) ENGINE=InnoDB AUTO_INCREMENT=34 DEFAULT CHARSET=utf8
```

这个表用来存储执行类型为--split的时候的执行结果的，总共有6列，分别如下。

- id：这个表的自增列。
- Task_Id：用来存储这个任务的sequence值。
- Sequence：表示当前行被分成第几个语句块，这个值是从1开始的。
- sql_statement：表示的是可以在一起的所有SQL语句， 都合在一起并且都以分号分开。
- DDL_Flag：第三列表示的是当前被拆分之后，可以一起执行的语句中，有没有 `alter/drop table` 语句，如果有则输出1，否则输出0。这主要是为了避免在arkit执行这两种语句时有可能带来的危险，这样输出之后，可以为上层提供更友好的选择，这个更友好的选择可以作为一个提醒。
- SQL_Count：表示当前被分开的块儿里面，有多少条SQL语句。

如果第一列返回的序号为0，那么此次结果集只会有一行，并且这一行的第二列的值是出错信息，也就是说这次操作是有错误的，具体错误可以直接看到。是不是有错误发生，只需要看序号值是不是为0就可以了。

# 审核规范

今天，业界已经基本普遍认同，在MySQL的应用实践中需要对应用程序使用的SQL语句进行审

核，以确保生产环境中的MySQL数据库服务的正常稳定运行。实际上，对SQL语句审核还会带来其他附加的收益，比如语句风格的统一与规范、SQL的标准化、SQL变更的审计等。建立审核标准是一项非常有意义的工作，而之前的人工审核，针对标准这个问题其实是很吃力的，标准越多，需要记忆和评估的点就越多，DBA越累，开发也越累。arkit出现之后，它是不怕标准多的。只要DBA定义了更好地管理数据库的规则，arkit通过编程实现之后就可以很好地执行，并且返回审核结果。

这一章主要介绍当前arkit在审核时，是使用什么样的规则来做的，以及哪些规则是可以配置的，哪些规则是不可以配置的。这样，针对不同的部门或公司，可以定制不同的规则，配置参数可以参考第49章。

# 支持的语句类型

- use db：此时会检查这个库是否存在，需要连接到线上服务器来判断。arkit分析到这个语句的时候，也是变更环境变量的时候，这条语句之后的所有语句执行都会在这个库下面，除非重新设置或语句中明确指定了库名。
- set option：现在只支持set names charset，设置其他变量时都会报错不支持。这个操作也是一个环境变量设置语句。
- 创建数据库语句：创建数据库，会在arkit内部对这个新数据库做缓存，这样就可以做到在一个语句块中，新建数据库并且在当前语句块中直接使用（而不需要先在线上创建这个数据库），给实际应用带来了非常大的便利性。
- 插入语句：包括多值插入操作，比如insert t1 values(c11, c21), (c12, c22)类似的语句。
- 查询插入语句：这种类型也属于插入语句，只不过后面跟着的是一个SELECT查询，和简单的插入有区别，比如预计影响行数的处理就不同，简单的INSERT可以直接计算出精确的影响行数，而查询插入就做不到这一点。
- 删除语句：包括多表删除语句，主要分析影响行数，表达式的合法性等方面的问题。
- 更新语句：包括多表更新语句，主要分析影响行数，表达式的合法性等方面的问题。
- 创建表语句：包括新表创建及根据一个已经存在的表做LIKE创建。创建表之后，也会被arkit做本地缓存，这样在同一个语句块中，创建之后马上就可以使用，给业务开发同学及DBA带来了很大的便利性。
- 删除表语句：根据实际需求，删除也是有必要的，不过要谨慎行事。
- 修改表语句：改表是常用的需求，这里有可能会使用到我们熟悉的OSC，其支持丰富的改表类型。
- Truncate表语句：和删表的道理一样。
- arkit命令集语句。

# 公共检查项

- 使用到的库、表、列对象是否存在。如果不存在，则会报错并将错误信息加入到对应语句的结果集列中。这种类型的错误，是真的错误，而不是警告，结果集中的errlevel为2。
- 检查创建对象的名字的合法性。如果名字长度大于64个字节，就会报错。包括库名、表名、索引名、列名等。
- 检查标识符的合法性。如果发现名字中存在除数字、字母、下划线之外的字符，则会报错Identifier "invalidname" is invalid, validoptions: [a-z,A-Z,0-9,_]。
- 检查SET OPTION语句类型。目前只支持set names charsetname语句，类型为错误，不可配置。
- 检查创建表、库对象时使用的字符集是否是参数中所定义的，类型为警告，可配置。
- 检查语句类型。如果是Create Index语句，或是Rename Table语句，又或是Drop Index语句，都建议使用Alter Table语句，类型为警告，不可配置。
- 检查同一个表是否存在DDL与DML共存的问题，类型为错误，不可配置。虽然我们在选项中支持--enable-split来解决这个问题，但是为了防止没有做这个操作而出现问题，还是要从安全性层面上检查一次。
- 检查语句中有没有出现MySQL关键字，类型为警告，可配置。
- 检查影响行数是否超过某一个设定的值，类型为警告，可配置。

# 插入语句检查项

- 插入指定的列列表中，同一个列不能出现多次，类型为错误，不可配置。
- 必须指定插入列表，也就是要对哪几个列指定插入值，如insert into t (id，id2) values(...)，类型为警告，可配置。
- 插入列列表与值列表个数相同，二者的个数需要相同，如果没有指定列列表（因为可配置），则值列表长度要与表列数相同。类型为错误，不可配置。
- 检查NOT NULL的列，插入的值是不是NULL，类型为错误，不可配置。
- 检查查询插入时，查询的列个数，与指定列的列表元素个数是否相同，类型为错误，不可配置。
- 检查查询插入语句中的SELECT子句，有没有WHERE条件，类型为警告，可配置。
- 检查查询插入语句中的SELECT子句，是不是“SELECT *”，类型为警告，可配置。
- 检查查询插入语句中的SELECT子句，有没有LIMIT子句，类型为警告，可配置。
- 检查查询插入语句中的SELECT子句中的ORDER BY子句，排列类是否为RAND()，类型为警告，可配置。

# 更新、删除语句检查项

- 检查更新删除语句中的SELECT子句有没有WHERE条件，类型为警告，可配置。
- 检查更新删除语句中的SELECT子句有没有LIMIT子句，类型为警告，可配置。
- 检查更新删除语句中的SELECT子句中的ORDER BY子句，排列类是否为RAND()，类型为警告，可配置。

# 表属性检查项

- 检查表的存储引擎是不是InnoDB，有参数可以配置。
- 检查表的字符集是不是与参数配置匹配，有参数可以设置支持的字符集。
- 检查表是不是有注释，有参数可以设置。
- 检查表是不是分区表，有参数可以配置。
- 检查LIKE建表时，参照的对象表是不是存在，类型为错误，不可配置。
- 如果建立的是临时表，则必须要以tmp为前缀，类型为错误，不可配置。
- 检查创建的列，是否存在重名的列，类型为错误，不可配置。
- 检查新表有没有主键，类型为警告，可配置。
- 检查自增列初始值是不是为1，不是则报警告，可配置。
- 检查表中是不是存在多个主键，类型为错误，不可配置。

# 列属性检查项

- 检查BLOB列是不是设置了默认值，如果是则报警告，无参数可设置。
- 检查列类型是不是DATETIME，并且其默认值为NOW()，如果是则报错，类型为错误，不可配置。
- 检查列设置是不是NOT NULL属性，或者为主键列，但其默认值被设置为NULL，如果是则报错，类型为错误，不可配置。
- 检查自增列是不是设置了默认值，如果是则报错，类型为错误，不可配置。
- 检查时间类型默认值的合法性，如果与对应的字段类型不匹配，则报错，类型为错误，不可配置。关于原则，可以参照MySQL在设置为strict mode时，对应的时间类型如何使用的原则。
- 检查列数据类型是不是SET，或者是ENUM，抑或是BIT，类型为警告，可配置。
- 检查列有没有注释，类型为警告，可配置。
- 检查CHAR类型的字段所定义的长度，如果超过设置，则建议转换为VARCHAR，类型为警告，可配置。
- 检查列的类型是不是BLOB，类型为警告，可配置。

- 检查列有没有设置NOT NULL属性，类型为警告，可配置。
- 检查BLOB类型的列是不是设置了NOT NULL属性，如果设置了则报错，类型为警告，不可配置。
- 检查自增列是不是无符号类型，类型为警告，可配置。如果长度小于4，则同时建议转换为INT或是BIGINT，类型为警告，可配置。
- 检查在MySQL 5.5及以下版本中，是不是存在多个TIMESTAMP类型的列，类型为错误，不可配置。
- 检查DATETIME类型的列，在MySQL 5.5及以下版本中，是不是设置了default CURRENT_TIMESTAMP，或者设置了ON UPDATE CURRENT_TIMESTAMP，如果是则报错，类型为错误，不可配置。
- 检查TIMESTAMP类型的列，是不是没有设置默认值，但设置了ON UPDATE CURRENT_TIMESTAMP，如果是则报错，类型为错误，不可配置。
- 检查不为BLOB类型、不是自增列又不是主键包含的列，是不是设置了默认值，类型为警告，可配置。
- 检查不是BLOB类型的列，是不是显式地单独设置了字符集，类型为警告，可配置。
- 检查自增列名，是否为id，类型为警告，可配置。
- 检查自增列的个数，只能有一个，类型为错误，不可配置。

# 索引属性检查项

- 检查索引是否有名字，如果没有名字则直接报错，属于错误，不属于警告。
- 如果索引类型为PRIMARY，则索引名字必须是“PRIMARY”，否则抛出错误。
- 检查创建的索引是否为外键，可配置，属于警告。
- 检查创建的索引名字是否符合前缀规则，唯一索引需要以“uniq”开头，普通索引需要以“idx_”开头，参数可配置，属于警告。
- 检查索引中列的个数，如果超过参数设置，就报警告。
- 检查主键所包含的列个数，如果超过参数设定数目，则报警告。
- 检查主键所包含的列，如果存在列的类型不为INT，就报警告，有参数可配置。
- 检查一个索引所占的空间是不是超过了767个字节，如果超过则报错，类型为错误，不可配置。
- 检查在创建索引时，是不是与线上目前已经存在的索引名字存在冲突，如果是则报错，类型为错误，不可配置。
- 检查在创建索引或表时，索引个数是不是超过了参数设置值，如果是则报警告。
- 检查索引列中是否包含BLOB列，类型为错误，不可配置。
- 检查建表时，是不是存在同名的索引，类型为错误，不可配置。
- 检查重复索引，类型为错误，不可配置。

# 修改表语句检查项

- 检查改表语句中，要修改的表属性类型，目前只支持修改存储引擎、修改注释、修改AUTO INCREMENT、修改字符集，在此之外的，都不支持，类型为警告。
- 检查在一个语句块中，是不是存在对同一个表的改表语句，类型为警告，可配置。
- 检查改表类型，目前支持的有加列、加索引、删列、改表名、修改列、删索引、修改列默认值、修改表属性（第1点中说明了具体支持的类型），以及CONVERT表字符集这几种类型，其他改表类型都不支持。除此之外，都报错，类型为警告。

## 总结

- 针对线上MySQL服务器是不是5.6以上（包含）版本，有不同的处理策略。比如在预估影响行数时，5.6可以直接对任何DML语句做EXPLAIN操作，而在5.5及以下版本中，只支持对SELECT语句执行EXPLAIN操作，并且有些DML语句不容易直接转换为SELECT语句去做EXPLAIN，这样就导致预估行数为0。
- 还是针对5.6以上版本与5.5版本的不同，DATETIME、TIMESTAMP系列类型在执行时，5.5的限制比较多，而5.6基本通用，所以这上面的处理可能在5.6及5.5版本上，相同语句返回的结果集是不同的（规则在5.5版本中，以在执行时报的错误信息为准），这与线上版本有关系。
- 规则是人定的，不能尽善尽美，也不能覆盖全部，更不可能一个规则适合所有人使用，所以通过配置参数的方式，让不同的用户做不同的选择，决定哪些才是最适合自己的。不过即使这样，我认为还是不够好，因为这些能配置的参数毕竟有限，都是内置的，想新增规则不太容易。一种方法就是通过修改源码来实现，但门槛不低，很多人因此望而却步，所以我们提供了另一种方法，即arkit的语法树打印，这样可以结构化地分析语句中的每一部分、每一个表达式，有了这个利器，想要增加自己的规则就不难了，并且有非常大的发挥空间，应用也不止于此，可以尽情地发挥自己的想象，更好地使用arkit，让arkit最大限度地为大家服务。

# 参数变量

## 语法和变量

考虑到不同用户定义和使用的规范会有所不同，arkit支持很多可配置的参数。这些配置参数都是全局参数。对于同一个服务的规则，不应该经常变化，或者说不应该出现一些业务是这样，而另一些业务是那样的规则，所以这些变量一经设置，就影响所有的审核规则。如果一家公司或一个项目确实需要有多个审核规则，那么建议配置多套arkit服务，在各自的配置文件中指定相应的参数值，分别进行审核。

可以通过show variables like "%arkit%";查看所有arkit支持的参数，arkit目前所支持的变量参数包括下面这些。

- arkit_license_file

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来表示arkit的License文件的地址。

- arkit_expire_warning_days

| 指标 | 说明 |
| --- | --- |
| 默认值 | 30 |
| 类型 | INT |
| 范围 | 全局 |

用来设置在License到期前多少天在做提醒，提醒会出现在Error Log中，提示内容如下：`2017-10-10T03:42:09.998799Z 3 [Warning] license is about to expire (2017-12-31), please pay attention to upgrade.`，这个提醒每天只会出现一次。

- arkit_remote_backup_host

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来表示当前arkit在执行完成之后，备份时对应的数据要存储的机器地址。

- arkit_remote_backup_port

| 指标 | 说明 |
| --- | --- |
| | |

| | |
|---|---|
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来表示备份数据库的的端口，与arkit_remote_backup_host共同决定备份数据库实例信息。

- arkit_remote_backup_user

| 指标 | 说明 |
|---|---|
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来表示arkit在连接备份数据库时，所需要的用户名。

- arkit_remote_backup_password

| 指标 | 说明 |
|---|---|
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来表示arkit在连接备份数据库时，所需要的密码，与上面的参数arkit_remote_backup_user一起使用。

- arkit_metadata_host

| 指标 | 说明 |
|---|---|
| 默认值 | 无 |
| 类型 | String只读 |
| 范围 | 全局 |

用来表示当前arkit在执行完成之后，执行/审核结果详细信息存储的位置。用来存储这些详细

信息的表都会自动创建。建议端口指定arkit插件所在MySQL实例的端口，这样与information_schema插件在同一个实例，操作起来方便一些。

- arkit_metadata_port

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String只读 |
| 范围 | 全局 |

用来表示当前arkit在执行完成之后，执行/审核结果详细信息存储的位置。用来存储这些详细信息的表都会自动创建。建议端口指定arkit插件所在MySQL实例的端口，这样与information_schema插件在同一个实例，操作起来方便一些。

- arkit_metadata_user

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String只读 |
| 范围 | 全局 |

用来表示当前arkit在执行完成之后，在存储执行/审核结果详细时arkit连接数据库所需要的用户名。

- arkit_metadata_password

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String只读 |
| 范围 | 全局 |

用来表示当前arkit在执行完成之后，在存储执行/审核结果详细时arkit连接数据库所需要的密码。

- arkit_slave_ports_range

| | |
| --- | --- |

| 指标 | 说明 |
| --- | --- |
| 默认值 | 3306:3321 |
| 类型 | String |
| 范围 | 全局 |

在arkit在线改表功能中使用，用来检查正在改表的数据库的从库节点的，主要是检查改表过程中，有没有造成复制延迟等问题。

- arkit_sql_statement

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String |
| 范围 | 会话 |

这个参数是用来提交任务的，将要审核的语句集，按照之前说的格式组织好，拼成一个字符串，然后将这个参数赋的值设置为这个字符串，即可完成任务的提交操作，在提交时，会初步检查语句的合法性，包括注释部分、License合法等问题，如果不合法则会报出相应的错误信息。举例说明：set arkit_sql_statement='/* *--user=username;--password=xxxx;--host=127.0.0.1;--port=3306;* */use db1; select * * from table1; insert into table2 values(1,1);'，很明显，只要在注释后面跟上所有要提交的语句并用“;”分隔即可。

- arkit_status_cache_expire_time

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String |
| 范围 | 全局 |

用来设置任务执行完成之后多久，information_schema.arkit_progress表中对应的任务就过期了，也就是查不到了。因为执行过程是异步的，所以在查看进度时有可能不太及时，所以通过这个参数可以设置完成之后，在多少时间的范围内还是可以查到的。

- arkit_stop_order_sequence

| | |
| --- | --- |

| 指标 | 说明 |
| --- | --- |
| 默认值 | 无 |
| 类型 | String |
| 范围 | 会话 |

这个参数是用来停止一个任务的，当执行的任务引起了数据库的压力过大或者其它问题时，可以通过这个参数来停止这个任务，参数的值是任务的标识符——sequence。这个参数在设置之后，如果有改表（OSC）正在执行，那么就会将正在执行的改表操作取消掉，如果没有则会将当前任务中止掉，即在执行完当前语句之后，从下一条语句开始就不再执行了，这样这个任务就会结束了，可以从arkit.arkit_feedback表中查看具体详细信息。

如果有改表操作在执行，有可能遇到的问题是，执行了一部分，突然发现对线上造成了MDL等待的现象。这种影响对一些业务是不可接受的，因为很多语句此时就不能执行当前表上的任何操作了，必须要等OSC的一些辅助操作（建立/删除触发器）完成之后才可以，而OSC的这些操作又是在等待线上一些慢查询语句执行完成之后才能继续执行下去。这种情况下，一般的处理方式是，先退出OSC执行，压力小的时候多试几次，才可以继续执行下去，此时最需要DBA操作的就是取消当前这个OSC的执行，所以arkit支持OSC执行的中止功能。在取消之后，还有几点需要注意，如下。

i. 在多个ALTER语句一起执行的情况下，如果取消某一个，那么整个执行过程就都会中止，同时被取消的语句在返回的结果集中是未执行状态。
ii. 在取消语句的错误描述信息中，报错为“`Execute has been abort in percent: 已执行比例, remain time: 剩余时间`”。
iii. 在取消之后，当前语句之后的所有语句都不会执行，当然状态为未执行。
iv. 被取消语句，在取消之后，结果集stagestatus列的信息会设置为“`Execute Aborted`”。

- arkit_worker_queue_size

| 指标 | 说明 |
| --- | --- |
| 默认值 | 20 |
| 最大值 | 10000 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

这个参数用来设置在arkit内部，每一个任务队列的大小，因为在通过arkit_sql_statement设

置任务之后，在内部实际上是排队的，如果设置的队列长度太小，可能会引起提交的失败。

- arkit_parallel_workers

| 指标 | 说明 |
| --- | --- |
| 默认值 | 5 |
| 最大值 | 24 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

这个参数用来控制在arkit内部，有多少个线程来处理提交的任务，同一个实现的执行都会被分发到同一个线程中执行，防止多个任务在同一个实例中并行执行引起数据库的压力过大的问题。

- arkit_max_key_length

| 指标 | 说明 |
| --- | --- |
| 默认值 | 767 |
| 最大值 | 1 |
| 最小值 | 3072 |
| 类型 | Integer |
| 范围 | 全局 |

这个参数用来设置一个索引的最大长度。

- arkit_max_allowed_statements

| 指标 | 说明 |
| --- | --- |
| 默认值 | 100000 |
| 最大值 | 1 |
| 最小值 | 204800 |
| 类型 | Integer |

| | |
|---|---|
| 范围 | 全局 |

这个参数用来设置在一个任务里，最多允许提交多少个SQL语句，用来防止开发提交的语句过多导致内存不足以及系统变慢等其它问题。

- arkit_alter_table_method

| 指标 | 说明 |
|---|---|
| 默认值 | auto |
| 可选值 | "build_in_osc", "pt_osc", "direct_alter" |
| 类型 | enum |
| 范围 | 会话 |

这个参数，是用来控制在改表时，采用什么方式来改，build_in_osc是采用arkit实现的通过Binlog来获取增量数据的方式来改表的，pt_osc是通过pt-online-schema-change来改表，而direct_alter表示不采用在线改表方式改，直接执行ALTER语句，在确定不会锁表的情况下，可以选择这种方式。
如果设置为direct_alter，则表示指定了当前改表方式为直接修改，如果是大表修改，很明显会导致锁表，当然需要谨慎选择。当然，在很明显的可以直接改表的语句时，即使参数设置不是direct-alter，也会选择直接改表的方式去修改，比如删除索引、修改列的默认值、修改表注释、5.6版本及以上修改自增ID值、5.6版本及以上修改默认字符集操作，如果只是单纯的做这些修改，则都是不需要锁表，所以都是直接改表的，而如果包含不属于以上情况的，则都需要锁表并且通过上述参数来决定该通过什么方式来改表。

- arkit_check_insert_field

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在插入语句中，用来控制是否指定插入列列表，如果没有指定，并且参数值为ON，则会报错。

- arkit_check_dml_where

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在审核DML语句时，如果发现没有WHERE条件，并且此参数设置为ON，就会报错，否则被忽略。

- arkit_check_dml_limit

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能说明是在DML语句中，如果使用了LIMIT表达式，并且此参数设置为ON，就会报错。这一般用来防止STATEMENT语句主从复制时导致主从不一致的问题。

- arkit_check_dml_orderby

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在DML语句中，如果使用了OrderBy表达式，并且此参数设置为ON，就会报错。这一般用来防止STATEMENT语句主从复制时导致主从不一致的问题。

- arkit_enable_select_star

| 指标 | 说明 |
| --- | --- |

| | |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在遇到查询语句为“select * from”，并且此参数设置为ON时，不会报错，否则会报错。

- arkit_enable_empty_index_name

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是如果建表，或者增加索引时，此索引没有名字，是否报错。

- arkit_enable_orderby_rand

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在语句中出现order by rand()时，用来控制是否报错，设置为ON表示不报错，否则会报错。

- arkit_enable_nullable

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |

| | |
|---|---|
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在创建或者新增列时，如果列为NULL，用来控制是否报错，如果设置为ON，表示不报错，否则会报错。

- arkit_enable_foreign_key

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在创建表或增加索引时，如果存在外键，用来控制是否报错，如果设置为ON，则不报错，否则会报错。

- arkit_enable_pk_columns_only_int

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在创建表或增加索引时，检查主键列类型是不是INT，BIGINT，如果不是则会报错。

- arkit_max_primary_key_parts

| 指标 | 说明 |
|---|---|
| 默认值 | 5 |
| 最大值 | 64 |
| | |

| | |
|---|---|
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

参数功能是在一个主键中，用来控制主键列的最大个数，如果超过这个数目则报错。在增加索引或新建表时，都会生效。

- arkit_max_key_parts

| 指标 | 说明 |
|---|---|
| 默认值 | 5 |
| 最大值 | 64 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

参数功能是在一个索引中，用来控制列的最大个数，如果超过这个数目则报错。在增加索引或新建表时，都会生效。

- arkit_max_update_rows

| 指标 | 说明 |
|---|---|
| 默认值 | 10000 |
| 最大值 | UINT_MAX |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

参数可选范围为1~MAX，参数默认值为10000，功能是在一个修改语句中，用来控制预计影响的最大行数，如果超过这个数就报错。这个参数的获取方法是explain，对于有一些语句或在MySQL 5.5版本中获取不到相应语句时，预计行数都会是0，这时这个参数就失效了。

- arkit_max_keys

| | |
|---|---|

| 指标 | 说明 |
| --- | --- |
| 默认值 | 16 |
| 最大值 | 1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

参数功能在一个表中，用来控制支持的最大索引数目，如果超过这个数则报错，不管在新增表，还是新增索引时，都有效。

- arkit_enable_not_innodb

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在新建表指定的存储引擎不是Innodb时，用来控制是否报错，如果设置为ON，则不报错，否则会报错。

- arkit_unique_index_prefix

| 指标 | 说明 |
| --- | --- |
| 默认值 | uniq_ |
| 类型 | String |
| 范围 | 全局 |

参数表示的是唯一索引名字的前缀，可以自己定义，字符串中可以指定的字符仅包括数字，字母，下划线。

- arkit_normal_index_prefix

| 指标 | 说明 |
| --- | --- |

| | |
|---|---|
| 默认值 | idx_ |
| 类型 | String |
| 范围 | 全局 |

参数表示的是普通二级索引名字的前缀，可以自己定义，字符串中可以指定的字符仅包括数字，字母，下划线。

- arkit_support_charset

| 指标 | 说明 |
|---|---|
| 默认值 | utf8mb4 |
| 类型 | String |
| 范围 | 全局 |

参数可选范围为MySQL支持字符集，参数默认值为“utf8mb4”，功能是表示在建表或建库时支持的字符集，如果需要多个，则用逗号分隔，影响的范围是建表、设置会话字符集、修改表字符集属性等。

- arkit_check_table_comment

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在建表及没有设置表注释时，用来控制是否报错，如果设置为ON，则会报错。

- arkit_check_column_comment

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |

| | |
|---|---|
| 范围 | 全局 |

功能是在建表或改表加列，并且没有设置列注释时，用来控制是否报错，如果设置为ON，则会报错。

- arkit_check_primary_key

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在建表时，如果没有创建主键，用来控制是否报错，如果设置为ON，就会报错。

- arkit_enable_partition_table

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数可选范围为ON/OFF，参数默认值为OFF，功能是在建表时，如果创建了分区表，用来控制是否报错，如果设置为ON，不会报错，否则会报错。

- arkit_enable_enum_set_bit

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数可选范围为ON/OFF，参数默认值为OFF，功能是在建表或加列时，如果列对应的数据类型指定的是enum、set、bit数据类型，用来控制是否报错，如果设置为ON，则不报错，否则会报错。

- arkit_check_index_prefix

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是用来检查新建或建表时的索引前缀，普通索引的前缀为“idx_”，唯一索引的前缀为“uniq_”，如果设置为ON，并且索引前缀不符合规则，则会报错。

- arkit_enable_autoincrement_unsigned

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在新建表时，如果自增列不是无符号整型的数据类型，用来控制是否报错，如果设置为ON，就报错，否则不报错。

- arkit_max_char_length

| 指标 | 说明 |
| --- | --- |
| 默认值 | 16 |
| 最大值 | UINT_MAX |
| 最小值 | 1 |
| 类型 | Integer |
| | |

| | |
|---|---|
| 范围 | 全局 |

参数功能是用来控制当char类型的长度大于多少时，就提示将其转换为VARCHAR。

- arkit_check_autoincrement_init_value

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是当建表时自增列的值指定不为1时，用来控制是否报错，如果设置为ON，则报错。

- arkit_check_autoincrement_datatype

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在建表时自增列的类型不为int或bigint时，用来控制是否报错，如果设置为ON，则会报错。

- arkit_check_timestamp_default

| 指标 | 说明 |
|---|---|
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在建表时，如果没有为timestamp类型指定默认值，用来控制是否报错，如果设置为

ON，则会报错。

- arkit_enable_column_charset

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数可选范围为ON/OFF，参数默认值为OFF，功能是在新建表或修改表加列改列时，用来控制是否能单独指定列的字符集，如果设置为ON，则表示可以设置，不报错。

- arkit_check_autoincrement_name

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是在建表时，如果指定的自增列名字不为ID，用来控制是否报错，如果设置为ON，则报错，表示这个列可能存在业务意义，起到提示的作用。

- arkit_merge_alter_table

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在同一个arkit任务中，多个语句修改同一个表的语句出现时，用来控制是否报错，如果设置为ON，则报错，并提示合成一个。

- arkit_check_column_default_value

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在建表、修改列、新增列时，用来控制新的列属性是否要有默认值，如果设置为ON，则说明必须要有默认值，否则会报错。

- arkit_enable_blob_type

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在建表、修改列、新增列操作时，如果存在BLOB类型的列，用来控制是否报错，如果设置为ON，说明支持BLOB类型，则不会报错。

- arkit_enable_identifier_keyword

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数功能是在所有审核的SQL语句中，如果有标识符被写成MySQL的关键字，用来控制是否报错。如果设置为ON，说明支持标识符为关键字，就不会报错，否则会报错。

- arkit_enable_subselect

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

用来设置是否支持子查询的，如果在被审核语句中出现子查询，则会有警告出现。

- arkit_enable_sql_statistic

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

用来设置是否支持在统计arkit执行过的语句中，记录各种语句分别占多大比例。如果参数值为ON，则每次执行的情况都会在备份数据库实例中arkit库的statistic表中，以一条记录的形式存储这次操作的统计情况，每次操作对应一条记录，这条记录中含有的信息是各种类型的语句执行次数情况。

- arkit_read_only

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

参数可选范围为ON/OFF，参数默认值为OFF。设置当前arkitarkit服务器是否为只读，这是为了防止一些人在具有修改权限的账号时，通过arkit误修改一些数据。如果arkit_read_only设置为ON，则即使打开了enable-execute，同时又有执行权限，也不会去执行，审核完成即返回。

- arkit_check_identifier

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 可选值 | OFF，ON |
| 类型 | Bool |
| 范围 | 全局 |

功能是打开与关闭arkit对SQL语句中各种名字的检查。如果设置为ON，则发现名字中存在除数字、字母、下划线之外的字符时，会报Identifier "invalidname" is invalid, valid options: [a-z,A-Z,0-9,_]。

- arkit_max_primary_key_parts

| 指标 | 说明 |
| --- | --- |
| 默认值 | 5 |
| 最大值 | 64 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 全局 |

参数可选范围为1~64，参数默认值为5，功能是在创建表时，如果主键所包含的列个数超过这个设置的值，则会报警告。

- arkit_osc_check_interval

| 指标 | 说明 |
| --- | --- |
| 默认值 | 5 |
| 最大值 | 1024 |
| 最小值 | 0 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--check-interval，意义是Sleep time between checks for --max-lag。

- arkit_osc_chunk_size

| 指标 | 说明 |
| --- | --- |
| 默认值 | 4 |
| 最大值 | 1024 |
| 最小值 | 0 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--chunk-size。

- arkit_osc_chunk_size_limit

| 指标 | 说明 |
| --- | --- |
| 默认值 | 4 |
| 最大值 | 1024 |
| 最小值 | 0 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--chunk-size-limit。

- arkit_osc_chunk_time

| 指标 | 说明 |
| --- | --- |
| 默认值 | 1 |
| 最大值 | 1024 |
| 最小值 | 0 |
| 类型 | Integer |
| | |

| | |
|---|---|
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--chunk-time。

- arkit_osc_critical_connected

| 指标 | 说明 |
|---|---|
| 默认值 | 1000 |
| 最大值 | 1024*1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--critical-load中的thread_connected部分。

- arkit_osc_critical_running

| 指标 | 说明 |
|---|---|
| 默认值 | 80 |
| 最大值 | 1024*1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--critical-load中的thread_running部分。

- arkit_osc_max_connected

| 指标 | 说明 |
|---|---|
| 默认值 | 1000 |
| 最大值 | 1024*1024 |
| 最小值 | 1 |
| 类型 | Integer |

| | |
|---|---|
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--max-load中的thread_connected部分。

- arkit_osc_max_running

| 指标 | 说明 |
|---|---|
| 默认值 | 80 |
| 最大值 | 1024*1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--max-load中的thread_running部分。

- arkit_biosc_drop_new_table

| 指标 | 说明 |
|---|---|
| 默认值 | ON |
| 类型 | Bool |
| 范围 | 会话 |

与通过pt-online-schema-change来改表一个道理，这个参数是用来控制通过build_in_osc来改表时，结束时要不要删除new_table的行为的。

- arkit_biosc_drop_old_table

| 指标 | 说明 |
|---|---|
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 会话 |

与通过pt-online-schema-change来改表一个道理，这个参数是用来控制通过build_in_osc来改表时，结束时要不要删除old_table的行为的。

- arkit_osc_drop_new_table

| 指标 | 说明 |
| --- | --- |
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--[no]drop-new-table。

- arkit_osc_drop_old_table

| 指标 | 说明 |
| --- | --- |
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--[no]drop-old-table。

- arkit_osc_check_replication_filters

| 指标 | 说明 |
| --- | --- |
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--[no]check-replication-filters。

- arkit_osc_check_alter

| 指标 | 说明 |
| --- | --- |
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--[no]check-alter。

- arkit_osc_max_lag

| 指标 | 说明 |
| --- | --- |
| 默认值 | 3 |
| 最大值 | 1024*1024 |
| 最小值 | 0 |
| 类型 | Integer |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数--max-lag。

- arkit_osc_recursion_method

| 指标 | 说明 |
| --- | --- |
| 默认值 | processlist |
| 可选值范围 | "processlist", "hosts", "none" |
| 类型 | enum |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数recursion_method，具体意义可以参考OSC官方手册。

- arkit_osc_alter_foreign_keys_method

| 指标 | 说明 |
| --- | --- |
| 默认值 | none |
| 可选值范围 | "auto", "none", "rebuild_constraints", "drop_swap" |
| 类型 | enum |
| 范围 | 会话 |

对应参数pt-online-schema-change中的参数alter-foreign-keys-method，具体意义可以参考

OSC官方手册。

- arkit_osc_min_table_size

| 指标 | 说明 |
| --- | --- |
| 默认值 | 16 |
| 最大值 | 1024*1024 |
| 最小值 | 0 |
| 类型 | Integer |
| 范围 | 会话 |

这个参数实际上是一个OSC开关，如果设置为0，则全部ALTER语句都使用OSC方式，如果设置为非0，则当这个表占用空间大小大于这个值时才使用OSC方式。单位为M，这个表大小的计算方式是通过语句 **select (DATA_LENGTH + INDEX_LENGTH)/1024/1024 from information_schema.tables where table_schema = 'dbname' and table_name = 'tablename'** 来实现的。

- arkit_osc_on

| 指标 | 说明 |
| --- | --- |
| 默认值 | yes |
| 类型 | Bool |
| 范围 | 全局 |

一个全局的OSC开关，默认是打开的，如果想要关闭则设置为OFF，这样就会直接修改。

- arkit_osc_print_sql

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 类型 | Bool |
| 范围 | 全局 |

对应参数pt-online-schema-change中的参数--print。

- arkit_osc_print_none

| 指标 | 说明 |
| --- | --- |
| 默认值 | OFF |
| 类型 | Bool |
| 范围 | 全局 |

用来设置在arkit返回结果集中，对于原来OSC在执行过程的标准输出信息是不是要打印到结果集对应的错误信息列中，如果设置为1，就不打印，如果设置为0，就打印。而如果出现了错误，则都会打印。

- arkit_biosc_check_delay_period

| 指标 | 说明 |
| --- | --- |
| 默认值 | 10000 |
| 最大值 | 1024*1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

会话级参数，用来控制，在把数据都COPY完之后，APPLY Binlog日志时，每处理多少个事件，检查一次复制延迟的值，如果发现复制延迟已经比较小了，则会开始准备锁表，而如果延迟还是比较大，则继续消费Binlog，那么这个参数所表示的就是每隔多少个事件来检查一次，所以应该是个比较大的数字，默认值为10000。

- arkit_biosc_min_delay_time

| 指标 | 说明 |
| --- | --- |
| 默认值 | 10 |
| 最大值 | 65535 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

会话级参数，用来控制，在Binlog增量消费时，会每隔arkit_biosc_check_delay_period秒的时间检查一次延迟时间，如果延迟时间小于参数arkit_biosc_min_delay_time的值，则会优先给表加锁，加完锁之后，继续处理Binlog增量，但此时Binlog就不会再增加了，处理完了，就会开始RENAME表了，所以参数arkit_biosc_min_delay_time所控制的就是临界值的处理窗口大小，设置的大，则会业务影响大一些，设置的小，对业务就影响小一些。默认值为10秒。

- arkit_biosc_lock_table_max_time

| 指标 | 说明 |
| --- | --- |
| 默认值 | 16 |
| 最大值 | 1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

会话级参数，用来控制在每次将表上锁之后，如果一直还没有把Binlog消费完，则达到这个时间之后，为了不影响业务，先将表锁释放了，那么这个值表示的就是最大的锁表时间值。单位为秒。

- arkit_biosc_lock_wait_timeout

| 指标 | 说明 |
| --- | --- |
| 默认值 | 16 |
| 最大值 | 1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

会话级参数，用来控制RENAME线程在RENAME操作时，如果上锁时间超过这个参数的值，则这个RENAME操作就超时了，这样就避免了对业务的更大的影响。单位为秒。

- arkit_biosc_retry_wait_time

| 指标 | 说明 |
| --- | --- |

| | |
|---|---|
| 默认值 | 100 |
| 最大值 | 1024 |
| 最小值 | 1 |
| 类型 | Integer |
| 范围 | 会话 |

会话级参数，上面已经知道，如果锁表时间超过这个值，则会释放锁让业务恢复正常，而恢复正常需要一个时间窗口，不可能是释放之后，马上就再加锁，那么这个参数所控制的就是在再次加锁前，等待多少时间，等待的时间越多，有可能产生的Binlog越多，改表时间越长，这个要根据实际的线上压力在改表前来动态调整。

# 环境变量的设置

在提交arkit任务时，整个语句块被arkit认为是一条单独的SQL语句，在内部会逐条地拆开。那么，在分析的过程中，必然会有一些参数一直影响着所有语句的执行，这些参数被arkit称为环境变量。目前支持的环境变量有两种，分别是use db和set names charsetname两种语句。

环境变量，在arkit的执行过程中，会全程陪伴。如果执行错误了，那么在重新生成任务时，不仅是要把未执行的语句提取出来，还需要把这些环境变量的变迁过程按照顺序获取到，目的是要得到未执行语句前面的最后一个环境变量的语句，这也正是为什么arkit返回结果集中stage列的值存在RERUN情况的原因了。

如果它的值是RERUN，这就表示当前语句是用来设置环境变量的，在开发自动化运维平台时，这是需要格外注意的。如果只是提取了未执行的语句，那是没有意义的，因为首先会丢失字符集设置（有时候会引起乱码），并且导致一些表找不到其对应的库，从而导致再次执行失败的问题。下面就分别来介绍一下两种设置环境变量语句的意义。

## use db

在执行过程中，必然会找到每一个使用到的库、表等对象在线上环境中的对应关系，也就是里面包含了所有SQL语句对应的上下文环境，从前面的例子中也可以看出，里面通过use语句来指定了库名。

关于`use database`，如果没有使用use语句，则必须要在使用表的同时指定库名，即`databasename.tablename`，否则arkit会直接报错。如果通过 use 语句指定过一次数据库，那么在当前语句块中，后面可以不再指定，除非想要修改当前库的名称。而如果只是通

过`databasename.tablename`方式指定了表的位置，则后面的操作不会受到前面的影响，也就是后面还需要明确指定库名。

对于上面两种不同的指定库名的方式，use database是可以影响上下文环境的，而databasename.tablename的方式，只会影响语句自己。当然在use db1之后，如果想要执行类似`insert db2.table1`的语句，也是可以的，此时这条语句本身就会临时切换为db2库，而之后的语句如果没有明确指定库名的话，还会恢复到db1的环境中。

## set names charsetname

对于我们熟悉的`set names utf8;`，可以在所有语句开头指定，如果后面不再出现并且不做修改，则所有语句都会以utf8的字符集来执行，而如果在中间某一个位置重新指定了，比如`set names utf8mb4;`，那么从这个设置开始，后面所有的语句执行都会切换为utf8mb4的字符集。

# arkit的备份回滚

## 备份存储架构

前面已经提到，arkit在做DML操作时，具有备份功能。它会将所有当前语句修改的行对应生成回滚语句并备份下来，同时也会将所有操作的任务备份下来，一起存储到一个指定的库中，这些库的指定需要通过4个参数，分别如下。

- arkit_remote_backup_host：指定远程备份MySQL实例的地址。
- arkit_remote_backup_port：指定远程备份MySQL实例的端口。
- arkit_remote_system_user：备份时，连接上面指定的MySQL实例时所需要的用户名，这个用户需要有相应的权限，一般包括CREATE、INSERT及SELECT权限。
- arkit_remote_system_password：备份时，连接备份库时所需要的用户对应的密码。

如果没有设置上面四个参数而又想要使用备份功能，那么在提交任务时，arkit就会报错，错误信息是Invalid remote backup information。

arkit 默认是开启备份功能的。为了使备份成为一个会话级的可选功能，有一个辅助选项可以设置，即前面讲述关于Incpetion支持选项的内容中所说到的--disable-remote-backup。

备份信息及回滚语句，在备份机器上的存储与线上被修改的数据库是一一对应的。但因为线上机器会很多，而备份机器只有一台，所以为了防止在备份数据库实例中存在库名冲突的问题，备份机器的库名是将线上机器IP地址的点换成下划线，再加上端口号及库名，由这三部分组成。这三部分也是通过下划线连接起来的。例如`192_168_1_1_3310_arkitdb`。下面用arkitdb作为线上库名

来举例。

对于一个备份库，它所包含的表与相应的线上表是一一对应的，也就是说线上库arkitdb中有什么表，在备份库`192_168_1_1_3310_arkitdb`中就有什么表，表名也完全相同，不同的只是表结构，它用来存储所有对这个表修改的回滚语句，表结构如下。

```
CREATE TABLE `tablename` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `rollback_statement` mediumtext,
  `opid_time` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

每一列的解释如下。

- rollback_statement：该列存储的是当某一行被修改后，生成的对应的回滚语句。因为Binlog 是 ROW 模式的，所以不管是什么语句，产生的回滚语句都是针对一行的。同时，如果一条语句的执行影响了多行，那么这里就会有多条回滚语句，但它们对应的是同一条SQL语句。
- opid_time：这一列存储的是被执行的 SQL 语句在执行时获取的一个序列号，这个序列号由3部分组成：timestamp（int值，是语句被执行的时间点）、线上服务器执行时所产生的thread_id及当前这条语句在所有被执行语句块中的一个序号。产生结果类似下面的样子：**1413347135_136_3**，这个序列号在指定的一个备份库中是唯一的。针对同一条语句影响多行的情况，在产生的多行回滚语句中，该列的值都是相同的，这样就可以找到一条语句对应的所有被影响数据的回滚语句。因为这个值是唯一的，所以可以通过这个值在备份库中找到某一条SQL语句对应的所有回滚语句。

除了在每一个库下面生成的回滚语句表之外，还有一个总的arkit公共库，这个库用来记录每一个语句的执行信息，表名为arkit_execution_backup。该表的结构如下。

```
CREATE TABLE `arkit_execution_backup` (
  `opid_time` varchar(50) DEFAULT NULL,
  `start_binlog_file` varchar(512) DEFAULT NULL,
  `start_binlog_pos` int(11) DEFAULT NULL,
  `end_binlog_file` varchar(512) DEFAULT NULL,
  `end_binlog_pos` int(11) DEFAULT NULL,
  `sql_statement` text,
  `host` varchar(64) DEFAULT NULL,
  `port` int(11) DEFAULT NULL,
  rollback_dbnames VARCHAR(4096) comment,
  tables_full_path VARCHAR(4096) comment,
  `time` timestamp NULL DEFAULT NULL,
  `type` varchar(20) DEFAULT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

下面详细解释一下每一列的作用及意义，表中的每一行，对应在线上执行的一条实际的SQL语句。

- opid_time：该列与上面备份表中的列 `opid_time` 是一一对应的，因为这个表中每一行对应的是在线上执行的一条实际的SQL语句，所以opid_time的意义就是用来唯一表示这条语句的。opid_time从各个备份表中查找这条语句对应的回滚语句，是一对多的关系。
- start_binlog_file：该列从名字中可以看出，表示的是执行这条语句前 Binlog 所在位置的文件名。当然这个值不一定准确，因为这是在执行语句前通过 `show master status;` 语句来获取的，在数据库并发比较高的情况下，这个值一般都不是当前语句的Binlog开始的准确位置，这个位置只能是这条语句产生Binlog前面的某个位置。同理，下面三个位置信息也是一样。
- start_binlog_pos：该列与上面的列对应，表示的是上面指定文件的位置信息。
- end_binlog_file：该列表示的是执行当前语句之后，Binlog所在的文件名。
- end_binlog_pos：该列与上面的列对应，它表示执行完成之后，Binlog在文件end_binlog_file中的偏移位置。
- sql_statement：该列存储的是当前这个被执行的SQL语句。
- host：表示在线上的哪个数据库实例中执行了这条语句。存储的是当前提交语句时在注释中的指定地址。
- port：与 host 对应，表示执行时数据库的端口是什么。
- rollback_dbnames：表示执行当前语句时所处的环境变量，指的是数据库名。
- tables_full_path：表示当前语句影响的表的表名，可以通过这个名字对应到备份表名。
- time：表示当前语句的执行时间。
- type：表示操作类型，现在只支持INSERT、UPDATE、DELETE、CREATEDB、CREATETABLE、ALTERTABLE、DROPTABLE等类型。

依照现在备份及回滚的实现方案，如果已经知道一条语句的执行序列，想拿到这条语句的回滚语句，那么要执行的 SQL 语句如下。

上面语句查出来的只是针对一个语句块中某一条语句的回滚语句，如果想要得到整个语句块的回滚语句，还需要在此基础上做二次开发。首先根据结果集中返回的每一条语句对应的opid_time值，找到所有的回滚语句，然后分不同语句按照执行顺序的倒序排列，再去线上执行，这样才可以保证回滚正确。不过为了安全起见，可以把这些回滚语句放到一个事务中进行处理，如果事务太大，可以按照opid_time来分组，并针对每一个opid_time使用一个事务来执行，保证同一条语句的回滚是原子性的。

关于 DDL 的回滚，其实很难做得完美，因为涉及数据的大批量更改，并且 Binlog 是语句模式的，所以很难处理。但是，在 arkit 中采取的策略是只处理定义，不处理数据。现在 DDL 操作的回滚只包括 CREATE TABLE 、ALTER TABLE 、DROP TABLE，其他类型的操作不支持，并且也

没有太大意义，在实际使用中如果有需求再考虑如何实现。

每条 DDL 语句算一个操作，同样地，这个操作也会存储在表**arkit_execution_backup**中，关于 Binlog 的一些列没有实际意义，对应的 opid_time 与 DML 是相同的。而同时在被操作表中存储了回滚语句，其中的列opid_time与上面表中的这个列关联。一个 DDL 操作（不管其中做了多少事），对应的回滚语句也是一条语句。

# 备份所需条件

arkit支持了DDL及DML执行的备份回滚，但要想正确地使用并产生相应的预期结果，还需要注意一些细节。下面是它要满足的一些条件。

- 被修改表需要有主键：执行时，被影响的表如果没有主键的话，就不会做备份了。这样更简单并且备份时间及数据都会少一点，不然回滚语句的WHERE条件就会将所有列写进去，这样会影响性能且没有太大意义，所以在WHERE条件中，只需要主键即可。
- 参数 server_id 必须要设置为非0及非1，通过语句set global server_id=server_id;来设置，不然在备份时会报错。因为在获取Binlog时，需要通过server_id在主库上注册arkit的信息。
- 线上服务器必须要打开 Binlog，在启动时需要设置参数log_bin、log_bin_index等关于Binlog的参数。不然不会备份及生成回滚语句，因为arkit的生成回滚语句是通过解析Binlog来做的。
- 参数binlog_format必须要设置为mixed或者row模式，通过语句set global binlog_format=mixed/row 来设置。如果是 statement 模式，则不做备份及回滚语句的生成。这考虑到设置为statement时可能存在特别的原因，所以arkit执行语句时，也不会将对应的会话级参数binlog_format改为ROW模式了，以免造成不可预知的影响。
- 备份相关的四个参数需要设置好，并且对应的用户在备份数据库实例中有相应的权限。

# 打印语法树

在实际工作中，经常会有对SQL语句进行格式化、结构化的需求。但通常情况下，对SQL语句做分析，是一个门槛较高的工作，所以业界现在很少有工具可以满足我们这样的需求。

arkit可以完成这样的工作，在分析语句之后，将分析之后的语法树转换为JSON格式的字符串，通过字典的形式定义每一个表达式，语句等，通过这样的转换，arkit很好地将分析SQL语句这样具有高门槛的事情，巧妙地转换为一个结构化的、程序可轻易解析的工作，这样就给我们自动化程序提供了很多机会。除了arkit内置的、已经定义好的规则之外，DBA也可以捕捉并分析结构化的信息，进一步做更具个性化的审核。

# 标签定义

这里先简单讲一下语法树JSON串中的一些标签，如下。

- command: 每条语句都是以command开头的，它表示是什么语句类型，现在支持的有insert、update、delete、select这四种类型。
- table_object: 表示当前语句对哪个表做的操作，它只针对插入、删除操作，比如是插入哪个表，删除哪个表。而更新操作在语法树中不太好确认哪些表被改了，所以这里没有明确拿出来，而是可以通过从更新列的信息中取到表信息，这就是被更新的表。这是一个字典，里面包括的是一个或多个表信息，并且已经对应到其对应的数据库。
- db：表示当前对象所处的数据库名。
- table：表示当前处理的表名。
- fields: 表示插入语句中指定的要插入的字段列表，如果没有指定，则没有这个信息。它是一个数组，包括了语句中所指定的所有列信息，每一列都有完整的信息，包括数据库、表及列名。因为这是一个表达式，所以其表达式类型为FIELD_ITEM，后面会专门列出所有支持的表达式信息。
- select_insert_values: 这表示的是查询插入的查询部分，它是一个字典，里面包括了这个查询语句的所有元素，包括查询列、查询涉及的表、WHERE及ORDER BY等信息。
- select_list: 表示当前查询语句（或子查询）要查询的表达式信息，这里可以是列，也可以是其他计算出来的值，例子中就有select sno+1…这样的查询。
- table_ref: 表示当前语句上下文中使用到的表信息，是一个数组，包括了所有的表，这里所谓的上下文，可以简单理解为，在一个子查询的可见范围内的所有表达式，都是属于同一个层次的。如果一个列在当前上下文中找不到，那么可能就需要到上一层（父亲层）的上下文中找，如果找到了，就算作相关子查询，对于这种一条语句中有不同层级的情况，就存在不同的上下文。在例子中也有相关反映。
- where: 表示的是查询表达式（包括查询、删除、更新及查询等）的语法树，因为WHERE语句其实就是一个表达式，只是有可能是多个表达式的逻辑运算而已。
- orderBy: 表示查询时使用到的排序列，是一个数组。
- limit: 表示在查询时使用到的LIMIT信息，因为LIMIT是一个复合信息，包括了限制行数及开始位置等，所以会有limit及limit_offset，而limit_offset有可能没有，只有限制行数。
- groupBy: 表示查询时使用到的分组列，是一个数组。

- having: 表示查询时，使用到的having表达式。
- subselect: 如果使用到子查询，则它就用来表示这个子查询。它是一个字典，包括了一个完整的查询语句中所具备的所有属性，可以是递归的。
- many_values: 在查询语句中，如果插入的是值，而不是查询结果，则用这个来表示它的值列表，因为在MySQL中可以同时插入多个值，则这里有可能是多个。
- values: 如果插入的是值，则它用来表示一行的插入内容，这是一个数组，每个元素是一个列的表达式，也是many_values数组的一个元素。而如果是一条更新语句，那么它表示的就是被更新的值表达式列表。
- set_fields: 用于表示更新语句的更新列的信息，这是一个数组，里面的每个元素对应被更新的一个列表达式。
- select_list：表示查询语句中，被查询的表达式数组。
- type：如果当前是一个表达式的类型时，其值可能会包括FIELD_ITEM、STRING_ITEM、INT_ITEM、SUBSELECT_ITEM等，如果当前被处理对象是一个表时，其类型包括derived，physical，分别表示对应的表是子查询，还是实体的物理表类型，可以通过这个值来判断表类型。
- engine：表达式如果是子查询时，这个用来指示当前子查询的类型，包括single_select、union等。
- join_on：表示哪些表做了连接，其值是一个数组。

上面就是目前支持的语句中出现的标签说明，但是还有很大一部分是表达式的处理，在打印表达式时，每一个对象都有一个公共的Key，名为type，而针对不同的type，其他的Key就不一定相同了，而具体的不同就不多叙述了，这里只给出支持哪些表达式，除type之外的其他信息，在使用过程中一试便知。

# 支持表达式类型

下面是目前支持的所有表达式的列表，仅列出了type的不同的值。

- STRING_ITEM：字符串，有其他Key用来存储其具体值信息。
- FIELD_ITEM：列信息，有其他Key用来存储具体对应的库、表及列名等。
- FUNC_ITEM/COND_ITEM：逻辑运算信息，包括比较运算符、AND、OR、ISNULL、ISNOTNULL、LIKE、BETWEEN、IN、NOT、NOW及其他自定义或内置函数等运算操作。
- INT_ITEM: 整数值表达式。
- REAL_ITEM：符点数值表达式。

- NULL_ITEM：NULL值表达式。
- SUBSELECT_ITEM：子查询表达式。
- SUM_FUNC_ITEM：集函数表达式。
- ROW_ITEM：行表达式，比如select * from t where (sno,name) = (select sno,name from t1)，这里where条件中等值表达式的左值就是这个表达式类型。
- DECIMAL_ITEM：DECIMAL表达式类型。

以上就是目前所支持的表达式类型，已经基本覆盖了所有常用的表达式。

最后要说明的是，这里打印出来的信息，已经不完全只是语法分析结束之后的信息了，而是经过arkit加工过的。比如在查询语句中用到了子查询，存在不同的上下文，同时还使用了别名，或者在使用列时没有指定其表名等这几种情况，arkit都打印了每一列对应的库名表名，这样打印出来的信息中，就已经不存在没有定位（找到其库名表名）的列名了，使用中更加友好准确。比如上面例子中就有这样的情况（名为alisa_t的t表的别名）。也就是说，arkit在内部已经将上下文的依赖分析完了，有了自动化运维程序之后，只管用就可以了。

# 线上配置需求

对线上配置的需求如下。

- 线上服务器必须要打开 Binlog，在启动时需要设置参数log_bin、log_bin_index等关于Binlog的参数。不然不会备份及生成回滚语句，因为arkit的回滚语句生成是通过解析Binlog来做的。
- 参数binlog_format必须要设置为mixed 或 row 模式，通过语句set global binlog_format=mixed/row 来设置。如果是 statement 模式，则不做备份及回滚语句的生成。考虑到设置为statement时可能存在特别的原因，所以在arkit执行语句时，也不会将对应的会话级参数binlog_format改为ROW模式了，以免造成不可预知的影响。
- 参数 server_id 必须要设置为非0非1，通过语句set global server_id=server_id;来设置，不然在备份时会报错。
- 线上服务器一定要有指定用户名的权限，这是在语句前面的注释中指定的，做什么操作就要有什么权限，否则还是会报错，如果需要执行的功能，则要有线上执行语句的权限，比如DDL及DML。如果要执行arkit show 等远程命令，则有些语句是需要特殊权限的，且这些权限是需要授予的。关于权限这个问题，因为一般arkit是运行在一台固定机器上面的，那么在选项中指定的用户名和密码实际上是arkit机器对线上数据库访问的权限，所以建议在使用过程中，使用专门固定的账号来让arkit使用，最好是一个只读一个可写，这样在执行时用可写，审核/查看线上状态或者表库结果时用只读，会更安全。建议权限要包括SELECT、INSERT、UPDATE、DELETE、CREATE、DROP、PROCESS、ALTER、SUPER、REPLICATION SLAVE、REPLICATION CLIENT、TRIGGER。

- 对涉及的表的要求是需要有主键。如果没有主键，则arkit会选择不生成回滚语句。原因很明显，就是对应的回滚语句的WHERE条件没法指定，指定一个全表的列也没有意义，所以选择不生成。
- Binog参数binlog_row_image需要设置为full，因为在备份生成回滚语句时都需要解析Binlog，需要拿到记录的完整信息，不然备份不成功。