ArkProxy User Manual

1 概述

通过**MySQL中间件介绍ArkProxy**白皮书中，大家能够知道ArkProxy的整体架构以及相应的功能，用户手册主要就是介绍相应功能对应参数的介绍。作为一个DBA这部分是要最关心的，日常运维全靠这部分。

1.1 简介

ArkProxy不仅仅是简单的流量转发功能，在转发过程中，对所有请求做分析，通过智能分发算法、权重配置等特性，将后端数据库的架构隐藏起来，实现了对业务的透明化，业务不用再关注后端DB架构。

1.2 ArkProxy主要特性

- 读写分离
- 负载均衡
- 权重分发
- Hint分发
- 智能统计
- 动态加载配置
- 连接池
- 失败重试
- 消息压缩
- 丰富地配置参数
- 友好的操作指令

1.3 版本

时间	版本	备注
2018年2月23日	v 18.02.23	当前版本

2 使用教程

2.1 安装与部署

- 安装

  下载安装，这个有极数云舟提供，并会附一份license

- 创建、修改基本的配置文件：arkproxy.cnf。

```
##代表ArkProxy的配置，具体每个参数的意义，请参考后面说明
[arkproxy]
proxy_backend_passwd                    = proxy
proxy_backend_user                      = proxy
proxy_port                              = 3336
proxy_check_slave_lag_period            = 10
proxy_config_host                       =
proxy_config_passwd                     =
proxy_config_port                       = 0
proxy_config_user                       =
proxy_digest_trace                      = OFF
proxy_flush_thread_num                  = 5
proxy_format_fullpath                   = OFF
proxy_format_num_per_time               = 10
proxy_format_parameterize               = ON
proxy_format_thread_num                 = 5
proxy_general_log                       = ON
proxy_general_log_file                  = /data/proxy/proxy.log
proxy_license_key_path                  = /data/proxy/arkproxy.key
proxy_log_error                         = /data/proxy/error.log
proxy_multi_write_mode                  = OFF
proxy_namespace                         =
proxy_non_encrypted_ips                 = 127.0.0.1
proxy_recv_queue_size                   = 1000
proxy_send_queue_size                   = 10
proxy_server_heartbeat_period           = 100
proxy_shell_listen_port                 = 3335
proxy_shell_password                    = password
proxy_shell_username                    = proxyshell
proxy_sql_trace                         = OFF
proxy_sql_trace_thread_num              = 5
proxy_trace_busy_strategy               = 0
proxy_trace_busy_wait_time              = 500
proxy_trace_client_queue_size           = 1000
proxy_trace_flush_thread_sleep_msec     = 500
proxy_trace_format_thread_sleep_msec    = 500
proxy_trace_hash_mem_length             = 134217728
proxy_trace_hash_size                   = 200
proxy_trace_max_delay_time              = 3
```

```
proxy_trace_queue_array_length           = 500
proxy_trace_sql_bucket_length            = 100

#配置数据库服务
[server1]
#proxy类型，表示当前节点类型为Server
proxy_type=server
#数据库地址
backend_host=127.0.0.1
#数据库端口
backend_port=3306
#路由权重
weight=50
#当前节点，如果延迟达到这个值，就会停止前其路由请求
max_slave_lag= 200
#这个节点默认状态，中间层启动之后就是这个状态。
server_status = ONLINE
#对这个节点的注释
config_comment = day day up

#同上
[server2]
proxy_type=server
backend_host=127.0.0.1
backend_port=3315
weight=50

#路由设置
[router1]
#proxy类型，当前配置节点为路由节点
proxy_type=router
#路由类型：readwrite，读写节点
router_type=readwrite
#配置路由db
router_servers=server1
#路由注释
config_comment = xxxxxxxxx

[router2]
proxy_type=router
#只读路由
router_type=readonly
#有多个路由目标时，以逗号分隔，写在一起。
router_servers=server1,server2
```

注意：同一个类型的路由，不能设置多次，比如readonly，只能有一个配置节点，如果对应多个Server，则可以将多个Server的名字写到router_servers中，放在一起。

2.2 配置权限

- 必须配置项：

  proxy_backend_user && **proxy_backend_passwd**：
  只需要配置这个权限就好

  ```
  ALL PRIVILEGES
  ```

  proxy_shell_username && **proxy_shell_password**：
  只需要在配置文件中填写用户和密码就好，就可以直接访问管理端口

- 可配置项：
  proxy_config_user：

- 启动

只需要启动：

```
##暂时先这么写
arkproxy --defaults-file=/data/proxy/arkproxy.cnf &
```

3 参数说明

- proxy_version

Name	**proxy_version**
Scope	Global
Dynamic	NO
Type	只读string

这个参数用来显示当前中间层的版本号，版本信息以时间为序列，比如“18.02.23”，则表示是2018年2月23号的版本。参数类型为只读，所以只能查看，不能通过任何方式来设置这个参数。

- proxy_backend_user

Name	**proxy_backend_user**

Scope	Global
Dynamic	NO
Type	string

这个参数是ArkProxy用来连接MySQL数据库集群的用户名，这个用户名的作用非常简单，只负责获取业务连接的权限信息，不会实际执行具体的业务请求语句，参数要求对后端所有数据库都具有相同访问权限。

- proxy_backend_passwd

Name	**proxy_backend_passwd**
Scope	Global
Dynamic	NO
Type	string

这个参数是ArkProxy用来连接MySQL数据库集群的用户的密码，与参数proxy_backend_user对应，目前不支持加密，只能明文存储。

- proxy_check_slave_lag_period

Name	**proxy_check_slave_lag_period**
Scope	Global
Dynamic	NO
Type	string
Default	10
Minimum	1
Maximum	100000

中间层在分发过程中，还会检查从库是不是延迟，如果延迟太久，就会放弃分发到这个Slave上面，而这个参数就是用来设置Slave延迟间隔的检测时间的，单位s。

- proxy_format_fullpath

Name	**proxy_format_fullpath**

Scope	Global
Dynamic	NO
Type	boolean
Default	FALSE

这个参数用来设置中间层在格式化业务访问的SQL语句时，是否要统一将表名前面加上路径，即库名，如dbname.table_name，默认是off，建议开启。

- proxy_format_num_per_time

Name	**proxy_format_num_per_time**
Scope	Global
Dynamic	NO
Type	int
Default	10
Minimum	1
Maximum	100000

这个参数用来设置，trace模块中每次批量处理的格式化SQL个数，保持默认值即可。

- proxy_license_key_path

Name	**proxy_license_key_path**
Scope	Global
Dynamic	NO
Type	file name

ArkProxy所用到的合法license文件名，设置时要注意文件访问权限，不然可能会启动不了。

- proxy_multi_write_mode

Name	**proxy_multi_write_mode**
Scope	Global

Dynamic	NO
Type	boolean
Default	FALSE

这个参数用来设置当前分发模式是不是支持多点写入，默认值为FALSE，如果设置为FALSE，则在配置中不支持设置多个在线的写节点，并且在分发的过程中，始终只会向一个写节点分发写操作（包括Hint等），并且如果已经有一个写节点在线了，那么也不支持将另一个写节点上线。
如果设置为TRUE，则Arkproxy支持设置多个写节点，并且也支持将他们都上线，并且在写分发时，会在多个写节点之间，做负载均衡。

- proxy_non_encrypted_ips

Name	**proxy_non_encrypted_ips**
Scope	Global
Dynamic	NO
Type	string

这个参数用来设置中间层对用户名的解析规则，如果在这个名单里面的，则不会对用户名和IP做HASH操作，这样登录中间层和登录后端数据库所使用的IP和用户名可以保持原样，这样是为了方便DBA运维管理，而不可以将业务正常访问的IP加入白名单，不然可能导致不可预知的问题。这个名单IP个数尽量少。具体的权限管理，请参考手册对应小节说明。

- proxy_recv_queue_size

Name	**proxy_recv_queue_size**
Scope	Global
Dynamic	NO
Type	int
Default	100
Minimum	100
Maximum	1000000

Arkproxy针对每一个链接，内部维护了一个从后端到中间层接收消息的队列，这个队列用来

存储后端MySQL向业务客户端返回的消息的，设置稍微大一些，可以一定程度上提高中间层处理性能。

- proxy_send_queue_size

Name	**proxy_send_queue_size**
Scope	Global
Dynamic	NO
Type	int
Default	10
Minimum	1
Maximum	100000

Arkproxy针对每一个连接，内部维护了一个SQL接收队列，这个队列不出意外，只会用到一个节点，所以可以设置到最小，即保持默认即可。

- proxy_server_heartbeat_period

Name	**proxy_server_heartbeat_period**
Scope	Global
Dynamic	NO
Type	int
Default	100
Minimum	1
Maximum	100000

Arkproxy在内部会保持不断地做心跳检测，以保证每一个连接都正常处于活动状态，这个参数用来设置每两次心跳检测的时间间隔。

- proxy_general_log

Name	**proxy_general_log**
Scope	Global

Dynamic	Yes
Type	boolean
Default	OFF

与MySQL参数general_log意义相同。

- proxy_general_log_file

Name	**proxy_general_log_file**
Scope	Global
Dynamic	Yes
Type	file name
Default	host_name.log

与MySQL参数general_log_file意义相同。

- proxy_log_error

Name	**proxy_log_error**
Scope	Global
Dynamic	Yes
Type	file name

与MySQL参数log_error意义相同。

- proxy_shell_listen_port

Name	**proxy_shell_listen_port**
Scope	Global
Dynamic	NO
Type	int

ArkProxy的管理端口，通过这个端口登录，可以管理、查看中间层的状态，连接数量、配置参数等，当然也可以通过一系列命令来设置，控制中间层的行为。具体支持什么命令，请参

阅相关章节。

- proxy_shell_username

Name	proxy_shell_username
Scope	Global
Dynamic	NO
Type	string

中间层设置一个用户名及密码（proxy_shell_password），来让DBA登录到SHELL端口来操作，DBA只需要指定相应的用户名密码，就可以通过mysql客户端登录了。

- proxy_shell_password

Name	proxy_shell_password
Scope	Global
Dynamic	NO
Type	string

与上面参数proxy_shell_username相对应。

trace相关参数：

- proxy_digest_trace

Name	proxy_digest_trace
Scope	Global
Dynamic	NO
Type	boolean
Default	TRUE

ArkProxy的digest trace是否开启开关，默认是TRUE，开启之后，会产生相应的业务访问语句的digest信息。统计之后的信息，会被存储到proxy_config_host:proxy_config_port设置的目标数据库中，虽然可能会对中间层的性能有点影响，但影响不大。

- proxy_sql_trace

Name	proxy_sql_trace
Scope	Global
Dynamic	NO
Type	boolean
Default	TRUE

ArkProxy的sql trace是否开启开关，默认是TRUE，如果开启了，则会将所有的SQL访问都记下来，存储到proxy_config_host:proxy_config_port设置的目标数据库中，不过可能会对中间层的性能有点影响。

- proxy_config_host

Name	proxy_config_host
Scope	Global
Dynamic	NO
Type	string

ArkProxy的digest/sql trace远端MySQL数据库的地址，此参数如果不设置，则默认是中间层对应的后端MySQL库中（写节点）。

- proxy_config_port

Name	proxy_config_port
Scope	Global
Dynamic	NO
Type	int

ArkProxy的digest/sql trace远端MySQL数据库的端口，此参数如果不设置，则默认是中间层对应的后端MySQL库中（写节点）。与参数proxy_config_host成对设置。

- proxy_config_user

Name	proxy_config_user
Scope	Global

Dynamic	NO
Type	string

ArkProxy的digest/sql trace远端MySQL数据库的用户，需要到数据库中赋权限，与proxy_config_host相对应。

- proxy_config_passwd

Name	**proxy_config_passwd**
Scope	Global
Dynamic	NO
Type	string

ArkProxy的digest/sql trace远端MySQL数据库（目前只能是MySQL数据库）密码，目前不支持加密，只能明文存储，与参数proxy_config_user成对设置。

- proxy_format_parameterize

Name	**proxy_format_parameterize**
Scope	Global
Dynamic	NO
Type	boolean
Default	ON

这个参数用来设置，在digest生成时，是否开启将where条件中的值常量值用“？”来替代，如果开启，则中间层会认为语句相同，但常量值不同的语句是同一个语句，这样就会被合并到同一个digest中，如果不开启，则会被认为是不同的语句，生成的是不同的digest。

- proxy_format_thread_num

Name	**proxy_format_thread_num**
Scope	Global
Dynamic	NO
Type	int

Default	5
Minimum	1
Maximum	100000

ArkProxy生成digest时批量处理的线程数，默认是5。

- proxy_flush_thread_num

Name	**proxy_flush_thread_num**
Scope	Global
Dynamic	NO
Type	int
Default	5
Minimum	1
Maximum	10000

ArkProxy的digest中，这个参数用来指定，批量将digest刷入到存储库中的并发线程个数。

- proxy_namespace

Name	**proxy_namespace**
Scope	Global
Dynamic	NO
Type	string
Default	mysql

ArkProxy digest/sql trace存储数据库的名字，如果不指定proxy_config_host等参数，就会默认存储到业务库的mysql库中。如果指定了proxy_config_host参数，就会以proxy_namespace为数据库名，存储到这个数据库中。

- proxy_sql_trace_thread_num

Name	**proxy_sql_trace_thread_num**

Scope	Global
Dynamic	NO
Type	int
Default	5
Minimum	1
Maximum	10000

ArkProxy处理sql trace的线程个数，默认是5

- proxy_trace_busy_strategy

Name	**proxy_trace_busy_strategy**
Scope	Global
Dynamic	NO
Type	int
Default	0
Minimum	0
Maximum	5

当服务运行繁忙的，记录的digest和sql trace是否丢掉，默认丢掉，0表示丢掉，1表示阻塞并等待，其它值目前无效。

- proxy_trace_busy_wait_time

Name	**proxy_trace_busy_strategy**
Scope	Global
Dynamic	NO
Type	int
Default	500
Minimum	1
Maximum	360000000000

这个参数要设置的是，当服务运行繁忙的，如果记录的digest和sql trace设置为等待的话，等待的时间间隔，单位是ms。

- proxy_trace_client_queue_size

Name	**proxy_trace_client_queue_size**
Scope	Global
Dynamic	NO
Type	int
Default	1000
Minimum	1
Maximum	100000

这个参数用来设置，SQL trace在处理任务时的队列大小，收集线程不断地向这个队列中放置，刷出线程不断地从这个队列中取出来，然后写入到后端数据库中。

- proxy_trace_flush_thread_sleep_msec

Name	**proxy_trace_flush_thread_sleep_msec**
Scope	Global
Dynamic	NO
Type	int
Default	500
Minimum	1
Maximum	360000000000

从队列里取数据时，如果队列是空的，则需要等待一段时间，再重试，这个参数就是用来设置这个等待时间间隔的，单位为ms。

- proxy_trace_format_thread_sleep_msec

Name	**proxy_trace_format_thread_sleep_msec**
Scope	Global

Dynamic	NO
Type	int
Default	500
Minimum	1
Maximum	360000000000

中间层在每次批量处理SQL语句的格式化并生成digest之后，都会去休息一小段时间，这样就可以适当地减少中间层的压力，同时考虑到在没有访问，或者访问比较小的时候，收集太过频繁导致出现空转的问题。这个参数就是用来设置暂停时间的，位置为ms。

- proxy_trace_hash_mem_length

Name	**proxy_trace_hash_mem_length**
Scope	Global
Dynamic	NO
Type	int
Default	134217728
Minimum	1
Maximum	549755813888

在sql trace生成时，中间层会使用hash来做去重，以尽可能地减少流量与压力，但这个HASH表所使用的内存空间不能太大，而这个参数就是用来设置这个最大内存空间的，默认是128M，如果超过这个值，则这个缓存不能继续写了，需要flush到后端数据库中之后才可以继续。这个值设置的越大，则会有更多的trace信息做了合并。

- proxy_trace_hash_size

Name	**proxy_trace_hash_size**
Scope	Global
Dynamic	NO
Type	int
Default	200

Minimum	1
Maximum	100000000

这个参数与上面的参数意义差不多，也是用来设置HASH表的容量的，上面设置的是内存占用量，而这个参数设置的是记录个数，如果记录数超过这个设置的值之后，也会做清空操作，然后才可以继续使用。

- proxy_trace_max_delay_time

Name	**proxy_trace_max_delay_time**
Scope	Global
Dynamic	NO
Type	int
Default	3
Minimum	1
Maximum	10000

这个参数与上面两个参数的意义类似，为了防止trace信息及时地刷到后端数据库中，而不能一味地只等待占用内存数量以及记录个数，还需要通过设置这个时间值，来保证trace在超过这个时间之后，即使内存占用比较小，或者记录数很小，也会将trace 信息刷到后端数据库中。

- proxy_trace_queue_array_length

Name	**proxy_trace_queue_array_length**
Scope	Global
Dynamic	NO
Type	int
Default	500
Minimum	1
Maximum	10000000

资源池，每个客户端连接对应一个队列成员，所有连接争抢这个资源，这个队列是用来保证

及加工digest信息的。如果设置太小，可能会引起资源不足，性能下降的问题。

- proxy_trace_sql_bucket_length

Name	proxy_trace_sql_bucket_length
Scope	Global
Dynamic	NO
Type	int
Default	100
Minimum	1
Maximum	100000000

这个参数用来指定，在sql trace缓存中，通过参数proxy_sql_trace_thread_num指定了总共有多少个线程来处理sql trace信息，而参数proxy_trace_sql_bucket_length指定的是，每一个线程要处理的sql trace信息队列的大小。

4 权限管理

Arkproxy的权限管理，与其它中间层是不一样的，所有的变化，都是出于对安全性的考虑，同时也考虑到了用户的友好性。
中间层，作用是将客户端连接请求，转发到合适的后端数据库节点中，并且将对应的返回结果，再原路返回给客户端，这样的架构，就存在三层，分别是客户端、中间层以及后端服务器，他们之间建立的连接，有两层，分别是客户端与中间层的连接，以及中间层与后端数据库的连接。
从运维角度来看，在后端数据库中，比如执行一个简单show processlist;命令，从结果中可以尽可能的看到最原始的连接信息，比如用户名、连接IP等信息，就像我们直连一个MySQL服务器时能看到的那样。
但现在是三层，连接有两层，不管客户端是从多少个IP连接过来，不管用到了什么用户名，在后端服务器，看到的连接IP只有一个，那就是中间层的IP地址。而能看到的用户名，有可能也是固定的一个，也有可能是与客户端连接时所使用的用户名相同，这取决于如何实现这个中间层了，也就是其友好性。
IP是固定的，只能是中间层程序所在机器的IP，这个是没法改变的。而用户名的显示，一般有两种方式来实现。

1. 直接在中间层配置文件中，配置一个在后端所有数据库中具有所有权限的帐号，这样不管任何用户具有任何权限，只要有权限连接中间层，那就有权限做任何操作了，因为真正在后端做操作的，就是在配置文件中配置好的具有所有权限的一个用户，这种情况下，毫无疑问，

在后端数据库中看到的所有连接，都是同一个用户名及同一个IP地址，这样的话，show processlist;在处理问题时，变得毫无意义，因为没办法区分什么语句是从什么IP发过来的，是什么用户名执行的。这种实现方式是最不友好的，因为它将权限无限制的放大了，造成了非常大的安全隐患，极不可取。

2. 另一种实现方式，就是保持原始连接中间层的用户名，但不同用户名的情况下，就需要不同用户名在后端数据库，具有中间层IP的权限了。也就是在后端有很多类似A@127.0.0.1，B@127.0.0.1，等等这样的帐号，但还是有一个问题，如果同一个用户名，具有不同权限的业务，比如针对一个帐号A@127.0.0.1，某一个业务申请了SELECT，另一个业务申请了DELETE，那此时在后端数据库中，A@127.0.0.1帐号就拥有了SELECT和DELETE的权限，那当初申请权限的两个业务当然也同时具有了这两个权限，相当于是把权限合并了。这种情况当然是可以在后端数据库中看到真实用户名的，但还是将权限放大了，还是不安全。当然有一种方法可以解决这个问题，就是保证用户名的唯一性，但很明显，需要要求业务在起名的时候，考虑再三不要唯一，很不友好，也不可取。

Arkproxy能做到的是，权限没有任何放大的可能性，同时连接时都会使用原始用户名，这种解决办法，需要在中间层来创建用户，中间层会在后端创建与之一一对应的用户，但用户名是加密的，加密之后，在后端数据库中的用户名信息都是不会被破解，增强了数据库的安全性，同时也保证了用户名的唯一性，这样客户端使用原始信息连接时，中间层会将其加密后再连接到后端数据库，只要加密之后是同一个用户名，那说明来源用户必然是同一个帐号，权限没有任何放大，权限验证也只发生在后端数据库中，这样的中间层，才是真正的中间层。

在配置中间层时，首先需要设置参数proxy_non_encrypted_ips，这个参数设置之后，只要与这个名单里的用户，都不会对其加密，这样就可以直接使用后端MySQL中已经有的用户登录了。这样是为了更方便的运维，让DBA做到所见即所得，不需要再去转换。

这里我们假设proxy_non_encrypted_ips参数的值为127.0.0.1。

在需要程序帐号的时候，这个程序就需要部署在proxy_non_encrypted_ips之外的机器上面了，此时给这个程序授权，只能在中间层的服务端口来授权。比如：

```
mysql>  grant all privileges on *.* to  proxytest@114.115.128.250 identified by 'proxytest';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

从上面结果看出，这个授权已经成功了，我们可以通过show grants命令，来查看其权限：

```
mysql>  grant all privileges on *.* to  proxytest@114.115.128.250 identified by 'proxytest';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show grants for proxytest@114.115.128.250;
+-----------------------------------------------------------------+
| Grants for 6f67587aca20a657@127.0.0.1                           |
+-----------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO '6f67587aca20a657'@'127.0.0.1' |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

从上面结果看出，显示的用户名，其实并不是“proxytest@114.115.128.250”，而是“'6f67587aca20a657'@'127.0.0.1'”，这是因为在后端数据库中，真正的用户就是'6f67587aca20a657'@'127.0.0.1'，是为中间层开通的权限。而不是为114.115.128.250开通的权限。

此时我们可以在114.115.128.250机器上登录中间层，并且查看其权限，如下所示：

```
[root@hwtest ~]# mysql -uproxytest -pproxytest -h118.190.89.67 -A -P3336
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 10.2.4-log Percona Server (GPL)

Copyright (c) 2009-2017 Percona LLC and/or its affiliates
Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show grants;
+-----------------------------------------------------------------+
| Grants for 6f67587aca20a657@127.0.0.1                           |
+-----------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO '6f67587aca20a657'@'127.0.0.1' |
+-----------------------------------------------------------------+
1 row in set (0.02 sec)

mysql>
```

可以看出，输出的权限结果，与上面授权之后马上查看时，显示的结果完全相同。

当然如果此时想要创建一个用户，其IP地址在参数proxy_non_encrypted_ips中，此时中间层不会对其加密，会直接在后端数据库中创建这个用户。如下图所示：

```
mysql> grant all privileges on *.* to  localtest@127.0.0.1 identified by 'proxytest';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show grants for localtest@127.0.0.1;
+-----------------------------------------------------+
| Grants for localtest@127.0.0.1                      |
+-----------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'localtest'@'127.0.0.1' |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

那可能有人会问，在有业务连接中间层之后，此时如何知道业务用户与加密用户之间的对应关系呢？这点Arkproxy也考虑到了，我们可以在show backend connections命令的结果中看到，我们登录中间层管理端口：

```
mysql> show backend connections\G
*************************** 1. row ***************************
     Parent_Id: 19
          User: proxytest
          Host: 114.115.128.250
      MD5_HASH: 6f67587aca20a657
          Port: 3308
            Db:
    Route_Type: Write
  Wait_Timeout: 900
Last_Heartbeat: 2018-02-26 10:18:54.839924
Last_Execution: 2018-02-26 10:18:54.839924
        Alived: 1
   Env_Version: 0
           QPS: 0
     Slave_Lag: 0
   Running_SQL: NULL
```

从这个结果来看，其中第二列User，对应的就是用户登录时使用的用户名，而第四列，MD5_HASH对应的才是后端数据库中存储的真实用户名，所以可以在业务运行过程中，通过这个命令来找他们的对应关系。具体关于这个命令详细的说明，请参考相关章节。

5 特性

5.1 兼容MySQL语法

目前ArkProxy兼容99.9%的MySQL语法。不支持的还未可知。

5.2 限制

不允许在中间层中执行的命令如下：

1. start slave
2. stop slave
3. start all slaves
4. stop all slaves
5. change master to ...
6. select for update 也可能会走从库，使用select for update必须在显示事务中或者加hint强制走Master

5.3 获取上下文语句的限制

对于found_rows、row_count以及show warning、select last_insert_id等获取上下文信息的语句，仅在产生该类信息的语句(比如SQL_CALC_FOUND_ROWS语句、insert auto increment等)正确执行后立即执行有效，否则结果无效。

5.4 temporary表限制

temporary表的相关操作执行结果可能错误，temporary表在不同的session中查询结果不同。

5.5 不支持 load data

目前不支持load data操作

6 ArkProxy管理

目前ArkProxy支持对以下功能模块的管理：

- 透明读写分离
- 权重分发
- 友好性
- Hint分发
- 智能统计
- 在线配置Reload
- 户连接数限制及统计
- 连接池

下面章节便依次展开介绍各个模块的功能以及用法。

6.1 管理Command

- **Command说明**

命令系列	说明
/*!999999 route to {write|read}*/	中间层的hint使用方式，在每条语句的最前面加上这些信息，就可以主动让ArkProxy将查询路由到写节点或者读节点，适用于对于主从延迟敏感的业务
show backend servers	查看后端配置数据库的状态
show backend routes	查看后台server对应route的状态
show backend connections	查看ArkProxy链接后端的信息
show user config list	可以查看配置user的信息
show config cache	查看当前缓存中的配置信息
show trace status	查看当前中间层关于trace状态的信息
show trace full status	查看当前中间层关于trace状态的更详细的信息
show variables local	显示管理用户的所有variables
config help	显示所有的管理端口的命令帮助
config add {read|write} server {servername} [server_attribute [,server_attribute] ...]; server_attribute: {host='string'|port=integer|max_lag=integer|weight=integer}	增加一个server节点，在server_attribute中指定了这个节点的属性。新增节点状态都是OFFLINE，如果需要上线，则将其手动上线。
config write outfile '{filename}'	将当前配置，写出到外部文件中，文件名通过filename来指定。
config set server {servername} online/offline	将指定server上下线
config set user '{username}'@'{userip}' max_user_connections={integer}	在有权限的基础上，可以再限制某用户可连接的最大连接数
config set user '{username}'@'{userip}' datab	在有权限的基础上，可以再限制可连接的

ases={dbname}	后端数据库，不过这个功能暂未支持
config delete {integer}	用来删除某个设置项的。integer表示序号，序号需要从show config cache中拿到。
config {delete all|reset}	表示当前未生效的配置缓存都不要了
config reload	将外部配置文件中的内容reload到内存。当手动修改配置文件中之后，可以通过config reload让其生效。
config flush	设置完后，此时所有的配置还未生效，只是缓存到当前会话中的，需要通过config flush命令让其生效。

6.2 透明读写分离

- 查看server端配置

  可以通过下边的命令：

```
mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status  | Comments        |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
|  1 | server1 | 127.0.0.1 | 3306 |      3 |           200 | Yes    | ONLINE  | day day up      |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | OFFLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
2 rows in set (0.01 sec)

mysql>
```

column	含义
Weight	当前配置的服务的权重
Max_Slave_Lag	当从库超过相应值后就不路由
Routed	是否配置了路由
Comments	备注

  从输出结果中，我们能够知道，那些后端配置的Server正在提供服务，以什么权重来进行路由，何时不路由。而且用户只需要知道ArkProxy的ip就可以进行访问，无需知道后端数据库架构。

- 查看后端server的路由情况

column	含义
Route_Type	Write/read路由

目前配置的只有Server1提供服务

```
mysql> show backend routes;
+----+---------+-----------+------+------------+-----------+
| Id | Name    | Host      | Port | Route_Type | Comments  |
+----+---------+-----------+------+------------+-----------+
|  1 | server1 | 127.0.0.1 | 3306 | Write      | NULL      |
|  2 | server2 | 127.0.0.1 | 3309 | Read       | xxxxxxxxx |
|  3 | server1 | 127.0.0.1 | 3306 | Read       | xxxxxxxxx |
+----+---------+-----------+------+------------+-----------+
3 rows in set (0.00 sec)

mysql>
```

- 查看目前后端连接用户数及详细情况

结果集column名字	含义
Parent_Id	表示当前中间层与客户端的thread_id。
User	表示当前程序连接到中间层的原始用户名。
Host	表示当前程序连接到中间层的来源IP地址。
MD5_HASH	表示当前程序连接到中间层的被加密用户名。如果与上面的User相同，则说明当前的连接来源host在白名单中（参数proxy_non_encrypted_ips）。
Port	表示当前连接到的数据库端口。
Db	当前连接的dbname，如果不指定，则为空
Route_Type	Write
Wait_Timeout	超时断开时间28800s
Last_Heartbeat	上一次心跳时间
Alived	当前状态
Env_Version	连接中环境变量的版本中，这个可以不用关心。

QPS	当前session的qps
Slave_Lag	延迟时间
Running_SQL	当没有运行的SQL，如果有，就是当前正在跑的SQL

能够清楚的知道当前连接ArkProxy的client端信息，并通过信息来进行诊断

```
mysql> show backend connections\G
*************************** 1. row ***************************
 Parent_Id: 10
      User: test_1
      Host: localhost
  MD5_HASH: test_1
      Port: 3306
        Db: test1
Route_Type: Write
Wait_Timeout: 28800
Last_Heartbeat: 2017-12-11 17:40:32.513054
    Alived: 1
Env_Version: 0
       QPS: 1
 Slave_Lag: 0
Running_SQL: NULL
1 row in set (0.00 sec)
```

4.3 权重分发

可以在my.cnf中，配置每个server的权限比重

```
##weight参数就是权重比，是按照逻辑比例来分的，设置1000和1000也还是50%的比例进行分发
[server1]
proxy_type=server
backend_host=127.0.0.1
backend_port=3306
weight=50

[server2]
proxy_type=server
backend_host=127.0.0.1
backend_port=3315
weight=50
```

当然可以通过ArkProxy内部查看权重，也可以通过配置文件来查看

6.4 友好性

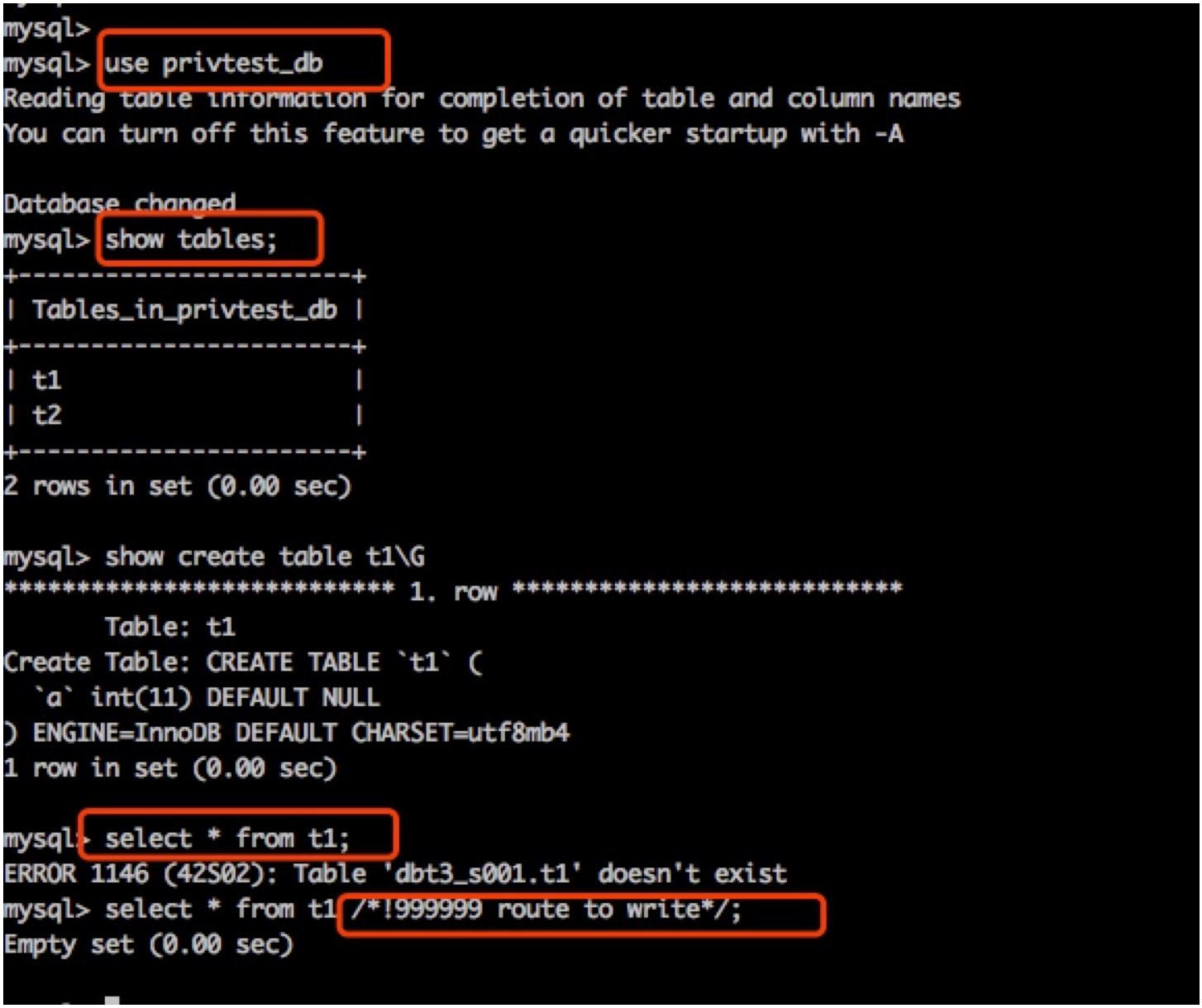
用户连接近ArkProxy，就会误以为进入到了真正的后端数据库。命令也基本都兼容。
在管理用户界面，增加了ArkProxy的参数，而且只需要输入命令config help，就能出现所有的在线管理命令，如下图所示。

```
mysql> config help\G
*************************** 1. row ***************************
         Id: 1
    Command: /*!999999 route to {write|read}*/ ...
Description: 中间层的hint使用方式，在每条语句的最前面加上这些信息，就可以主动让ArkProxy将查询路由到写节点或者读节点，适用于对于主从延迟敏感的业务
*************************** 2. row ***************************
         Id: 2
    Command: config reload
Description: 将外部配置文件中的内容reload到内存。当手动修改配置文件中之后，可以通过config reload让其生效。
*************************** 3. row ***************************
         Id: 3
    Command: config set server {servername} online/offline
Description: 将指定server上下线
*************************** 4. row ***************************
         Id: 4
    Command: show backend servers
Description: 查看后端配置数据库的状态
*************************** 5. row ***************************
         Id: 5
    Command: show backend connections
Description: 查看ArkProxy链接后端的信息
*************************** 6. row ***************************
         Id: 6
    Command: show user config list
Description: 可以查看配置user的信息
*************************** 7. row ***************************
         Id: 7
    Command: show config cache
Description: 查看当前缓存中的配置信息
*************************** 8. row ***************************
         Id: 8
    Command: show trace status
Description: 查看当前中间层关于trace状态的信息
*************************** 9. row ***************************
         Id: 9
    Command: show trace full status
Description: 查看当前中间层关于trace状态的更详细的信息
*************************** 10. row ***************************
         Id: 10
    Command: show backend routes
Description: 查看后台server对应route的状态
*************************** 11. row ***************************
         Id: 11
    Command: show variables [like %name%]
Description: 显示管理用户的所有或指定variables
*************************** 12. row ***************************
         Id: 12
    Command: config help
Description: 显示所有的管理端口的命令帮助
*************************** 13. row ***************************
         Id: 13
    Command: config add {read|write} server {servername} [server_attribute [,server_attribute] ...]; server_attribute: {host='string'|port=integer|max_lag=integer|weight=integer}
Description: 增加一个server节点，在server_attribute中指定了这个节点的属性。新增节点状态都是OFFLINE，如果需要上线，则将其手动上线
*************************** 14. row ***************************
         Id: 14
```

```
    Command: config write outfile '{filename}'
Description: 将当前配置，写出到外部文件中，文件名通过filename来指定。
*************************** 15. row ***************************
         Id: 15
    Command: config set user '{username}'@'{userip}' max_user_connections={integer}
Description: 在有权限的基础上，可以再限制某用户可连接的最大连接数
*************************** 16. row ***************************
         Id: 16
    Command: config set user '{username}'@'{userip}' databases={dbname}
Description: 在有权限的基础上，可以再限制可连接的后端数据库，不过这个功能暂未支持
*************************** 17. row ***************************
         Id: 17
    Command: config delete {integer}
Description: 用来删除某个设置项的。integer表示序号，序号需要从show config cache中拿到。
*************************** 18. row ***************************
         Id: 18
    Command: config {delete all|reset}
Description: 表示当前未生效的配置缓存都不要了
*************************** 19. row ***************************
         Id: 19
    Command: config reload
Description: 将外部配置文件中的内容reload到内存。当手动修改配置文件中之后，可以通过config reload让其生效。
*************************** 20. row ***************************
         Id: 20
    Command: config flush
Description: 设置完后，此时所有的配置还未生效，只是缓存到当前会话中的，需要通过config flush命令让其生效。
```

6.5 Hint分发

这是连接ArkProxy的server层来进行分发的，不是在Admin命令窗口。
默认所有的select都会分发到非写节点，所以当有对延迟响应低的要求很高的业务线，
可以通过Hint来实现，立即提交，立即可读。通过截图可以更直观地观察：



使用方法特别简单。

但可能有人会问，这怎么说明了这个功能的有效呢？很明显，做测试的这个中间层，所指定的后端数据库，并不是真正的主从，其实没有任何关系，所以数据都是不同的，第一个select * from t1;报出了表不存在的错误，而第二次在指定了hint之后，没有报这个错误，则说明他路由到了指定的write节点。

6.6 智能统计

通过配置可以开启trace，开启后性能会有损耗，如果配置了远端trace数据库，性能损耗会更小，毕竟不占用本地数据库的QPS。

```
proxy_digest_trace=on
proxy_sql_trace=on
```

打开两个参数后，通过digest是按照某类SQL进行归并的，所以一眼就能看到其中SQL的使用比

例，以及执行时间等。
通过sql trace可以看到具体的SQL语句，可以在该基础上进行二次开发分析，将有用的信息整理不同维度的数据，供后边（日常诊断、日常预警等）分析。

trace参数非常丰富，可以根据业务调整到最佳。如果不清楚怎么配置，默认即可。

统计部分基本上都是在digest和sql trace部分在配置的远端库中：

登陆到远端数据库查看：

```
mysql> show tables;
+---------------+
| Tables_in_tes |
+---------------+
| proxy_user    |
| query_digest  |
| query_sql     |
+---------------+
```

digest的统计情况：

```
mysql> select * from query_digest limit 10;
+----+------------+----------+----------------------------------+-----------------------------------------------------------------------------------------------------------------+--------+---------------+---------------+----------+----------+----------+
| id | schemaname | username | digest                           | formatsql                                                                                                       | count  | first_seen    | last_seen     | sum_time | min_time | max_time |
+----+------------+----------+----------------------------------+-----------------------------------------------------------------------------------------------------------------+--------+---------------+---------------+----------+----------+----------+
|  1 |            | test_1   | BAFAFD71754D3836836F6D935269EC53681FABA7 | SELECT  @@version_comment LIMIT ?                                                                               |     13 | 1511929171315033 | 1512992436850967 |     3393 |      181 |      406 |
|  2 |            | test_1   | 1122F36226C43A87D4DAC9B50467BE2DD648C1F0 | SELECT database()                                                                                               |      6 | 1511929176060377 | 1512988613189016 |     1206 |      163 |      267 |
|  3 | test1      | test_1   | 2C16BE04F1FEB1DFDCAB69B2F881519 42ED04EF1 | SELECT * FROM test1.sbtest1                                                                                     |      1 | 1511929183302096 | 1511929183302096 |  1504111 |  1504111 |  1504111 |
|  5 | test       | test_1   | 4FD9B629E719B6091FD36B129DDC772844061CE3 | SELECT * FROM test.test                                                                                         |      3 | 1511929211408853 | 1511929241373931 |     2206 |      537 |     1073 |
|  7 | test8      | test_1   | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM test8.sbtest1 WHERE id BETWEEN ? AND (?+?)                                                        | 1178301 | 1511934287142301 | 1512994155154124 |  934737047 |       18 |    54294 |
|  8 | test8      | test_1   | 8D0D9E16299E250058A0AE5529A8898535106F3C | SELECT c FROM test8.sbtest1 WHERE id=?                                                                          | 1184436 | 1511934287140497 | 1512994155154113 |  421667691 |        5 |    56600 |
|  9 | test8      | test_1   | 7853D008409851F280FCE288EA278FE8F7A9B151 | SELECT SUM(K) FROM test8.sbtest1 WHERE id BETWEEN ? AND (?+?)                                                   | 1179513 | 1511934287145839 | 1512994155155903 |  795990084 |        5 |    72260 |
| 11 | test8      | test_1   | F870A7555F264CB7745340A86D08AF9E2C040BD7 | SELECT c FROM test8.sbtest1 WHERE id BETWEEN ? AND (?+?) ORDER BY c                                             | 2376970 | 1511934287147630 | 1512994155159219 | 2543013764 |       22 |    98195 |
| 12 | test8      | test_1   | 462D8BE0C9B2664D9456CA5433A03A9A41CE8A43 | DELETE FROM test8.sbtest1 WHERE id=?                                                                            |  397519 | 1511934287152937 | 1512994155159231 |  255860192 |        5 |    99139 |
| 13 | test8      | test_1   | 3B32B6ED4C795DBA484838800BAB2C5BD54196C2 | UPDATE test8.sbtest1 SET k=(k+?) WHERE id=?                                                                     |  397849 | 1511934287152923 | 1512994155159223 |  424653646 |        6 |    55536 |
+----+------------+----------+----------------------------------+-----------------------------------------------------------------------------------------------------------------+--------+---------------+---------------+----------+----------+----------+
10 rows in set (0.00 sec)
```

如果要看具体的SQL，可以统一digest列，进行where过滤：

```
mysql> select * from query_sql where digest_key='EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0' limit 10;
+-----+------------+-----------+----------+----------------------------------+----------------------------------+-----------------------------------------------------------------------------+-------------+---------------+----------+
| id  | schemaname | client_ip | username | sql_key                          | digest_key                       | origin_sql                                                                  | affect_rows | appear_ts     | exe_time |
+-----+------------+-----------+----------+----------------------------------+----------------------------------+-----------------------------------------------------------------------------+-------------+---------------+----------+
|  11 | test8      | 127.0.0.1 | test_1   | 86760B2CB406970C22267EFE52CC8C87A31CB732 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 449867 AND 449867+99 |           0 | 1511934287142302 |      888 |
|  12 | test8      | 127.0.0.1 | test_1   | A7441DBF771DCFEA769D3B7196CBE53CF6322AA8 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 448497 AND 448497+99 |           0 | 1511934287144060 |      590 |
|  13 | test8      | 127.0.0.1 | test_1   | EE98DA0A5D8EBE286FF63DA761EAB4F43DDE4319 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 356416 AND 356416+99 |           0 | 1511934287144065 |      548 |
|  49 | test8      | 127.0.0.1 | test_1   | 6F741A701EC28D123910C81ED78A21F5FB1C3CFC | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 430784 AND 430784+99 |           0 | 1511934287168747 |      479 |
|  50 | test8      | 127.0.0.1 | test_1   | 1F7D9F745064800E714870D7AEDD5DD532C2C6A7 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 452209 AND 452209+99 |           1 | 1511934287168750 |      436 |
|  51 | test8      | 127.0.0.1 | test_1   | 2B295F2571739B9D75D27BE769415BA1C68ZF130 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 358970 AND 358970+99 |           0 | 1511934287168753 |      468 |
| 163 | test8      | 127.0.0.1 | test_1   | E938C1839AAF8A3070750FB6349AC069421C16D4 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 456372 AND 456372+99 |           0 | 1511934287223657 |      393 |
| 164 | test8      | 127.0.0.1 | test_1   | B81BB74244CED70A87EFC6FBDDC32661250848BD | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 447117 AND 447117+99 |           1 | 1511934287223660 |      347 |
| 165 | test8      | 127.0.0.1 | test_1   | BF9FB7AE88F5D6763D5DF8B8CB8FACE997312492 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 449162 AND 449162+99 |           0 | 1511934287223663 |      382 |
| 182 | test8      | 127.0.0.1 | test_1   | 2BDDF4E6EB3F74D673778C67082A9E728E521170 | EA0E47A1C0664FD82986BF23ADC45C1B3B8B7AB0 | SELECT c FROM sbtest1 WHERE id BETWEEN 450436 AND 450436+99 |           1 | 1511934287232162 |      382 |
+-----+------------+-----------+----------+----------------------------------+----------------------------------+-----------------------------------------------------------------------------+-------------+---------------+----------+
10 rows in set (0.00 sec)
```

可以很清晰的看到，执行时的时间点，执行时间等。

6.7 在线配置Flush

如果我们想在线下线某个Server，我们可以在管理端口直接通过命令设置即可，无需要修改配置，也不需要重启服务。对业务没有任何影响。如下图所示：

```
mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status  | Comments        |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE  | day day up      |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | OFFLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
2 rows in set (0.00 sec)

mysql> config set server server2 online;
Query OK, 0 rows affected (0.00 sec)

mysql> show config cache;
+----+--------------+------------+-----------+-----------+
| Id | Config_Class | Config_Name | New_Value | Old_Value |
+----+--------------+------------+-----------+-----------+
|  4 | SERVER_STATUS | server2    | ONLINE    | OFFLINE   |
+----+--------------+------------+-----------+-----------+
1 row in set (0.00 sec)

mysql> config flush;
Query OK, 0 rows affected (0.00 sec)

mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status | Comments        |
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE | day day up      |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | ONLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
2 rows in set (0.00 sec)

mysql>
```

最开始Server2节点是下线状态的，我们想让其上线服务，此时需要通过 config set server server2 online;将其上线，不过执行这个命令之后，这个服务器状态还处于下线状态，需要通过config flush让其生效，当然此时可以通过show config cache来查看当前已经设置过的，但还未生效的配置。当flush之后，我们再次查看backend servers的状态，可以看到已经上线了。

如果我们将在线服务的配置更改了，想让其在外部配置文件中持久化，此时我们可以通过另一个命令，让其写入到外部文件，命令如下：

```
mysql> config write outfile '/tmp/proxy.cnf';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

此时可以查看生成的文件，已经有了：

```
[root@iZm5efufdvqjhvxtbccbnrZ test]# ll /tmp/proxy.cnf
-rw-rw---- 1 root root 3061 2月   26 10:37 /tmp/proxy.cnf
```

6.8 在线配置reload

我们可以在线修改配置，也可以通过手动修改配置文件，然后让其生效，此时需要用到的命令是config reload。举例如下：
我们先查看一下Server的状态：

```
mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status | Comments        |
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE | day day up      |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | ONLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+--------+-----------------+
2 rows in set (0.00 sec)
```

可以发现，Server2的状态为上线状态，此时我们修改一下配置文件，将其修改为下线状态：

```
[server2]
proxy_type=server
backend_host=127.0.0.1
backend_port=3309
config_comment=good good study
max_slave_lag=100
weight=3
server_status=offline
```

然后执行config reload：

```
mysql> config reload;
Query OK, 0 rows affected (0.01 sec)

mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status  | Comments        |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE  | day day up      |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | OFFLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+---------+-----------------+
2 rows in set (0.00 sec)

mysql>
```

此时可以看到，Server2的状态已经变为下线状态了。此时业务连接也不会被路由到这个节点上面去了，这一点可以通过在Server2上面打开general log来验证，这里就不做验证了。

6.9 在线增加节点

有时候我们也想在线增加节点，Arkproxy已经支持了这个功能，这样给自动化运维工作提供了更多的操作方式：

```
mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+--------+----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status | Comments       |
+----+---------+-----------+------+--------+---------------+--------+--------+----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE | day day up     |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | ONLINE | good good study |
+----+---------+-----------+------+--------+---------------+--------+--------+----------------+
2 rows in set (0.00 sec)

mysql> config add read server server3 host ='127.0.0.1', port=3309, max_lag=10000, weight=9;
Query OK, 0 rows affected (0.00 sec)

mysql> show config cache;
+----+--------------+-------------+-----------------------------------------------------------------------------------------------
-------------------------------------------+-----------+
| Id | Config_Class | Config_Name | New_Value
                                          | Old_Value |
+----+--------------+-------------+-----------------------------------------------------------------------------------------------
-------------------------------------------+-----------+
|  5 | ADD_SERVER   | server3     | {"server_name":"server3","route_type":"Read","host_name":"127.0.0.1","
port":"3309","max_lag":"10000","weight":"9"} | NULL      |
+----+--------------+-------------+-----------------------------------------------------------------------------------------------
-------------------------------------------+-----------+
1 row in set (0.00 sec)

mysql> config flush;
Query OK, 0 rows affected (0.00 sec)

mysql> show backend servers;
+----+---------+-----------+------+--------+---------------+--------+---------+----------------+
| Id | Name    | Host      | Port | Weight | Max_Slave_Lag | Routed | Status  | Comments       |
+----+---------+-----------+------+--------+---------------+--------+---------+----------------+
|  1 | server1 | 127.0.0.1 | 3308 |      3 |           200 | Yes    | ONLINE  | day day up     |
|  2 | server2 | 127.0.0.1 | 3309 |      3 |           100 | Yes    | ONLINE  | good good study |
|  3 | server3 | 127.0.0.1 | 3309 |      9 |         10000 | Yes    | OFFLINE | NULL           |
+----+---------+-----------+------+--------+---------------+--------+---------+----------------+
3 rows in set (0.00 sec)

mysql>
```

从上面的操作流程可以看到，通过命令增加一个新的节点server3，通过show config cache来查看缓存情况，New_value表示的是新增节点，内容通过Json来表示，确认无误之后，就通过config flush来将其加到全局配置中，此时可以通过show backend servers来查看结果，但我们发现，其状态是OFFLINE的，这是没错的，新增的节点默认都是下线状态的，需要我们再次确认之后，将其上线提供服务。

此时我们再可以查看一下路由状态：

```
mysql> show backend routes;
+----+---------+-----------+------+------------+-----------+
| Id | Name    | Host      | Port | Route_Type | Comments  |
+----+---------+-----------+------+------------+-----------+
|  1 | server1 | 127.0.0.1 | 3308 | Write      | NULL      |
|  2 | server2 | 127.0.0.1 | 3309 | Read       | xxxxxxxxx |
|  3 | server1 | 127.0.0.1 | 3308 | Read       | xxxxxxxxx |
|  4 | server3 | 127.0.0.1 | 3309 | Read       | xxxxxxxxx |
+----+---------+-----------+------+------------+-----------+
4 rows in set (0.00 sec)
```

我们发现有四条路由规则，有两种路由策略，分别是Write、Read，此时我们可以将当前配置写出来外部配置文件中，查看其变化：

```
mysql> config write outfile '/tmp/proxy.cnf';

Query OK, 0 rows affected (0.00 sec)
```

我们会发现，配置文件中，多了一个Server3：

```
[server3]
proxy_type                              = server
backend_host                            = 127.0.0.1
backend_port                            = 3309
max_slave_lag                           = 10000
server_status                           = OFFLINE
weight                                  = 9
```

并且在route2中，多了一个路由规则：

```
[router2]
proxy_type                              = router
router_type                             = readonly
router_servers                          = server2,server1,server3
config_comment                          = xxxxxxxxx
```

没错，在router_servers中增了一个server3。

6.10 户连接数限制及统计

除了权限控制之外，还可以对单个用不进行限制。
这样可以有效的控制某个业务出现问题，把ArkProxy/MySQL打满，甚至直接把资源用尽的情况。
在打到MySQL层之前，可以在ArkPorxy进行拦截并阻塞其请求。

```
mysql> config set user 'test_1'@'127.0.0.1' databases='mysqltest1' MAX_USER_CONNECTIONS=0;
Query OK, 0 rows affected (0.00 sec)

mysql> config flush;
Query OK, 0 rows affected (0.01 sec)

mysql> config set user 'test_1'@'127.0.0.1' databases='mysqltest1' MAX_USER_CONNECTIONS=100;
Query OK, 0 rows affected (0.00 sec)

mysql> config flush;
Query OK, 0 rows affected (0.00 sec)
```

高可用方案

中间层对于后端数据库的可用性，不做判断，因为中间层的角色，并不太适合去把后端数据库的切换工作也负责起来，但中间层尽可能的考虑了这些问题，并且给出了比较方便的操作方法，这里大概分两种情况：

支持多点写入

这种情况，也就是当参数proxy_multi_write_mode为ON的时候，因为后端数据库支持多点写入，而中间层就不需要考虑是不是会出现多写的问题，此时在配置中，也就更加灵活，可以配置多个写节点，也可以将它们上线，当然，也可以做负载均衡，当某一个节点出现问题的时候，只需要数据库集群的监控程序，将其通过arkproxy提供的命令，下线即可，这样就不会将请求分发到那个节点，故障即可恢复。

不支持多写

这种情况，也就是当参数proxy_multi_write_mode为OFF的时候，众所周知，这种问题比较棘手，目前各种切换方案层出不穷，当然Arkproxy还是不能替你做切换的决定，只能是辅助业务更好的做切换，这要依赖于你的业务架构方案了，当你最终确定切换的那一刻，还是通过Arkproxy给你提供的友好的命令，不管是在线config，还是直接修改配置文件再去reload，都是可以达到切换的目的的。