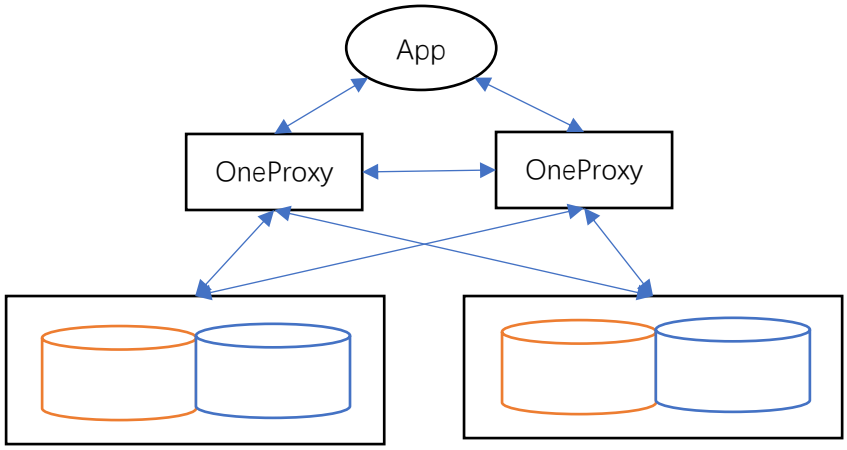
OneProxy 是一个高效稳定的 MySQL 协议层代理软件，可以透明地支持 MySQL 架构的横向扩展。它具备以下令人心动的功能特点：

- 超 10 个不同行业用户已经稳定使用长达 5 年之久，商业软件品质保障。
- 低时延，每个查询增加不到 100us 的时延，远超其他同类 Proxy 软件。
- 稳定性，软件采用 C&C++语言编写，无内存 GC 问题，多个用户场景验证超过一年不用重起。
- 高性能，代理需要解释 SQL 语句，是 CPU 消耗型的操作，每个核每秒能处理超 30000+的查询转发，良好的单机扩展能力，也可构建 Proxy 集群做横向扩展。
- 后端连接池功能，可以有效控制到 MySQL 节点上的总连接数。
- 支持 MySQL Group Replication 和 Percona Xtradb Cluster 集群架构，自动快速识别后端节点切换识别主节点。
- 支持读写分离等多种流量转发策略，具备应用无感知的读写分离扩展能力。
- 支持分库分表，并有跨节点的结果集聚合操作支持能力，对应用的透明度较高。
  - 支持 Int、Big Int、Char、Date、Timestamp 类型的单列分区。
  - 支持 Range、Hash、List 分区方式，支持二级子分区。
  - 支持分区信息冗余到不同的字段，以更好地进行分区过滤。
  - 支持跨结点的结果集合并及排序（order by）。
  - 支持跨节点的结果集汇总操作（count/sum/max/min）。
  - 支持跨节点的结果集的分组汇总（group by）。
  - 支持跨节点的结果集分页操作（limit/offset）。
  - 支持基于分片的并行查询操作（parallel query）。
- 支持结果集缓存，透明提升查询操作性能。
- 支持前后端密码分离，确保数据库密码不外泄。
- 支持 IP 白名单，支持表级别安全设置，支持 SQL 防火墙，阻止 SQL 注入式攻击。
- 内置 HA 机制，无须安装配置第三方软件，实现代理节点的快速故障转移。
- 具备丰富的 SQL 性能统计信息，内置 Http 服务，可以轻松查看性能数据。
- 内置序列号生成器，可以高效生成单向增长的序列值。
- 支持 MySQL 8 协议，支持最新版本的 MySQL JDBC 驱动程序。

今天大量的企业已在使用 MySQL，也熟悉了分库分表操作，一个好的数据访问层（DAL）可以起到事半功倍的作用，OneProxy 就是一个经得起考验的优秀数据访问层软件。

要理解 OneProxy 的关键配置项，就需要了解它的基本布署架构，如下图所示：



由于要支持分库分表，所以需要支持挂载多个 MySQL 主备或 MGR 集群，每个集群需要有唯一的名字，可以称之为组名（Group Name）。每个集群可以是一主多从多个结点，也可以是 MGR 或 PXC 集群。怎么样来添加所有的节点？只要提供三个关键的信息：

- MySQL 节点 IP 地址
- MySQL 节点端口号
- 节点所在集群的名字（Group Name）

配置节点可需要到以下两个选项：

- proxy-master-addresses.[1-256] = 节点 IP:端口@集群的名字
- proxy-slave-addresses.[1-256] = 节点 IP:端口@集群的名字

需要注意的是，这里并不需要提供 MySQL 数据库的名字。MySQL 数据库名字在登录信息中提供，需要包含以下 4 个信息：

- 集群的名字（Group Name）
- 登录用户名
- 登录密码（用自带 mysqlpwd 加密，MySQL 8 需要用 mysql_native_password 认证）
- 登录数据库名字

配置登录信息需要用到以下两个选项（前者指定前端登录密码，后者指定后端真实密码）：

- proxy-user-list.[1-256] = [集群的名字:]用户名/加密后口令@数据库名字
- proxy-user-group.[1-256] = 集群的名字:用户名/加密后口令@数据库名字

前者指定了应用连接 OneProxy 的登录信息，后者指定了某个集群的后端登录信息。如果对同一个用户名，在两个地方都指定了信息，则起到前后端密码分离的作用。

许多HA软件(包括MGR)都会通过设置“read_only”变量来标识节点是读写节点(Primary)还是只读节点（Slave），可以在 OneProxy 中设置以下变量值为 1 来自动识别读写状态，以设置准确的节点类型:

- proxy-auto-readonly = {0|1}

OneProxy 会以 50ms 的频率去检查后端节点状态，基本上是应用无感的秒切。只要 HA 或 MGR 集群能保证同一时间点只有一个节点会关闭“read_only”变量，就不会存在双写情况。这样可以省去通知前端应用写库切换的工作，在大规模应用布署下，这个通知机制还是比较复杂的，使用 OneProxy 则简洁多了。

只要再提供其他几个关键的选项，就可以启动 OneProxy 来进行测试了。如下所示:

- mysql-version = 虚拟版本号（应当是所有节点中最小的 MySQL 版本号）
- event-threads = 允许的 CPU 核数（一般不超过 16，平均每核 2.5 万 QPS 转发）
- proxy-address = 监听地址（默认是“:3307”，等同于 MySQL 的 Listener 地址）
- proxy-group-policy = 集群的名字:流量策略（master-only 或 read-balance）

接下来就可以来准备一个完整的配置文件（conf/proxy.conf）了，如下所示:

```
[oneproxy]
#proxy-address              = :3307
mysql-version               = 8.0.27
event-threads               = 2

proxy-auto-readonly         = 1
proxy-group-policy          = default:read_balance
proxy-master-addresses.1    = 127.0.0.1:3306@default
proxy-user-list.1           = default:test/password@test
```

OneProxy 会自动启用一个守护进程，以保证单实例的高可用，如下所示:

```
[root@RH6SRV1 ~]# ps -ef | grep oneproxy
root      6545     1  0 Mar29 ?        00:00:00 /data/oneproxy/bin/oneproxy
root      6546  6545  3 Mar29 ?        00:30:15 /data/oneproxy/bin/oneproxy
root      8603  8587  0 00:41 pts/2    00:00:00 grep oneproxy
```

接下来就可以开始测试了，如查你用 sysbench 工具，则需要加上“--db-ps-mode = disable”选项（在分库分表的情况下，Prepare Statement 的管理过于复杂，并且 MySQL 目前并不能从 Prepare Statement 模式得到多少性能提升，其 jdbc 驱动默认也未开启)。如果需要停止 OneProxy，请用“kill -INT”命令向子进程发送 INT 信号。

现在可以将 OneProxy 当成 MySQL 节点，将连接信息配置成它的监听地址即可。

OneProxy 在设计时充分考虑了安全性，可以在整个 Proxy 实例层面、集群层面、表层面指定安全级别。可以通过以下几个选项来灵活控制：

- proxy-security-level = level
- proxy-group-security = 集群的名字:level
- proxy-table-security = 表名:level

这里“level”是一个整数，不同的标志位表示不同的含义，其定义如下表所示：

标志位	含义
0x01	禁止 DDL 操作
0x02	针对 Update 和 Delete 操作，必须有明确的 Where 条件
0x04	不允许 Delete 操作
0x08	不允许 DML 操作，只允许查询语句

集群级别的默认值是“1”，即禁止 DDL 操作，除非在可信的 IP 范围内（一般情况下应用代码中是不会有 DDL 操作的，不排除少数应用代码中有 Truncate 操作）。默认情况下本地连接（“127.0.0.1”和“unix_socket”）为可信连接，远程连接可以通过如下参数进行设置：

- proxy-secure-client.[1-255] = IP 地址

比如 DBA 常用的登录机器，可以设置为可信地址，以便日常管理。还可以通过开启 IP 白名单功能来进行 IP 维度的访问控制。可以使用以下两个选项：

- proxy-enable-ipfirewall = {0|1}
- proxy-firewall-ip = IP 列表文件（一行一个 IP 地址，在起动时加载）

也可以临时禁止某个已经访问过的 IP 地址的访问，将其设置为拒绝模式，可以登录 OneProxy 管理端口（默认用户名 admin，默认密码 OneProxy，默认端口 4041，都可以定制）使用以下两个命令进行控制：

- set blackip '192.168.0.1'，禁止某个 IP 的后续登录。
- set greenip '192.168.0.1'，允许某个 IP 的后续登录。

可以通过”list ipqos”命令来查看有哪些 IP 地址登录过 OneProxy，如下图所示：

```
mysql> list ipqos;
+-------------+------+------+------+------+------+------+------+------+------+------+------+------+
| ADDRESS     | DENY | Sess | QoS  | T01  | T02  | T03  | T04  | T05  | T06  | T07  | T08  | T09  | T10  |
+-------------+------+------+------+------+------+------+------+------+------+------+------+------+
| 127.0.0.1   |    0 |    1 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |
| unix_socket |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |    0 |
+-------------+------+------+------+------+------+------+------+------+------+------+------+------+------+
2 rows in set (0.00 sec)
```

第一列表示是否禁止登录，第二列表示有多少个连接，后续字段是过去 10 个秒级时间点的实时 QPS 情况。

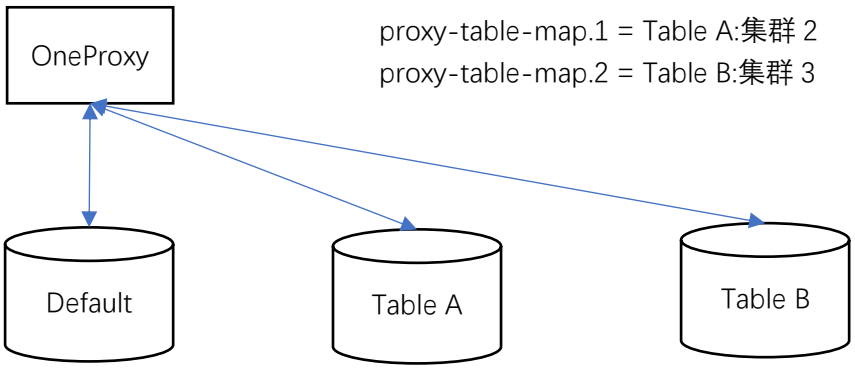
OneProxy 在设计时做了一些功能取舍，以便更简洁地在一个软件中同时支持读写分离和分库分表，也就是对应用还是有一些要求，如果不满足则无法使用 OneProxy 来做数据访问层了。主要的取舍如下：

- 不支持 Prepare Statement 接口。
  - 对于 JDBC，默认未使用此接口，可以运行。
  - 对于 PHP PDO，可以加上“PDO::ATTR_EMMULATE_PREPARE”连接属性。
  - 对于 Go 的 MySQL 驱动，可以加上“interpolateParams”连接属性。
  - 对于 MySQL 的 C 客户端，不要使用“mysql_stmt_*”等函数来操作数据。
- 不支持 Set 语句（直接返回成功，但不做任何事情），因此不能设置用户级变量，也不能更改会话级的设置。因为后端的连接池机制需要保证连接不带状态属性。但”set autocommit = {0|1}”可以安心使用，不用担心 OneProxy 不支持。
- 不支持”USE”命令来切换当前数据库。从 OneProxy 角度来看，不同的集群相当于原先 MySQL 实例的数据库，“USE”命令也用来切换默认集群，而不是切换后端的默认数据库。
- 不支持分布式事务语及跨集群的 SQL 语句。虽然 OneProxy 后面能同时挂载多个集群，但其核心本质是一个功能较强、稳定和高效的流量路由软件，没有完整的 SQL 优化器和执行层，目前还不是一个真正的分布式数据库。

虽然 OneProxy 不支持一个 SQL 跨多个集群的操作，但仍可以用于垂直拆分，将无关联的表透明地移到其他集群，并且标记一下此表所在的集群名字，OneProxy 就可以实现灵活的表级路由控制，将后端多个集群虚拟化成一个大集群。可以通过如下选项：

- proxy-table-map.[1-255] = 表名:集群名字

分库分表情况下，也是如此，会自动维扩每个分表所在的集群名字，自动建立对应关系。



OneProxy 使用自己打点的方式来检测主从时延，在单主模式的流量策略下会自动创建心跳表（包含 Proxy UUID 和时间点两个字段，表名为：oneproxy_replication_timestamp），并以 100ms（可配置）的频率进行更新和查询，以计算主从时延。心跳表信息如下：

```
mysql> select * from oneproxy_replication_timestamp;
+--------------------------+------------------+
| proxy_uuid               | proxy_stamp      |
+--------------------------+------------------+
| ANGV-FXAA-GVJV-AAJQ-JVXA | 1648677892621686 |
+--------------------------+------------------+
1 row in set (0.00 sec)
```

通过以下两个选项来控制，更新和查询的频率：

- checkpoint-interval = 100 查询 Slave Binlog 时间点的频率
- checkpoint-confirm = 100 更新 Master Binlog 时间点的频率

采用自己打点的原因是“show slave status”里的 SBM 的信息不足够准确，并且只有秒级精度，而 OneProxy 希望能做到 100ms 级别的精度。当从节点的时延超过 60 秒（可配置）时，在读写分离时就不会向从节点分配流量。目前这个配置项是整个 OneProxy 实例级别的，没有实现集群级别的设置。控制项如下所示：

- binlog-maxdelay = 60 超过此值则自动踢除，低于此值则自动加回

由于 MySQL 上存在读写干扰情况，为了保证有一个节点能够及时跟上主库，可以设置节点为 HA 类型节点，这类节点在读写分离时不会承担读流量，以保证其恢复速度。可通过如下选项进行设置：

- proxy-haonly-addresses.[1-255] = 主机名:端口

这里需要你的 HA 机制确保主节点切换操作只在 HA 类型节点之间发生。除了主从时延检测之外，OneProxy 还实现了更强的一致性读写分离功能，当某个表被更新后，会记录表级别的复制位点，对此表的后续查询会进行从库复制位点比较，这样可以确保更新后的查询（和更新不在同一个事务内）可以被看到刚更新的数据，避免更新后看不到数据的现象。

所有的流量策略可以在管理端口执行“list policy”来获取，常见的有如下几类：

- master_only 读写主节点
- read_failover 读写主节点，主节点不可用则读从节点
- read_slave 写主节点，随机读 Slave 类型节点
- read_other 写主节点，随机读 Master 之外的节点
- read_balance 写主节点，随机读所有节点

OneProxy 丰富的流量策略和全局一致性功能，可以让你轻松应对各种业务场景。

大多数互联网应用能够默认接受读写分离，然而许多传统行业的应用很难透明地接受读写分离，OneProxy 曾为保险、电信、供应链、金融行业客户服务，基于 DML 更新时间点的一致性读写分离功能让应用接入更加透明。OneProxy 会记录每个表和当前会话的最后 DML 时间，在一致性读写分离路由逻辑中，会检查记录的 DML 时间，自动选择已经恢复到 DML 时间点之后的备节点提供服务，以免出现更新后读取不到的情形，提供等同单节点的体验。

OneProxy 还实现了基于表级别 DML 时间点的一致性查询缓存（Query Cache）的功能，可以透明地对 SQL 进行结果集缓存，以降低数据库的访问压力。只有事务外的语句能使用查询缓存，在事务中的语句则不走，以满足事务的强一致性要求。

一致性读写分离和查询缓存可能会降底流量分摊的效果，OneProxy 提供了 SQL 级 Hint 机制（有特定意义的注释）来灵活地管理流量路由，如下所示：

- select /* hint */ … from … where …

可以使用如下 Hint 进行灵活控制：

- master：指定在 Master 节点上执行，不进行读写分离路由。
- slave：指定在 Slave 节点上执行，不作读写分离的一致性检查，比如报表查询。
- delay=n：指定 SQL 可接受的复制时延，同样不作读写分离的一致性检查。
- cache[=n]：指定 SQL 可以使用查询缓存功能来加速。

通过 Hint 方式可能还需要更改应用 SQL，OneProxy 还提供后端命令由 DBA 同学来透明地控制 SQL 级的路由和缓存策略，以便在紧急情况下先解决问题。可以登录到 OneProxy 管理端口，列出执行过的 SQL 语句，如下所示：

```
mysql> list sqltext;
+------+------------+-------+-------+--------+-------+-------+--------+----------------------------------+
| SEQ  | HASHCODE   | CACHE | DELAY | MASTER | BLACK | AUDIT | TABCNT | SQLTEXT                          |
+------+------------+-------+-------+--------+-------+-------+--------+----------------------------------+
|    3 |  459744787 |     0 |     0 |      0 |     0 |     0 |      1 | SELECT c FROM sbtest1 WHERE id = ? |
|    1 | 3015039612 |     0 |     0 |      0 |     1 |     0 |      1 | select count(*) from t_test      |
|    2 |  743059242 |     0 |     0 |      0 |     0 |     0 |      1 | SELECT c FROM sbtest2 WHERE id = ? |
+------+------------+-------+-------+--------+-------+-------+--------+----------------------------------+
3 rows in set (0.00 sec)
```

其中 CACHE 表示缓存有效的时间，如果不为零则表示开启 SQL 级查询缓存功能；DELAY 表示允许的主从复制时延，对复杂的 SQL 语句；MASTER 表示 SQL 是否只从主库查询。可通过如下命令进行修改：

- set qcache hashcode intvalue
- set qdelay hashcode intvalue
- set qmaster hashcode {0|1}

OneProxy 为 DBA/SRC 提供更多的应急响应手段，比如透明开启高频 SQL 的缓存加速。

OneProxy 是带分库分表功能的透明代理，要实现分库分表就需要对现有的数据访问特征有较好的了解，比如哪些表或 SQL 语句常在事务中出现，哪些表通常会出现在同一个 SQL 中存在强关联关系。OneProxy 提供了多个维度的可观察性信息来帮助你快速了解你的应用，如下所示：

- 表粒度访问统计，记录每个表的增、删、改、查情况。

Table Name	SQLs	DMLs	Insert			Update			Delete			Select		
			Exec	Rows	Time	Exec	Rows	Time	Exec	Rows	Time	Exec	Rows	Time
sbtest1	1	0	0	0	0	0	0	0	0	0	0	851636	851636	87482
sbtest2	1	0	0	0	0	0	0	0	0	0	0	853517	853517	87677
t_test	1	0	0	0	0	0	0	0	0	0	0	1	1	17

- SQL 访问统计，按 SQL 分类汇总执行情况（执行次数、返回记录数、流量等）。

Hashcode	SQL Text	Tabs	Exec Total	Cache Hits	Exec (%)	Tran	Fail	Time Total	Time (%)	Avg Time	Max Time	Rows Total	Rows (%)	Avg Rows	Row Dist (1/2/3/4/5)	Bytes (KB)	Bytes (%)
743059242	SELECT c FROM sbtest2 WHERE id = ?	1	853517		50.06	0	0	87677.0	50.05	0.1	0.0	853517	50.06	1.00	0/0/0/0/0	160867	50.06
459744787	SELECT c FROM sbtest1 WHERE id = ?	1	851636		49.94	0	0	87482.6	49.94	0.1	0.0	851636	49.94	1.00	0/0/0/0/0	160513	49.94
3015039612	select count(*) from t_test	1	1		0.00	0	0	1.7	0.00	1.7	0.0	1	0.00	1.00	0/0/0/0/0	0	0.00

- 多表关联统计，按表的关联关系统计 SQL 的执行情况。

Tabes	Exec	Trans
trx: sbtest1, sbtest2	0	30

- 事务关联统计，按构成事务的 SQL 关联关系统计事务的执行情况。

SQL List	SQL Text	Exec	Roll Back	Call	DML Call	Time	Avg Time	Busy	DML Busy	Avg Busy	Rows	DML Rows	Avg Rows
459744787, 3598558246, 2774479865, 3488784266, 4142060818	SELECT c FROM sbtest1 WHERE id = ?; SELECT DISTINCT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c; SELECT SUM(k) FROM sbtest1 WHERE id BETWEEN ? AND ?; SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ?; SELECT c FROM sbtest2 WHERE id BETWEEN ? AND ? ORDER BY c;	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0	0	0.00
1265665596, 1317153989, 743059242, 4142060818, 654476956	SELECT DISTINCT c FROM sbtest2 WHERE id BETWEEN ? AND ? ORDER BY c; SELECT SUM(k) FROM sbtest2 WHERE id BETWEEN ? AND ?; SELECT c FROM sbtest2 WHERE id = ?; SELECT c FROM sbtest2 WHERE id BETWEEN ? AND ? ORDER BY c; SELECT c FROM sbtest2 WHERE id BETWEEN ? AND ?;	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0	0	0.00
4268898178, 2774479865, 3488784266, 743059242, 3598558246	SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c; SELECT SUM(k) FROM sbtest1 WHERE id BETWEEN ? AND ?; SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ?; SELECT c FROM sbtest2 WHERE id = ?; SELECT DISTINCT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c;	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0	0	0.00

这四类信息充分了解业务特征的重要信息，对于研究合理分库分表是非常必要的手段。回忆第一次在支付宝推动分库分表时，花了巨大的时间代价，写了不少代码去扫描 SQLMAP 文件，去分析表和表之间的强关联关系，通过 OneProxy 就简单多了。OneProxy 在多个级别上统计了这四类信息，如下表所示：

	表粒度统计	SQL 粒度统计	多表关联	事务关联
Proxy 级别	支持	支持	支持	支持
IP 级别	支持（*）	支持（*）	支持（*）	支持（*）
数据库	支持（*）	支持（*）	支持（*）	支持（*）

带星号的需要专业版本的 OneProxy 才能支持，基中 IP 级别也可以理解为应用级别的统计，可以让你快速了解某个应用是如何使用数据库的。想要查看这些信息，可以开启 OneProxy 内置的 HTTP 服务器，只需要配置一个 HTTP 服务器地址就行：

- proxy-httpserver = :8080

然后用浏览器访问指定的地址就可以了，默认用户名和密码为“admin”和”OneProxy”。

OneProxy 主要消耗 CPU 和网卡资源，除此之外 QPS 转发效率还取决于 event-threads 参数的设置，在一般的物理机上（如下测试机器为至少 3 年前淘汰下来的线下环境设备），其 SQL 转发性能（用 sysbench point select 小表纯内存进行测试）如下：

- event-threads = 1，最多使用 1 个 CPU 核，大约每秒 49000 次转发。
- event-threads = 2，最多使用 2 个 CPU 核，大约每秒 90000 次转发。
- event-threads = 4，最多使用 4 个 CPU 核，大约每秒 160000 次转发。
- event-threads = 8，最多使用 8 个 CPU 核，大约每秒 280000 次转发。
- event-threads = 16，最多使用 16 个 CPU 核，大约每秒 390000 次转发。

可见 OneProxy 的 SQL 转发效率还是很高的，性能是一般 Java 类 Proxy 的一倍以上。OneProxy 功能比较丰富，设计上采用多线程机制，内部由十几类不同线程紧密协作，来实现所有的相关功能。如下所示：

1. 主线程（oneproxy），监听器线程，用于处理工作端口和管理端口的新建连接。
2. 内存回收线程 (jemalloc_bg_thd)，静态编译 jemalloc, 开启了一个内存回收线程 。
3. Web 线程（http_server），提供简单的 SQL 性能统计信息查询页面服务。
4. 工作线程（worker_event_XX），可以有多个，主要用于 SQL 的前后端转发，可以确保简单 SQL 的转发效率。
5. 工作线程（worker_admin_01），只有一个，用于处理管理端口相关命令。
6. 工作线程（worker_query_XX），可以有多个，主要用于跨分片结果集聚合操作，以确保跨分片的复杂 SQL 操作不会影响普通 SQL 的转发。
7. 后端探活（ping_backend），检测后端状态，包括检测后端只读（read only）状态。
8. 本地探活（ping_local），用于维护后端的连接池，使之保留一定数量空闲连接数。
9. 远程探活（ping_remote），用于 OneProxy 集群多个节点之间的重要信息同步。
10. 性能监控（ping_osload），秒级采集 OneProxy 节点自身的资源利用率。
11. 性能统计（ping_realstats），计算两个时间点之间的 SQL 性能统计差值数据。
12. 异步日志（flush_errrlog），异步日志刷出线程，确保写日志不影响转发性能。
13. VIP 管理（ping_vip），管理 OneProxy 集群的 VIP 地址（专业版本特有）。

这十几类线程的高内聚设计，让 OneProxy 更易于维护和管理，无需借助第三方系统来支撑 OneProxy 运行。比如自带守护进程，当子进程因为程序缺限或内存原因 Crash、或被其他用户意外杀死时，守护进程会在几秒内迅速重起一个实例；并且自带集群功能，无须另外安装 Keepalived 等第三方辅助软件，以及编写相关脚本，仅需简单增加几个配置项。

OneProxy 支持分库分表功能，下面是一个 Hash 分表的用例，将"my_hash"表按 ID 列分成了 5 个表。在 OneProxy 中创建表时，将自动给每个分表加上 4 位数字的后缀。如下所示：

Hash 分区表配置	MySQL 后端真实表

```
{
        "table"      : "my_hash",
        "pkey"       : "id",
        "type"       : "int",
        "method"     : "hash",
        "partitions" : 5,
        "groups"     : [ "default" ]
},
```

```
mysql> show tables like 'my_hash_%';
+----------------------------+
| Tables_in_test (my_hash_%) |
+----------------------------+
| my_hash_0000               |
| my_hash_0001               |
| my_hash_0002               |
| my_hash_0003               |
| my_hash_0004               |
+----------------------------+
5 rows in set (0.00 sec)
```

支持一定能力的结果集合并操作，虽然功能上受限（一般 OLTP 系统上不会有复杂的汇总操作），但比一般的客户端 SDK 功能要强大。比如 Order By 操作的支持（同时支持升序和降序），避免了客户端的复杂设计：

```
mysql> select * from my_hash order by id;
+----+------+-------+
| id | col2 | col3  |
+----+------+-------+
|  1 |  100 | Row 1 |
|  2 |  101 | Row 2 |
|  3 |  102 | Row 3 |
|  4 |  100 | Row 4 |
|  5 |  101 | Row 5 |
|  6 |  102 | Row 6 |
|  7 |  100 | Row 7 |
|  8 |  101 | Row 8 |
|  9 |  102 | Row 9 |
| 10 |  102 |       |
+----+------+-------+
10 rows in set (0.00 sec)
```

```
mysql> select * from my_hash order by id desc;
+----+------+-------+
| id | col2 | col3  |
+----+------+-------+
| 10 |  102 |       |
|  9 |  102 | Row 9 |
|  8 |  101 | Row 8 |
|  7 |  100 | Row 7 |
|  6 |  102 | Row 6 |
|  5 |  101 | Row 5 |
|  4 |  100 | Row 4 |
|  3 |  102 | Row 3 |
|  2 |  101 | Row 2 |
|  1 |  100 | Row 1 |
+----+------+-------+
10 rows in set (0.00 sec)
```

支持同时带 Order By、Limit、Offset 的分页操作：

```
mysql> select * from my_hash
    ->     order by id limit 5 offset 1;
+----+------+-------+
| id | col2 | col3  |
+----+------+-------+
|  2 |  101 | Row 2 |
|  3 |  102 | Row 3 |
|  4 |  100 | Row 4 |
|  5 |  101 | Row 5 |
|  6 |  102 | Row 6 |
+----+------+-------+
5 rows in set (0.00 sec)
```

```
mysql> select * from my_hash
    ->     order by id limit 5 offset 2;
+----+------+-------+
| id | col2 | col3  |
+----+------+-------+
|  3 |  102 | Row 3 |
|  4 |  100 | Row 4 |
|  5 |  101 | Row 5 |
|  6 |  102 | Row 6 |
|  7 |  100 | Row 7 |
+----+------+-------+
5 rows in set (0.00 sec)
```

支持带 Group By 的 count、sum、max、min 汇总操作：

```
mysql> select col2, count(*), sum(id)
    ->     from my_hash group by col2;
+------+----------+---------+
| col2 | COUNT(*) | SUM(id) |
+------+----------+---------+
|  100 |        3 |      12 |
|  101 |        3 |      15 |
|  102 |        4 |      28 |
+------+----------+---------+
3 rows in set (0.01 sec)
```

```
mysql> select col2, min(col3), max(col3)
    ->     from my_hash group by col2;
+------+-----------+-----------+
| col2 | MIN(col3) | MAX(col3) |
+------+-----------+-----------+
|  100 | Row 1     | Row 7     |
|  101 | Row 2     | Row 8     |
|  102 |           | Row 9     |
+------+-----------+-----------+
3 rows in set (0.02 sec)
```

只要在 SELECT 关键字后面加上"/* parallel */"，就可以让上述语句并行执行以提升性能。

MySQL 8.0 已经将 Query Cache 相关代码移除，并不是说 Query Cache 机制不好，而是其实现不够优化。主要问题有三：第一是使用全局锁来保护，没有使用无锁编程，也有没有拆锁，甚至也没有使用读写锁；第二是没有使用版本号机制，DML 或 DDL 操作会主动去清理 Query Cache 中的失效缓存项，这一步会造成严重阻塞；第三是自定议了一个内存分配器，也是用全局锁保护，完全没有 jemalloc 好用。所以在多核的 CPU 上测试时，性能可能不升反降。OneProxy 重新设计了 Query Cache 机制，采用拆分后的读写锁来保证并发，并采用版本号机制来隔离读写干扰，可以让 Query Cache 重新飞起来。在测试环境中，单 Worker 配置下（即最多使用 1 个核）全缓存命中的 QPS 高达 10W+；而 8 个 Worker 配置下，全缓存命中的话，QPS 可高达 75W+，这是 MySQL 内置 Query Cache 功能不可能达到的。

对于缓存来讲，除缓存自身机制外，还有一件复杂的事情是，确定哪些 SQL 语句适合开启缓存，哪些 SQL 语句不适合开启缓存。如果错误地为更新频繁的表相关联的 SQL 开启缓存，可能会适得其反，而开启 OneProxy 的查询缓存，则非常简单。在 OneProxy 的可观测性数据中，可以很好地根据 SQL 对应表的查询次数、更新频率、SQL 重复执行率去评估一个是否适合开启缓存的分数，分数值越高则表示越适合查询缓存。如下图（QC 列）所示：

Hashcode	SQL Text	QC	Tabs	Exec Total
2867479701	SELECT c FROM sbtest1 WHERE id = ?	975	1	2239430
3767472819	SELECT SUM(k) FROM sbtest1 WHERE id BETWEEN ? AND ?	353	1	7502
2228742620	SELECT DISTINCT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c	290	1	7501
2312534216	SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ?	290	1	7502
3187428896	SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c	289	1	7502

图中数据是 sysbench 工具点查（point select）测试场景，可以测试不同大小结果集时的缓存分值，会发现结果集较小时，QC 分值因重复执行率较高而变高，逐渐加大结果集测试，QC 分值会因重复执行率变低而降底。并且 QC 分值的计算非常实时，结合 OneProxy 的自动查询缓存功能，可以有效地分担数据库的查询成本，降底公共云上数据库实例的资源成本。只需要配置如下参数：

- qcache-autotune：设置自动启用查询缓存的 QC 最小分值。

设置非 0 选项值后，只要 QC 分值高于设置值的，则会自动开启查询缓存，而低于的则会自动关闭，实现了查询缓存的自动和实时调优，避免 DBA 和应用研发人员逐个 SQL 进行低效高成本的调整，可极大地提升运维及研发效率。

并且在运行的过程中，还会根据缓存的命中率自动调整缓存时长，对于缓存命中率高的 SQL 会自动延长其在缓存有效期；而对于命中率低的，则会自动缩短缓存有效期，以节约内存给其他更适合的缓存项，并可以设置缓存项总数上限以及缓存总内存上限。