Mycat 功能解析

文档信息

文档编写人	HanSenJ	编写日期	2016-08-20
文档评审人		评审日期	

版本修订历史记录

版本号	作者	参与者	起止日期	备注
V1.0				

内容修订历史记录

章节	修改内容	修订人	修订日期	修订原因

目录

优势

1、 分布式数据库种类扩展

目前已经支持 Mysql、mongodb、oracle、sqlserver 、hive 、db2 、 postgresql 等主流数据。

2、 单点服务高可用性增强

Mycat 单点服务属分布式数据库无状态中间件，对于挂接数据库的高可用支持可以灵活的根据数据库本身的结构做配置支持。多模式的支持以及自动故障切换与恢复为本次高可用性增强的重要功能。

支持种类包括双多主多从模式、多主一从模式、双主双从模式、双主模式、主从模式、一主多从模式、单机版模式。

多模式基础上提供 3 中高可用方式：

不开启读写分离。所有读写操作都发送在写库上。

开启读写分离。

① 读写隔离模式：该种方式将读写完全隔离，写操作分发到写库，读操作分发到读库。

② 随机读模式：该种方式为增强功能，将写操作只分发到写库，读操作随机分发到所有读库与所有写库，分担读库压力，适用于写操作相对较少的情况。

③ 相对读模式：该种为增强功能呢，将写操作只分发到写库，读操作随机分发到读库与当前读库对应的写库上，只对当前读库对应的写库分担压力，适用于写操作频繁的情况。

以双主双从模式为例，双主为双写节点，双从为双读节点：

当双主节点主写发生了故障，服务会自动检测其健康状态，当服务发送三次心跳检测确认心跳失败后，会自动将将写操作切换到当前可用的从写节点上。同时记录服务日志，将发生的故障节点进行标识，以供故障恢复以及当前的健康状态检查使用。这样即做到了自动切换以保证使用方前端业务正常。

当主读节点发生故障，健康检查检测到主读状态，将读操作自动切换到从读或者从写上。

当从读节点发生故障，健康检查检测到从读状态，将读操作自动切换到主读或者主写上。

当双读节点发生故障，健康检查检测到双读状态，将读操作自动切换到主写或者从写上。

另一种情况，当主写节点发生故障后从写节点也相继发生了故障，或者主写及从写节点同时发生了故障的情况下，对于业务使用方应立即停止对其提供服务。此时无论读写都会反馈错误，毕竟写库都发生故障业务继续进行的意义已经不大，Mycat 不对外提供服务。

单点服务的高可用有健康检查及自动切换状态做了多模式的支持，但是对于自动恢复目前存在一定的缺陷。在下面的章节介绍。

3、 集群服务高可用性增强

集群服务高可用这里只描述 mycat 服务间高可用功能的增强，对于集群环境引入的三方架构如负载、健康检查、数据分发、选举机制等不包含在该范围。只针对 mycat 之前版本存在集群环境发生故障进而引发数据不一致的问题进行描述。

案例：

版本 cobar；

环境 三台 cobar 服务器；四台数据库服务器；

问题描述及影响：

双主双从模式下，三台 cobar 服务连接四台数据库服务。正常情况下 cobar 所有服务默认将写操作写入主写数据库，当主写数据库发生故障后，所有 cobar 服务会做心跳检测，检查到主写故障并将写操作自动切换到从写库，此时前端业务正常，数据库数据保持一致。

当主写库做故障修复后，重新将服务纳管人集群，纳管后主写库做为从写库保持正常的读写操作，此时前端业务依然正常，数据库仍可以保持一致。但在集群环境下，cobar 服务本身有一天发生了故障或者发生了需要重新启动服务的情况，重启任意一 cobar 服务，会导致该服务按照原有的主从配置进行读写，而忽略了数据库本身已经发生过了主从的写入故障切换。这样会造成重启的 cobar 服务与其余的 cobar 服务主从写入的不一致行，并且发生该问题没有任何系统性问题告警，以致在不同的写入模式下会并行运行而无法知会问题。进行造成主写库与从写库都写入数据导致数据不一致的问题，改问题随着运行时间的推移会越发严重化，数据难以恢复。

问题解决及功能提升：

该种情况下，mycat 在发生故障切换的时候记录了主从模式，故障恢复后重新启动故障服务的同时读取到历史记录的模式信息改变当前的主从状态与切换后的状态保持一致，从而保证服务的一致性与数据的一致性。

4、 异步非阻塞通讯方式的实现

Cobar 版本存在重要的隐患问题就是阻塞。先描述阻塞的业务场景，再介绍 mycat 解决方案。

阻塞场景，形象化描述为生产者与使用者场景。生产者提供产品供给使用者，消费者索要产品比生产者产出产品频繁量量也多，此时会产生请求积压，在阻塞中排长队的情况等待的情况。类似银行排队，在未办完业务前会一直处于排队等待状

态。同理在系统中，由于竞争资源处理排队等待的场景较多，阻塞模式不但影响了业务的时效，也没有必要的占用了服务资源。

解决方案为实现异步非阻塞的通讯方式，如在排好队列后不必一直等待，当知目前无法进行事项的办理，转而去做其他的事项，当可以办理该事项的时候通知再来办理，这样就避免了长时间排队等待的情况。

Mycat 将原前端后端同时存在的断开的两端线程池通过异步非阻塞方式贯通起来。解决由于数据库端连接通讯造成的阻塞问题，并且对连接通讯增加了监控告警功能。

5、 读写分离方式扩展及优化

读写分离在单点服务高可用中已经描述，以下正常环境的读写分离及故障情况下的读写分离介绍相关案例。

双主双从模式：M1 主写节点、M2 从写节点、S1 主读节点、S2 从读节点。

① 、正常环境

案例一、读写隔离模式：此时，所有操作会在 M1 上。

案例二、相对读模式：此时，写操作会在 M1 上，读操作会在 M2,S2,S1 上随机分配。观察日志以及查看数据库日志（数据库日志可以临时开启），可以发现这一点。

案例三：balance 设置为 2，此时，写操作在 M1 上，所有读操作会在 M1 ，M2，S1，S2 上随机分配。

② 、故障环境

案例一、相对读模式： M1 停止，此时，写操作会在 M2 上，读操作会在 S2 上

案例二、相对读模式: M1 停止 M2 停止，此时，写操作失败，读操作会在 S1,S2 上也会失败。

案例三、随机读模式： M1 停止，此时，写操作会在 M2 上，读操作会在 S2，M2 上

案例四、随机读模式： M1，M2 停止，此时，写操作失败，读操作会在 S1，S2 上也会失败。

案例五、相对读模式：S1 停止，此时，读操作会在 S2、M2 上 ，反之若停止 S2 ，则读操作在 S1、M1 上。

案例六、相对读模式： S1、S2 都停止，则读操作在 M2 上。

案例七、随机读模式： S1 停止，则读操作在 M2 上，M1、S2 上。

案例八、随机读模式： S1，S2 都停止，则读操作在 M2 上，M1 上。

6、 分片路由方式扩展

① 、增加 ER 分片方式

ER 分片根据数据表间的主从关系进行分片划分、从表的操作会以来主表的分片进行处理，从而保证了主从始终在同一分片，避免跨库关联操作。

② 、增加主键分片方式

对于无业务字段定义分片的表，需要根据主键做分片。通过主键做分片的优点在于缓存主键提升性能。

③ 、枚举值分片

通过枚举值方式进行分片，如省份编码、区域编码。

④ 、日期分片。

通过年月日数据字段分片。

⑤ 、冷热数据分片。

按照数据的使用频率情况分片。

7、 增加多平台部署方式

支持 linux 与 windows 两种平台部署方式。

8、 增加服务缓存机制

系统的缓存可以明显提升系统的性能，如本地缓存 oschache、ehcache，分布式缓存 memchached、redis。Mycat 自身增加了三种缓存方式，分别是路由缓存、主键节点缓存、关系缓存。

路由缓存记录路由计算结果。

表主键缓存为每个表主键建立缓存池。

关系缓存对于插入操作建立缓存。

通过以上三种数据放置于缓存中，不仅提升了性能同时避免了服务计算的开销。

9、 连接池管理功能增强

基于 Mycat 客户端连接池的故障检测机制，新增对连接池中失效连接资源的自动化策略管理，使其具备快速的连接恢复能力和僵尸连接回收能力。

新增对连接异常的管理能力，实现对频繁异常的连接进行自动断开处理，对长时间占用大量资源的连接进行超时释放，有效维护数据库连接资源。

10、增加数据库冗于管理

在分布式数据库架构下，由于一些模块部分表分片，部分表不分片（如参考表等），造成关联查询比较困难，Mycat 会面向应用开发者提供数据表冗余的规则配置能力，按照规则将指定数据表冗余到各个分片，并进行数据的同步管理，以降低分布式数据库下的部分关联查询场景的实现难度。

11、增加后台数据处理操作应用的开发支持功能

新增对逻辑库的元数据和表数据导入导出操作，包括导出表结构（包括分片结构），导入导出数据库数据（可以选择分片/所有分片）、导入导出数据表（可以选择分片/所有分片），支持标准 SQL 数据装载导出指令。

12、大数据操作压缩协议的支持

Mycat 增加压缩协议的支持，对于数据量较大返回大的结果集以及批量数据操作，可以明显的提升系统性能，同时减少服务资源的消耗，同时也可以节省网络流量。业务系统不可避免会产生使用大数据量的情况，使用压缩协议也是必要手段。

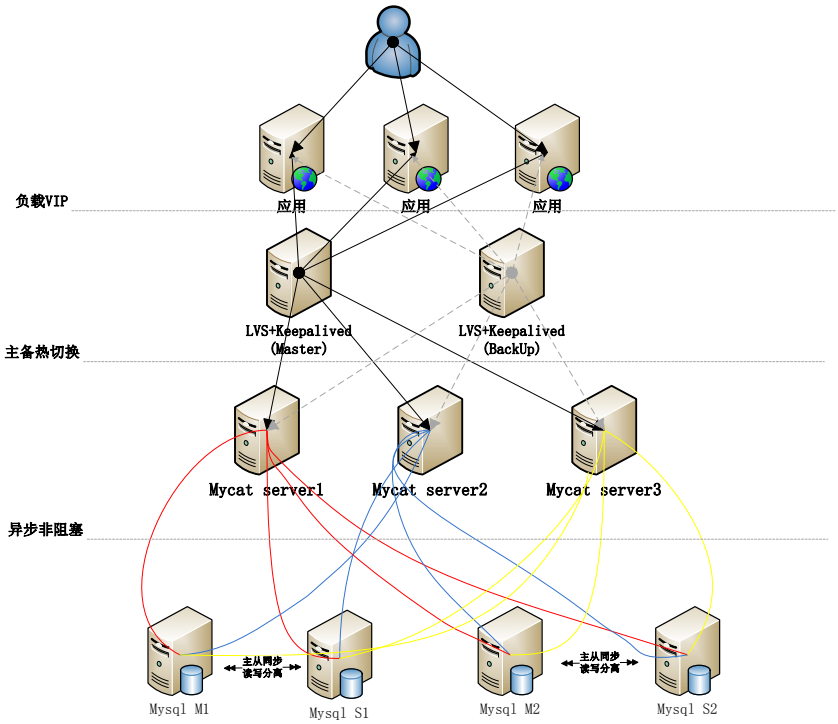
压缩协议的使用需要保证前端业务服务端、mycat 服务端以及数据库服务端三方开启压缩功能。大多主流数据库默认支持并开启，mycat 通过系统配置可开启关闭。

13、多租户应用支持

单租户模式下，需要给每个租户单独部署一整套服务及数据库环境，租户数量较大的情况下，增加管理及运维成本。Mycat 新增多租户应用支持功能，为不同租户共享一套服务环境。

14、增加集群环境高可用内置接口

集群环境的高可用指服务间的高可用，不包含单点服务高可用。集群环境高可用需要包含健康检查、主备热切换、故障恢复等功能。Mycat 提供 zookeeper 内置接口，通过 zookeeper 实现集群环境的高可用。由于 zookeeper 的分发、选举等核心功能在 mycat 服务间并没有真正的使用，而且造成了环境的臃肿，我们推荐使用了更为轻巧便捷的高可用方案。例如 lvs+keepalived 或 ha+keeplived 实现。



15、数据库语法的支持度提升及扩展

在原有基础上增加以下功能：

① 、支持存储过程

② 、支持自定义函数

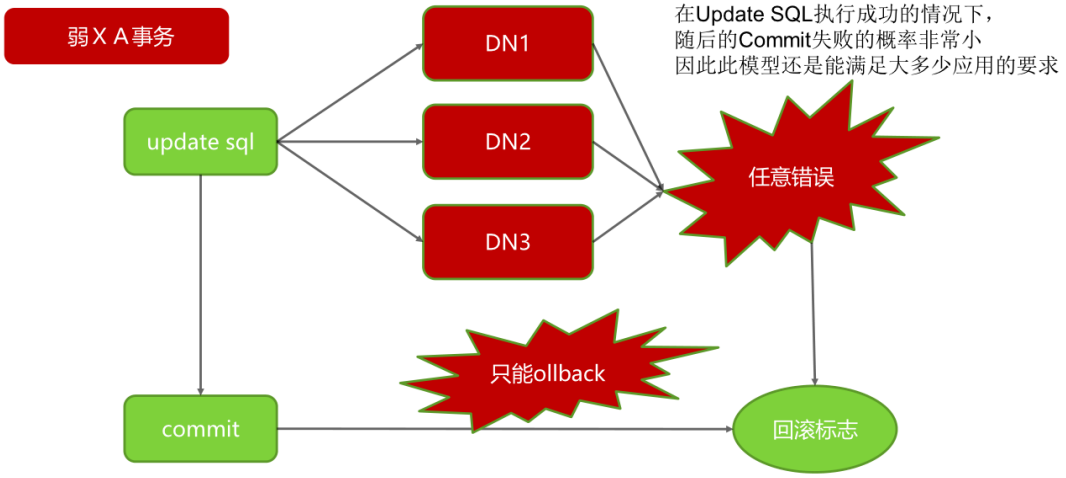
③ 、支持跨库关联

④ 、支持 SQL 批处理

⑤ 、支持跨库排序

缺陷

1、未完整支持分布式事务



随着并发量、数据量越来越大、业务已经细化到不能再按照业务划分的情况下不得不使用分布式数据库提高系统性能，在分布式系统中各个节点在物理上都是相对独立的，独立每一台节点数据操作都可以满足 ACID。但是，各独立节点之间无法知道其他节点事务的执行情况，如果要想让分多台机器中数据保存一致，就必须保证所有节点上面的数据操作要么全部执行成功，要么全部不执行，比较常规的解决方法是引入“协调者”来统一调度所有节点执行。

Mycat 可以通过用户会话设置是否启动事务，通过设置服务连接中变量配置参数来控制是否事务异常需要回滚。在 mycat 中的事务是一种二阶段提交事务方式，但是从实际应用场景出发这种出现故障的概率还是比较小的，因此这种实现方式可以满足很多应用需求。

但是二阶段提交事务的方式存在以下缺陷：

① 、同步阻塞问题，执行过程中所有参与节点都是事务阻塞型的，当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态。

② 、单点故障，由于协调者的重要性一旦协调者发生故障参与者会一直阻塞下去。

③ 、数据不一致，这也是最重要问题。在二阶段提交的阶段二中，当协调者向参与者发送 commit 请求之后，发生了局部网络异常或者在发送 commit 请求过程中协调者发生了故障，这回导致只有一部分参与者接受到了 commit 请求，而在这部分参与者接到 commit 请求之后就会执行 commit 操作，但是其他部分未接到 commit 请求的机器则无法执行事务提交，于是整个分布式系统便出现了数据部一致性的现象。

3、单点服务高可用存在故障恢复缺陷

单点服务故障不再累述，对于单点服务故障目前 mycat 根据记录的历史主从状态标识进行自动恢复。该操作只能保证 mycat 单点服务以及集群服务间主从状态一致性，无法干预数据库本身的主从同步机制，进而服务与数据库间主从状态的不一致，虽不会产生主从写入

同时写入造成数据不一致的问题，但是存在不一致性。解决该问题目前只能进行人工干预，停止某数据库服务进行手工改写，改写后重新纳入。不但没有实现自动恢复，同时也没有预留人工处理的时间，保持着一种主从双向错误并行但暂不产生问题的状态。

建议解决方案进行对数据库主从状态的检查，并以数据库的主从状态为基准切换 mycat 同步状态。

4、数据库语法部分尚未完善

① 、跨节点的联合查询有条件支持，三表及以上跨节点关联不支持。
举例：select * from a join b on a.id =b.id join c on b.id=c.id join d on c.id=d.id;

② 、视图功能不支持
举例：select * from v_viewtablea(数据库视图)

③ 、跨 schema 间数据库操作不支持
举例：select * from a(逻辑库).table a,b(逻辑库).table where a.id = b.id

④ 、插入的字段不包含分片字段(全逻辑库入库方式)不支持
举例：insert into a(column1,column2)values(values1,values2)(未包含分片)

⑤ 、复制插入不支持，但使用注解方式支持。
举例：insert into table select * from table2。支持方式：/*!mycat: sql=select 1 from test */insert into t_user(id,name) select id,name from t_user2;

⑥ 、多行插入不支持
举例： insert into tab(a1,a2) values(b1,b2),(c1,c2)…

⑦ 、不允许更新分片表中分片字段
举例：update table set sharding_id(分片)=,cloumn1=,cloum2= where …

⑧ 、多表更新不支持
举例：update a, b set a.column1=values1, b. column1=values2 where a.id=b.id

⑨ 、复杂更新不支持
举例：update a, b set a. column1= values1 where a.id=b.id;

⑩ 、复杂删除不支持
举例：delete a from a join b on a.id=b.id;

⑪ 、子查询方式删除支持，但不能引用别名
举例：delete from a where a.id in (select id from b)

⑫ 未支持存储过程定义，因而不允许调用存储过程，可通过注解来调用各个分片上的存储过程
举例：/*!mycat: sql=select 1 from test */ CREATE PROCEDURE `test_proc`() BEGIN END ;

⑬ 未支持直接调用自定义函数，但支持固定语法调用。
举例： select id, func() from table 需要分片 table 所在的所有分片上存在这个函数。

⑭ 排序局限性。
举例：Order by 字段必须出现在 select 中（先将结果取出，然后排序）

⑮ 复杂子查询存在结果集错误情况。
举例：外层查询没有分片查询条件， 则会在所有分片上执行。

⑯ 多分片执行服务自带函数 AVG 结果集错误。
举例：多分片执行的 AVG 函数情况。

5、尚未支持组合分片与二次分片

组合分片为分片库与分片表固定不变，分片表分片字段由多字段方式组成。这种情况用于一些特殊业务场景，由组合分片中单字段分片路由或组合分片组合字段路由，在保证分片库结构不变得情况下，路由到不同的分片库上。

二次分片为分片表不发生变化的情况下，分片库基于一次分片的基础上进行二次拆分。如某个分片存在单表数据量过于庞大的情况在一次分片下仍然不能完全解决大数据问题而需要进行的二次拆分，以提高性能。例如按省份一次分片后，经过生产常年运行积累，某一业务大省单表业务数据量仍然大于数亿。此时已经不能对前期做的一次分片整体推翻重新分片，需要对过大的分片库进行二次拆分，如将业务大省再按照地市做二次分片。

目前尚未支持这两种方式。

6、数据表级的访问授权功能暂未支持

目前支持逻辑库级的用户权限访问，包括只读、读写，但是权限控制粒度细化到数据表，可以在用户访问同一逻辑库时限制他访问的具体数据表、以及操作类型目前尚未支持。

7、跨库关联、跨库排序分页性能较差

跨库关联目前支持到两个及以下的表进行跨库关联操作，但是性能较差，存在大量的跨库逻辑计算。同样，跨库排序分页采用服务内存排序数据整合方式，不仅消耗服务器资源，同时性能也不理想。