

mysql常见问题分析

MySQL版本类问题

- 你之前工作中使用的是什么版本的MySQL？为什么选择这个版本？（为什么选择某一版本的MySQL）

知识点：

1、MySQL常见的发行版

MySQL的官方版本、Percona MySQL、MariaDB

2、各个版本之间的区别及优缺点

	MySQL	Percona MySQL	MariaDB
服务器特性	开源	开源	开源
	支持分区表	支持分区表	支持分区表
	InnoDB	XtraDB	XtraDB
	企业版监控工具 社区版不提供	Percon Monitor 工具	Monyog

	MySQL	Percona MySQL	MariaDB
高可用特性	基于日志点复制	基于日志点复制	基于日志点复制
	基于Gtid复制	基于Gtid复制	基于gtid复制，但gtid同MySQL不兼容
	MGR	MGR & PXC	Galera Cluster
	MySQL Router	Proxy SQL	MaxScale

	MySQL	Percona MySQL	MariaDB
安全特性	企业版防火墙	ProxySQL FireWall	MaxScale FireWall
	企业版用户审计	审计日志	审计日志
	用户密码生命周期	用户密码生命周期	-
	sha256_password caching_sha2_password	sha256_password caching_sha2_password	ed25519 sha256_password

	MySQL	Percona MySQL	MariaDB
开发及管理	窗口函数(8.0)	窗口函数(8.0)	窗口函数(10.2)
	-	-	支持基于日志回滚
	-	-	支持记在表中记录修改
	Super read_only	Super read_only	-

- 如何决定是否要对MySQL进行升级？如何进行升级？

升级前要考虑的问题：

1、升级可以给业务带来的益处：

是否可以解决业务上某一方面的痛点；

是否可以解决运维上某一方面的痛点；

2、升级可能对业务带来的影响：

对原业务程序的支持是否影响（数据库连接driver、sql_mode）；

对原业务程序的性能是否有影响；

3、数据库升级方案的制定：

评估受影响的业务系统；

升级的详细步骤（需在测试环境中多次演练）：

1、对待升级数据库进行备份

2、升级Slave服务器版本

3、手动进行主从切换

4、升级MASTER服务器版本

5、升级完成后进业务行检查

升级后的数据库环境检查；

升级后的业务检查；

4、升级失败的回滚方案；

升级失败回滚的步骤；

回滚后的数据库环境检查

回滚后业务检查

- 最新的MySQL版本是什么？它有什么特性比较吸引你？

MySQL8.0版本主要的新特性

一、服务器功能

- 所有元数据使用InnoDB引擎存储，无frm文件
- 系统表采用InnoDB存储并采用独立表空间
- 支持定义资源管理组（目前仅支持CPU资源）
- 支持不可见索引和降序索引，支持直方图优化

5、支持窗口函数

6、支持在线修改全局参数持久化

二、用户及安全

1、默认使用caching_sha2_password认证插件

2、新增支持定义角色 (role)

3、新增密码历史记录功能，限制重复使用密码

三、InnoDB功能

1、InnoDB DDL语句支持原子操作

2、支持在线修改UNDO表空间

3、新增管理视图用于监控INNODB表状态

4、新增innodb_dedicated_sever配置项

用户管理类问题

- 如何在给定场景下为某用户授权？

知识点：

如何定义MySQL数据库账号？

用户名@可访问控制列表

使用CREATE USER命令建立用户

MySQL常用的用户权限

	语句	说明
Admin	Create User	建立新的用户的权限
	Grant option	为其他用户授权的权限
	Super	管理服务器的权限
DDL	Create	新建数据库，表的权限
	Alter	修改表结构的权限
	Drop	删除数据库和表的权限
	Index	建立和删除索引的权限

	语句	说明
DML	Select	查询表中数据的权限
	Insert	向表中插入数据的权限
	Update	更新表中数据的权限
	Delete	删除表中数据的权限
	Execute	执行存储过程的权限

```
#获取当前MySQL版本支持的用户权限命令
show privileges;
```

如何为用户授权?

遵循最小权限原则

使用grant 命令对用户进行授权

```
#授权
grant select,insert,update,delete on db.tb to user@ip;
#收回权限
revoke delete on db.tb to user@ip;
```

- 如何保证数据库账号的安全?

知识点:

1、数据库用户管理流程规范 (防止人为泄露)

最小权限原则;

密码强度策略;

密码过期原则; (程序必须支持密码过期后从配置中心自动获取新的密码)

限制历史密码重用原则; (MySQL8.0新增)

2、密码管理策略 (增加暴力破解难度)

- 如何从一个实例迁移数据库账号到另一个实例?

解决思路:



导出用户建立及授权语句

```
pt-show-grants u=root,p=123456,h=localhost  
#还可以使用mysqlump工具导出MySQL用户信息
```

服务器配置类问题

- 请分析一个group by语句的异常原因

知识点: sql_mode的作用

配置MySQL处理SQL的方式

```
#persist, MySQL8.0新增重启不会失效  
set [session/global/persist] sql_mode='xxxxx'  
#在配置文件中配置, sql_mode=xxxxxxxx
```

常用的sql_mode:

1. ONLY_FULL_GROUP_BY:

对于GROUP BY聚合操作,如果在SELECT中的列,没有在GROUP BY中出现,那么这个SQL是不合法的,因为列不在GROUP BY从句中

2. NO_AUTO_VALUE_ON_ZERO:

该值影响自增长列的插入。默认设置下,插入0或NULL代表生成下一个自增长值。如果用户希望插入的值为0,而该列又是自增长的,那么这个选项就有用了。

3. STRICT_TRANS_TABLES:

在该模式下,如果一个值不能插入到一个事务表中,则中断当前的操作,对非事务表不做限制

4. NO_ZERO_IN_DATE:

在严格模式下,不允许日期和月份为零

5. NO_ZERO_DATE:

设置该值,mysql数据库不允许插入零日期,插入零日期会抛出错误而不是警告。

6. ERROR_FOR_DIVISION_BY_ZERO

在INSERT或UPDATE过程中,如果数据被零除,则产生错误而非警告。如果未给出该模式,那么数据被零除时MySQL返回NULL

7. NO_AUTO_CREATE_USER:

禁止GRANT创建密码为空的用户

8. NO_ENGINE_SUBSTITUTION:

如果需要的存储引擎被禁用或未编译,那么抛出错误。不设置此值时,用默认的存储引擎替代,并抛出一个异常

9. PIPES_AS_CONCAT:

将"||"视为字符串的连接操作符而非或运算符,这和Oracle数据库是一样的,也和字符串的拼接函数Concat相类似

10. ANSI_QUOTES:

启用ANSI_QUOTES后,不能用双引号来引用字符串,因为它被解释为识别符

- 如何比较系统运行配置和配置文件中的配置是否一致?

知识点:

- 使用set命令配置动态参数
- 使用pt-config-diff工具比较配置文件

```
pt-config-diff u=root,p=123456,h=localhost /etc/my.cnf
```

- 举几个MySQL中的关键性能参数

知识点:

常用的性能参数:

	参数	说明
服务器配置参数	max_connections	设置MySQL允许访问的最大连接数量
	interactive_timeout	设置交互连接的timeout时间
	wait_timeout	设置非交互连接的timeout时间
	max_allowed_packet	控制MySQL可以接收的数据包的大小
	sync_binlog	表示每写多少次缓冲会向磁盘同步一次binlog

	参数	说明
服务器配置参数	sort_buffer_size	设置每个会话使用的排序缓存区的大小
	join_buffer_size	设置每个会话所使用的连接缓冲的大小
	read_buffer_size	指定了当对一个MYISAM进行表扫描时所分配的读缓存池的大小
	read_rnd_buffer_size	设置控制索引缓冲区的大小
	binlog_cache_size	设置每个会话用于缓存未提交的事务缓存大小

	参数	说明
存储引擎参数	innodb_flush_log_at_trx_commit	0:每秒进行一次重做日志的磁盘刷新操作。 1:每次事务提交都会刷新事务日志到磁盘中。 2:每次事务提交写入系统缓存每秒向磁盘刷新一次
	innodb_buffer_pool_size	设置Innodb缓冲池的大小,应为系统可用内存的75%。
	innodb_buffer_pool_instances	Innodb缓冲池的实例个数,每个实例的大小为总缓冲池大小/实例个数。
	innodb_file_per_table	设置每个表独立使用一个表空间文件

日志类问题

- 常用的MySQL日志有哪些？我们在什么情况下使用这些日志？

知识点：

MySQL常用的日志类型

- 错误日志 (error_log)：记录MySQL在启动、运行或停止时出现的问题；
- 常规日志 (general_log)：记录所有发向MySQL的请求；
- 慢查询日志 (slow_query_log)：记录符合条件的查询；
- 二进制日志 (binary_log)：记录全部有效的数据修改日志；
- 中继日志 (relay_log)：用于主从复制，临时存储从主库同步的二进制日志。

各种日志的配置和使用场景

- 错误日志 (error_log)：

使用场景：

分析排除MySQL运行错误；

记录未经授权的访问；

配置：

log_error=\$mysql/sql_log/mysql-error.log

log_error_verbosity=日志错误级别[1,2,3]

log_error_services=[日志服务组件；日志服务组件]，MySQL8.0新增

- 常规日志 (general_log)：

使用场景：

分析客户端发送到MySQL的实际请求；

配置：

general_log=[ON | OFF]

general_log_file=\$mysql/sql_log/general.log

log_output=[FILE | TABLE | NONE]

- 慢查询日志 (slow_query_log)：

使用场景：

将执行成功并符合条件的查询记录到日志中；

找到需要优化的SQL；

配置：

slow_query_log=[ON | OFF]

slow_query_log_file=\$mysql/sql_log/slowlog.log

long_query_time=xx秒

log_queries_not_using_indexes=[ON | OFF]

log_slow_admin_statements=[ON | OFF]

log_slow_slave_statements=[ON | OFF]

4、二进制日志 (binary_log) :

使用场景：

记录所有对数据库中数据的修改；

基于时间点的备份和恢复；

主从复制

配置：

log-bin=[base_name]

binlog_format=[ROW | STATEMENT | MIXED]

binlog_row_image=[FULL | MINIMAL | NOBLOB]

binlog_row_query_log_events=[ON | OFF]

log_slave_updates=[ON | OFF]

sync_binlog=[1 | 0]

expire_log_days=days

```
#命令删除binlog日志
purge binary logs to 'mysql-bin.010';
purge binary logs before '2020-08-21 17:28:46';
```

5、中继日志 (relay_log) :

使用场景：

临时记录从主服务器同步的二进制日志

配置：

relay_log=filename

relay_log_purge=[ON | OFF]

- 如何通过日志来审计用户活动？

存储引擎相关问题

说说你了解的MySQL存储引擎及其适用场景

- MYISAM： MySQL5.6之前的默认引擎，最常用的非事务型存储引擎；

特点：

非事务型存储引擎

数据以堆表方式存储

使用表级锁

支持Btree索引，空间索引，全文索引

使用场景：

读操作远远大于写操作的场景

不需要使用事务的场景

- CSV：数据以CSV格式存储的非事务型存储引擎；

特点：

非事务型存储引擎

数据以CSV格式存储

所有列都不能为NULL

不支持索引

使用场景：

做为数据交换的中间表使用

- Archive：只允许查询和新增数据而不允许修改的非事务型存储引擎

特点：

非事务型存储引擎

表数据使用 zlib 压缩

只支持 insert 和 select 操作

只允许在自增ID上建立索引

使用场景：

日志和数据采集类应用

数据归档存储

- Memory：数据存放在内存中，是一种易失性非事务存储引擎

特点：

非事务型存储引擎

数据存放在内存中

所有字段长度固定

支持Btree和Hash索引

使用场景：

用于缓存字典映射表

缓存周期性分析数据

- INNODB：常用的事務型存储引擎

特点：

事务型存储引擎支持ACID（原子性、一致性、隔离性、持久性）

数据按主键聚集存储

支持行级锁及MVCC（多版本并发控制）

支持Btree和自适应Hash索引

支持全文和空间索引（5.6开始支持全文索引，5.7开始支持空间索引）

使用场景：

大多数OLTP场景（联机事务处理）

- NDB：MySQL集群所使用的内存型事务型存储引擎

特点：

事务存储引擎

数据存储在内存中（启动时会把数据从磁盘全部读入内存）

支持行级锁

支持高可用集群

支持Ttree索引

使用场景：

需要数据完全同步的高可用场景

在什么情况下Innodb无法在线修改表结构？

Innodb不支持在线修改表结构的场景：

加全文索引：

```
create fulltext index name on table(column)
```

加空间索引：

```
alter table geom add spatial index(g)
```

删除主键：

```
alter table tbl_name drop primary key
```

增加自增列：

```
alter table tbl_name add column id int auto_increment not null primary key
```

修改列类型：

```
alter table tbl_name change c1 c1 new_type
```

修改表字符集：

```
alter table tbl_name character set = charset_name
```

在线DDL存在的问题：

1. 有部分语句不支持在线DDL
2. 长时间的DDL操作会引起严重的主从延迟
3. 无法对DDL操作进行资源限制

在无法进行在线修改表结构的情况下，要如何操作？（如何更安全的在线修改表结构）

pt-online-schema-change [OPTIONS] DSN

```
pt-online-schema-change --alter='add column modified_time timestamp' --execute -D=stock,t=stock,u=root,p=123456
```

InnoDB是如何实现事务的?

知识点:

1、什么是事务?

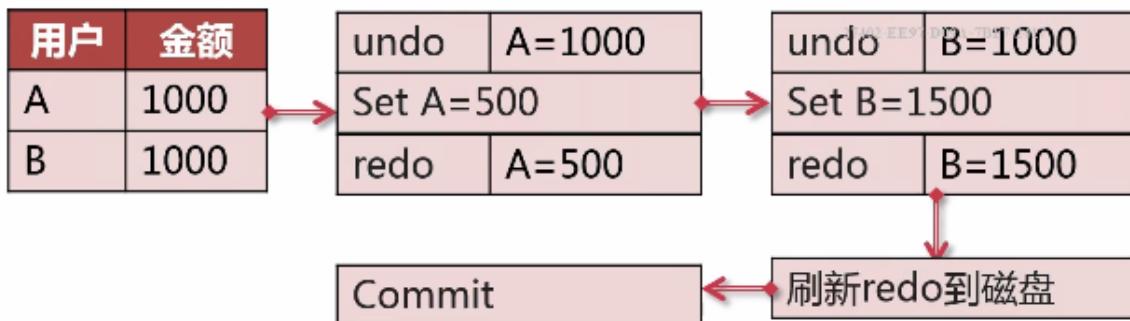
- 原子性 (A) : 一个事务中的所有操作, 要么全部完成, 要么全部不完成, 不会结束在中间某个环节;
- 一致性 (C) : 在事务开始之前和事务结束以后, 数据库的完整性没有被破坏;
- 隔离性 (I) : 要求每个读写事务的对象与其他事务的操作对象能相互分离, 即该事务提交前对其他事务都不可见;
- 持久性 (D) : 事务一旦提交了, 其结果是永久性的, 就算发生了宕机等事故, 数据库也能将数据恢复。

2、事务的实现方式

- 原子性: 回滚日志 (undo log), 用于记录数据修改前的状态;
- 一致性: 重作日志 (redo log), 用于记录数据修改后的状态;
- 隔离性: 锁, 用于资源隔离, 分为共享锁和排它锁;
- 持久性: 重作日志+回滚日志

示例:

A账户向B账户汇500元



Innodb读操作是否会阻塞写操作?

知识点:

查询需要对资源加共享锁

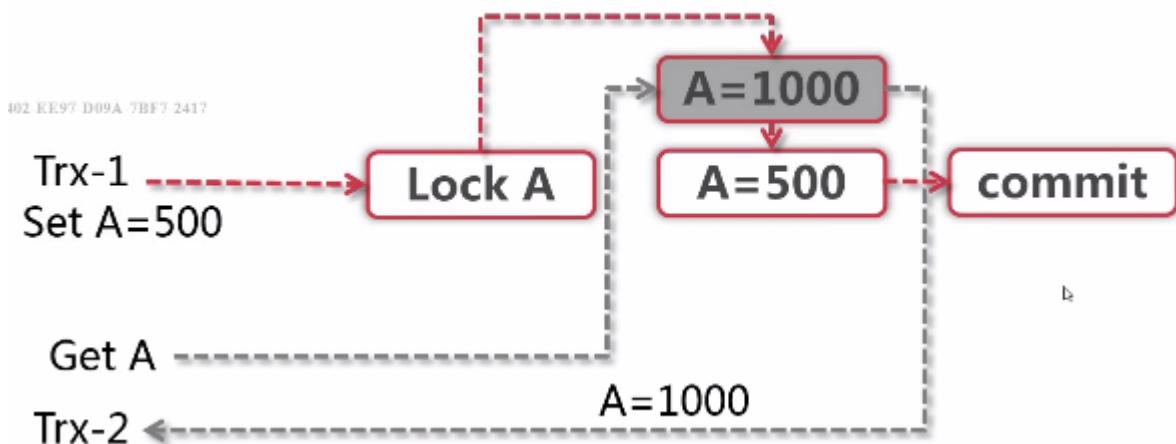
数据修改需要对资源加排它锁

innodb MVCC (多版本并发控制) 的实现方式 (使用undo log, redo log)

(事务隔离级别: 读未提交, 读已提交, 可重复读, 串行化)

示例:

MVCC(多版本并发控制)



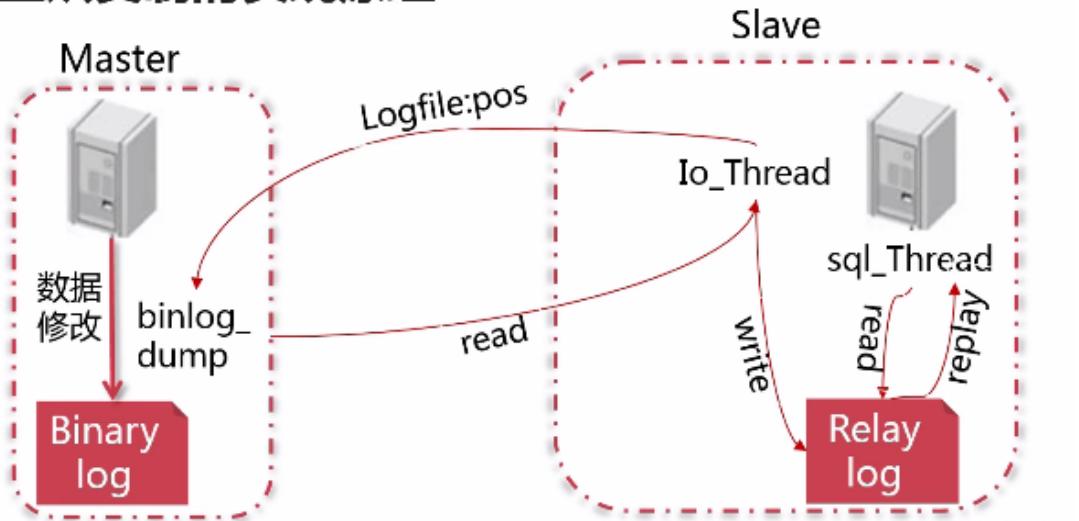
MySQL高可用架构类问题

- MySQL的主从复制是如何工作的?

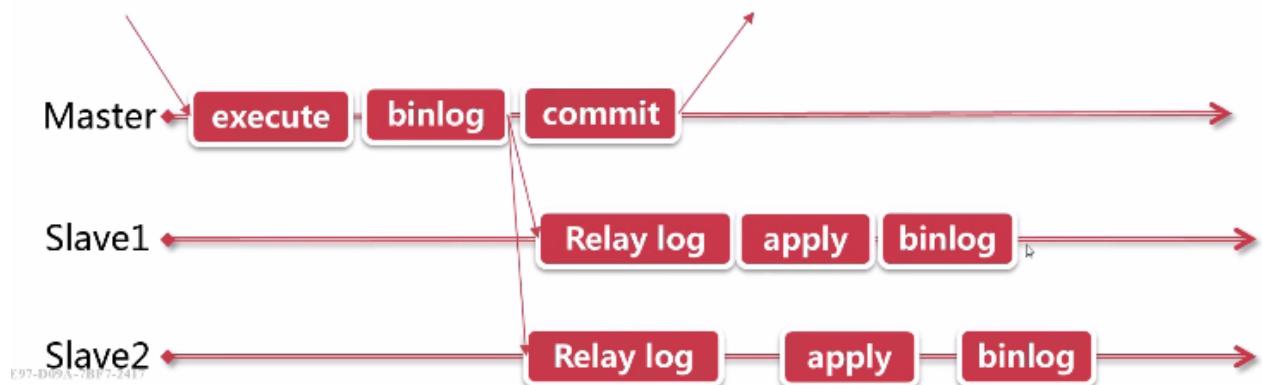
知识点：

MySQL主从复制的实现原理

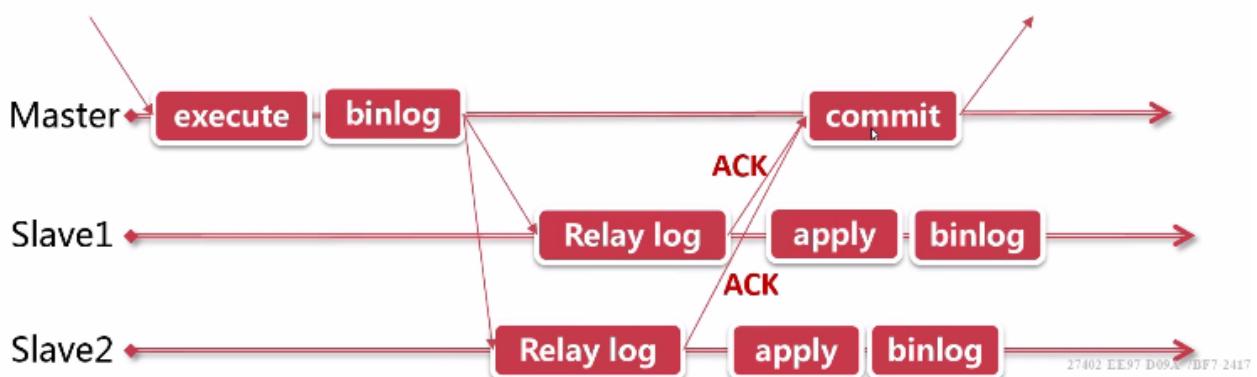
MySQL主从复制的实现原理



MySQL主从复制的实现原理—异步复制



MySQL主从复制的实现原理—半同步复制



2、MySQL主从复制的配置步骤

在master服务器上的操作：

开启binlog (必须) 开启gtid (可选)

建立同步所用的数据库账号

使用master-data参数备份数据库

把备份文件传输到slave服务器

在slave服务器上的操作：

开启binlog (可选) 开启gtid (可选)

恢复master上备份的数据库

使用change master配置链路

使用start slave启动复制

- 比较一下基于GTID方式的复制和基于日志点的复制。

知识点：

基于日志点的复制：

传统的主从复制方式；

slave请求master的增量日志依赖与日志偏移量

配置链路时需指定master_log_file和master_log_pos参数

基于GTID的复制：

GTID=source_id:transaction_id (全局事务ID)

slave增量同步Master的数据依赖于其未同步的事务ID

配置复制链路时， slave可以根据已经同步的事务ID继续自动同步

基于日志的复制	基于GTID的复制
兼容性好	同老版本MySQL及MariaDB不兼容
支持MMM和MHA架构	仅支持MHA架构
主备切换后很难找到新的同步点	基于事务ID复制，可以很方便的找到未完成同步的事务ID
可以方便的跳过复制错误	只能通过置入空事务的方式跳过错误

两种复制方式如何选择：

需要兼容老版本MySQL和MariaDB ----基于日志点的复制

需要使用MMM架构-----基于日志点的复制

其他各种情况---优先选择基于GTID的复制

- 比较一下MMM和MHA两种高可用架构的优缺点。

知识点：

1、 MMM和MHA两种架构的作用；

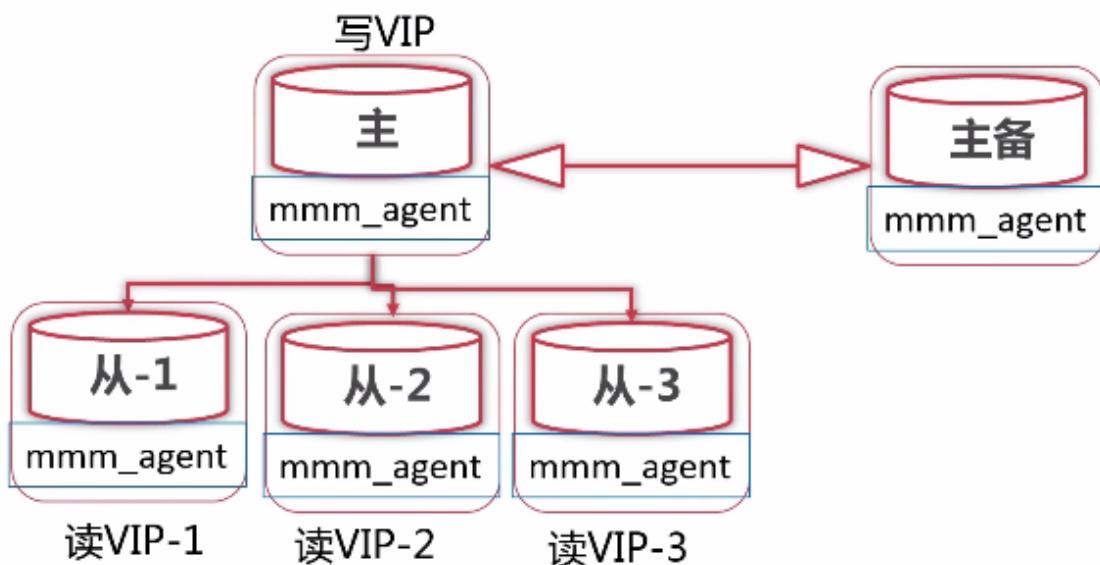
对主从复制集群中的master的健康进行监控

当master宕机后把VIP迁移到新的master

重新配置集群中的其他slave对新的master同步

2、 MMM架构的优缺点及适用场景； (master master mysql)

MMM适用的主从复制架构



MMM架构的需要的资源

资源	数量	说明
主DB	2	用于主备模式的主主复制配置
从DB	0-N	可以配置0台或多台从服务器
IP地址	2n+1	N为MySQL服务器的数量
监控用户	1	用于监控数据库状态的MySQL用户(replication client)
代理用户	1	用于MMM的agent端用于改变read_only状态 (super, replication client, process)
复制用户	1	用于配置MySQL复制的MySQL用户(replication slave)

故障转移步骤：

在slave服务器上的操作：

- 1、完成原主上已复制日志的恢复；
- 2、使用change master命令配置新主

在主备服务器上的操作：

- 1、设置read_only=off
- 2、迁移写VIP到新的主服务器

优点：

提供了读写VIP的配置，使读写请求都可以达到高可用；

工具包相对完善，不需要额外开发脚本；

完成故障转移后，可以持续对MySQL集群进行高可用监控

缺点：

故障切换简单粗暴易丢失事务（解决方案：主备使用5.7以后的本同步复制）

不支持GTID的复制方式 (解决方案：自行修改perl脚本实现)

社区不活跃，很久未更新版本

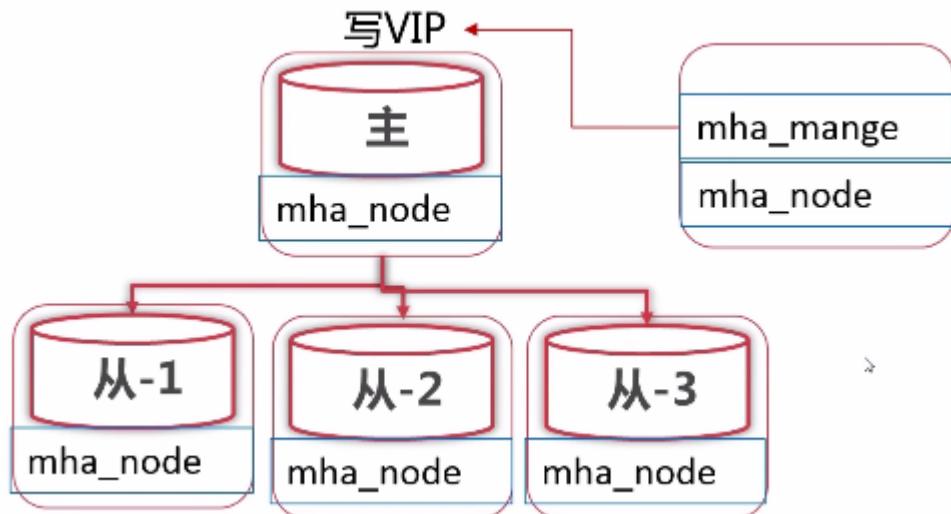
适用场景：

使用基于日志点的主从复制方式

使用住住复制的架构

需要考虑读高可用的场景

3、MHA架构的优缺点及适用场景；



MHA架构的需要的资源

资源	数量	说明
主DB	1	用于初始主从复制的Master服务器
从DB	2到N	可以配置2台或多台从服务器
IP地址	n+2	N为MySQL服务器的数量
监控用户	1	用于监控数据库状态的MySQL用户(all privileges)
复制用户	1	用于配置MySQL复制的MySQL用户(replication slave)

故障转移步骤：

- 1、选举具有最新更新的slave
- 2、尝试从宕机的master保存二进制日志
- 3、应用差异的中继日志到其他slave
- 4、应用从master保存的二进制日志；
- 5、提升选举的slave为新的master；
- 6、配置其他slave向新的master同步；

优点：

支持GTID的复制方式和基于日志点的复制方式

可从多个slave中选举最合适的新master

会尝试从旧master中尽可能多的保存未同步日志

缺点：

未必能获得到旧主未同步的日志（解决方案：主备使用5.7以后的半同步复制）

需要自行开发写VIP转移脚本

只监控master而没有对slave实现高可用的办法

适用场景：

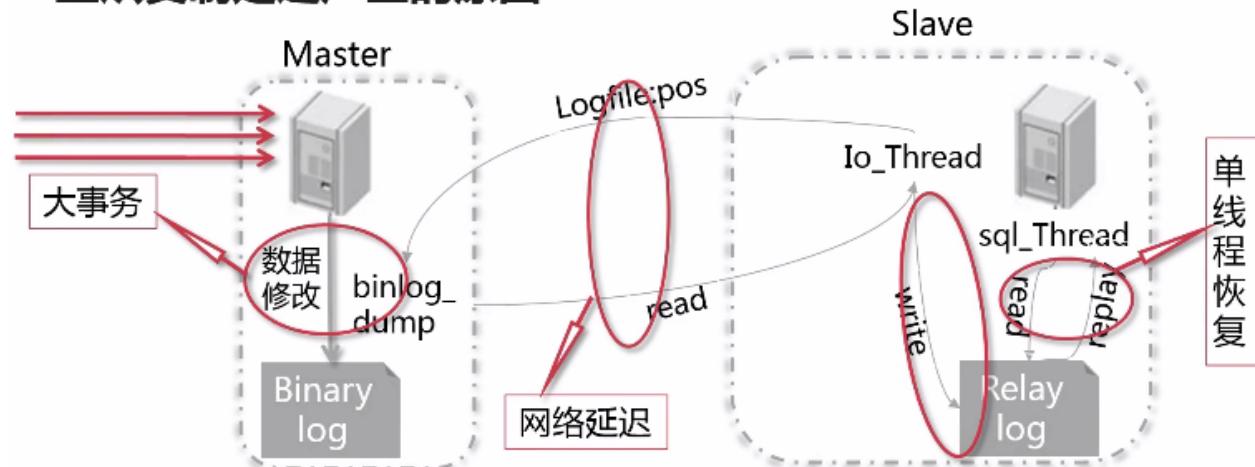
使用基于GTID的复制方式

使用一主多从的复制架构

希望更少的数据丢失场景

- 如何减小主从复制的延迟。

主从复制延迟产生的原因



1、大事务：数万行的数据更新或对大表的DDL操作

解决思路：

化大事务为小事务，分批更新数据

使用pt-online-schema-change工具进行DDL操作

2、网络延迟：

解决思路：

减小单次事务处理的数据量以减小产生的日志文件大小

减少master上所同步的slave的数量（一般不超过5个）

3、由master上多线程的写入，slave上单线程恢复引起的延迟

解决思路：

使用MySQL5.7之后的多线程复制

使用MGR复制架构

- 说说你对MGR的认识。

知识点：

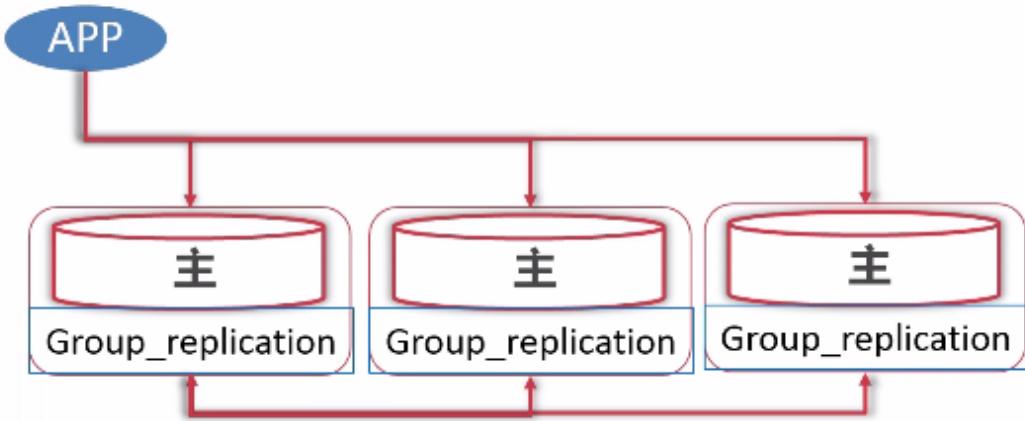
什么是MGR复制

MGR ((mysql group replication))

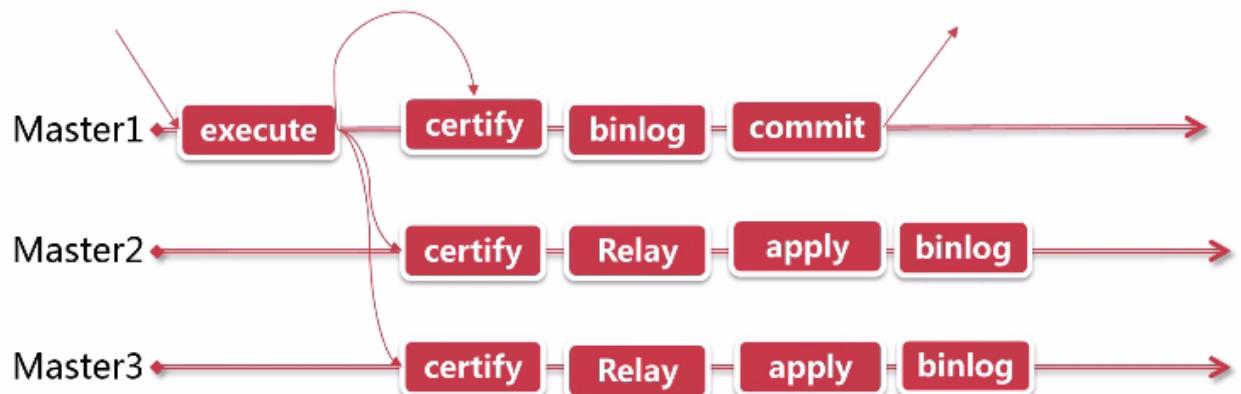
官方推出的一种基于paxos协议的复制

是一种不同于异步复制的多master复制集群

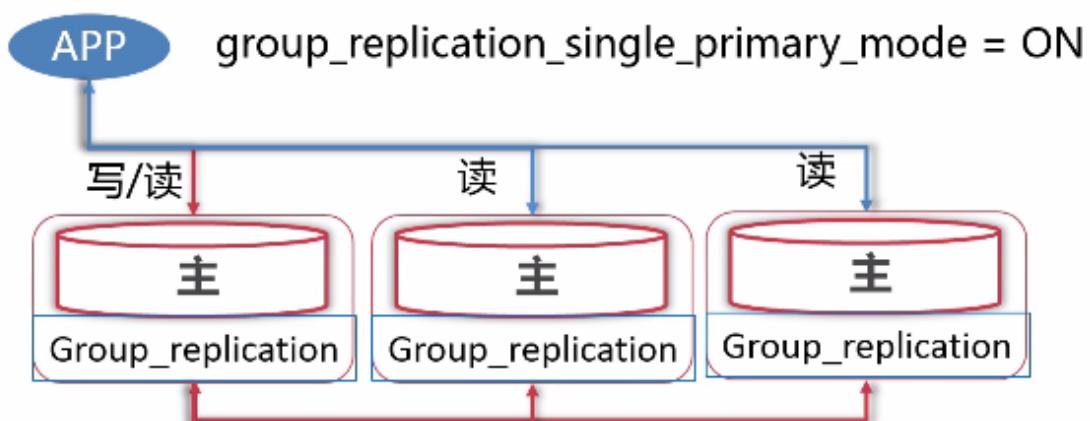
MGR复制架构



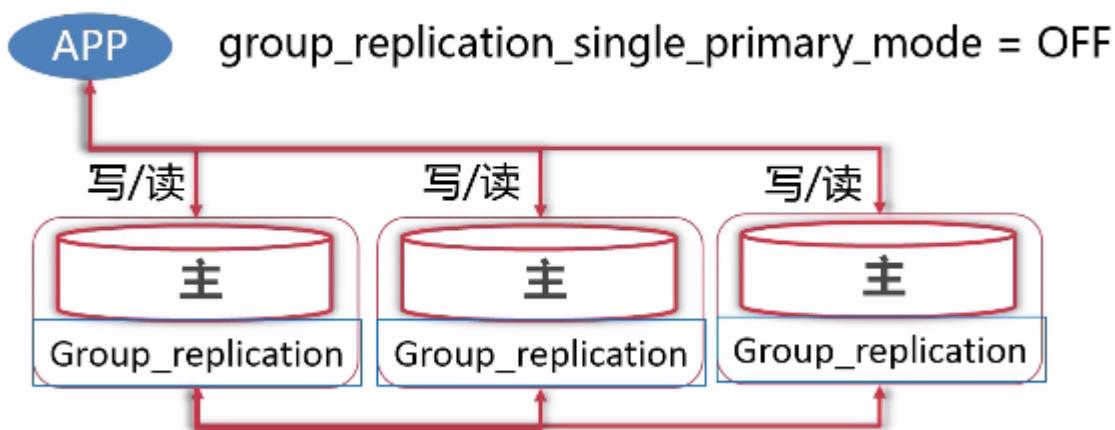
MGR复制的实现原理



MGR的两种模式—单主模式



MGR的两种模式—多主模式



MGR架构需要的资源

集群大小	投票数	允许宕机数量
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

如何使用MGR复制

配置步骤：

- 1、安装group_replication插件
- 2、在第一个实例上建立复制用户
- 3、配置第一个组实例
- 4、把其他实例加入组

当前MGR的优缺点及适用场景

优点：

- 1、group replication组内成员间基本无延迟；
- 2、可以支持多写操作，读写服务高可用；
- 3、数据强一致，可以保证不丢失事务。

缺点：

1、只支持InnoDB存储引擎的表，并且每个表上必须有一个主键

2、单主模式下很难确认下一个primary

3、只能用在gtid模式的复制形式下，且日志格式必须为row

适用场景：

1、对主从延迟十分敏感的应用场景

2、希望可以对读写提供高可用的场景

3、希望可以保证数据强一致性的场景

- 如何解决数据库读/写负载大的问题？

解决读负载大的问题：

1、为原DB增加slave服务器；

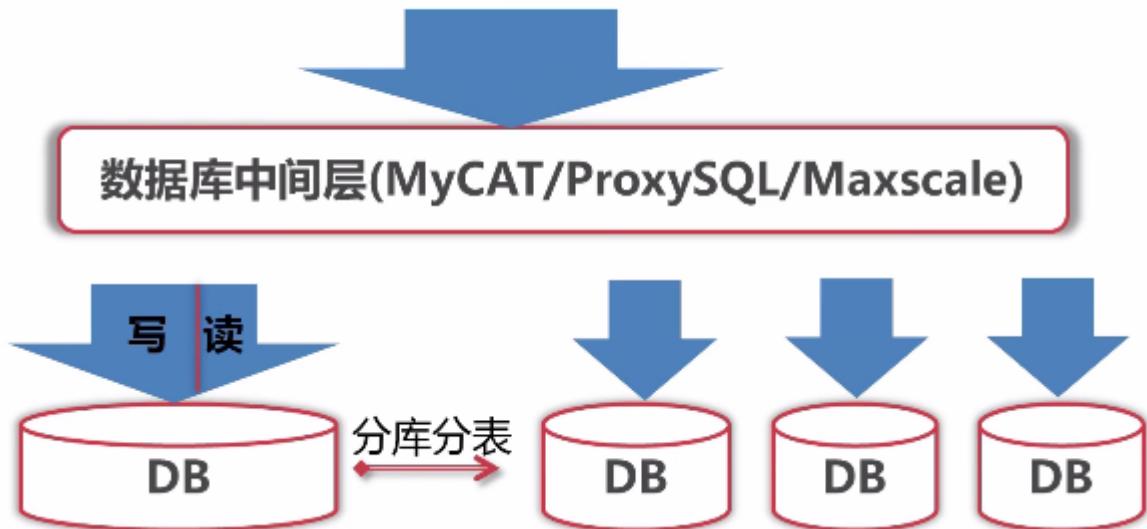
2、进行读写分离，把读分担到slave；

3、增加数据库中间层，进行负载均衡

解决写负载大的问题：

分库分表

如何解决写负载大的问题



备份恢复类问题

- 在之前的工作中是如何对数据库进行备份的？

知识点：

1、备份方式

逻辑备份：数据库表结构及结果SQL。

物理备份：直接拷贝MySQL数据文件。（memory存储引擎，这种方式不能备份其数据）

全量备份：备份全部数据

增量备份：备份上一次备份后的数据

差异备份：备份上一次全量备份后的数据

2、常用的备份工具

mysqldump：常用的逻辑备份工具，支持全量备份及条件备份；

优点：

备份结果为可读的SQL文件，可用于跨版本跨平台恢复数据；

备份文件的尺寸小于物理备份，便于长时间存储；

MySQL发行版本自带工具，无需安装第三方软件。

缺点：

只能单线程执行备份恢复任务，备份恢复速度较慢；

为完成一致性备份需要对备份表加锁，容易造成阻塞；

会对Innodb buffer pool造成污染（经常使用的数据被刷出缓冲池，降低查询效率）

mysqlpump：多线程逻辑备份工具，mysqldump的增强版；（MySQL5.7引入）

优点：（mysqldump的优点都有）

语句同mysqldump高度兼容，学习成本低；

支持基于库和表的并行备份，可以提高逻辑备份的性能；

支持使用 zlib 和 lz4 算法对备份进行压缩

缺点：

基于表进行并行备份，对于大表来说性能较差（一个表只能使用一个线程）

5.7.11之前版本不支持一致性并行备份

会队innodb buffer pool造成污染

xtrabackup：Innodb在线物理备份工具，支持多线程和增量备份；（percona公司提供）

优点：

支持Innodb存储引擎的在线热备份，对Innodb缓冲池没有影响；

支持并行对数据库的全备和增量备份

备份和恢复效率比逻辑备份高

缺点：

做单表恢复时比较复杂

完整的数据文件拷贝，故备份文件比逻辑备份大

对跨平台和数据库版本的备份恢复支持不如逻辑备份

- 如何对MySQL进行增量备份和恢复？

1、逻辑备份+二进制日志（mysqldump+mysqlbinlog）

2、使用xtrabackup工具（innobackupex命令）

使用xtrabackup进行增量备份

- ◆ innobackupex --user=root --password=pwd /backups
- ◆ innobackupex --user=root --password=pwd \
--incremental /home/db_backup/ \
--incremental-basedir=/home/db_backup/back-dir

→ 全备或是上一次增量备份的目录

使用xtrabackup进行增量恢复

- ◆ innobackupex --apply-log --redo-only 全备目录
- ◆ innobackupex --apply-log --redo-only 全备目录 \
--incremental-dir=第1....N次增量目录
- ◆ innobackupex --apply-log 全备目录

- 如何对binlog进行备份？

备份方式：

利用cp命令进行离线备份（不能复制正在使用的日志文件）

利用mysqlbinlog命令在线实时备份（MySQL5.6后开始支持）

使用mysqlbinlog命令在线时实备份

- ◆ mysqlbinlog --raw --read-from-remote-server \
--stop-never --host 备份主机IP --port 3306 \
-u repl -p xxxxxxx 启始二进制日志文件名

→ 具有replication slave 权限

管理及监控问题

- 说说你都对MySQL进行过哪些监控？

MySQL常见的监控指标：

1、性能类指标：

QPS：数据库每秒钟处理的请求数量；

```
#获取MySQL自启动以来服务器执行的请求数量  
show global status like 'Com%';  
#用sum函数统计，或使用以下语句  
show global status like 'Queries';  
#QPS= (Queries2-Queries1) /时间间隔
```

TPS：数据库每秒钟处理的事务数量；

```
#获取总数量  
show global status where Variable_name in ('Com_insert','Com_delete','Com_update');  
#Tc ≈ Com_insert+Com_delete+Com_update  
#TPS ≈ (Tc2-Tc1)/时间间隔
```

并发数：数据库实例当前并行处理的会话数量；

```
#查看数据库当前的并发线程数  
show global status like 'Threads_running';
```

连接数：连接到数据库会话的数量；

```
#查看数据库当前的连接线程数量  
show global status like 'Threads_connected';  
#报警阀值：Threads_connected/max_connections > 0.8
```

缓存命中率：innodb的缓存命中率；

```
#读取数据的次数  
show global status like 'Innodb_buffer_pool_red_requests';  
#从物理磁盘读取的次数  
show global status like 'Innodb_buffer_pool_reads';  
#命中率计算公式：  
#(Innodb_buffer_pool_red_requests -  
Innodb_buffer_pool_reads) / Innodb_buffer_pool_red_requests * 100%  
#繁忙系统中，命中率一般要求不小于95%
```

2、功能类指标：

可用性：数据库是否可正常对外提供服务；

```
#1、周期性连接到数据库服务器并执行SQL测试(select @@version)
#2、使用以下命令测试
mysqladmin -uxxxx -pxxx -hxxx ping
```

阻塞：当前是否有阻塞的会话；（一个事务锁住了其他事务所需的资源）

```
#version<5.7,在information_schema库中运行
select b trx_mysql_thread_id as '被阻塞线程',b trx_query as '被阻塞
SQL',c trx_mysql_thread_id '阻塞线程',c trx_query as '阻塞SQL',(unix_timestamp()-
unix_timestamp(c trx_started)) as '阻塞时间' from innodb_lock_waits a join innodb_trx b on
a.requesting_trx_id=b trx_id join innodb_trx c on a.blocking_trx_id=c trx_id where
(unix_timestamp()-unix_timestamp(c trx_started))>30;
#version>=5.7, 在sys库中运行
select waiting_pid as '被阻塞线程',waiting_query as '被阻塞SQL',blocking_pid as '阻塞线
程',blocking_query as '阻塞SQL',wait_age as '阻塞时间',sql_kill_blocking_query as '建议操作'
from innodb_lock_waits where (unix_timestamp()-unix_timestamp(wait_started))>30;
```

死锁：当前事务是否产生了死锁；（事务相互锁住了对方所需的资源）

```
#查看最近一次死锁的信息
#show engine innodb status;
#1、使用pt工具
pt-deadlock-logger u=root,p=123456,h=127.0.0.1 --create-dest-table -dest
u=root,p=123456,h=127.0.0.1,D=crn,t=deadlock
#2、使用MySQL错误日志，开启记录死锁信息
# set global innodb_print_all_deadlocks=on;
```

慢查询：实时慢查询监控

```
#1、通过慢查询日志监控
#2、通过information_schema.`PROCESSLIST`表实时监控，以下为示例
select * from PROCESSLIST where time>60 and command<>'sleep';
```

主从延迟：数据库主从延迟时间；

```
#1、使用以下命令，查看Seconds_Behind_Master的值;
show slave status;
#2、master上启动一个线程周期性更新监控表的数据，slave上启动一个线程查询监控表的数据算出延迟（可写程序实
现或用pt工具），以下为shell命令：
pt-heartbeat --user=xx -password=xx -h master --create-table --database xxx --update --
daemonize --interval=1;

pt-heartbeat --user=xx -password=xx -h slave --database xxx --monitor --daemonize --log
/tmp/slave_lag.log;
```

主从状态：数据库主从复制链路是否正常；

```
#1、使用以下命令，查看slave_IO_Running和slave_SQL_Running的值;
show slave status;
```

- 这些监控是如何实现的？

优化及异常处理

- 请例举三个你曾经处理过的让你印象深刻的问题。

1、数据库服务器负载过大的问题

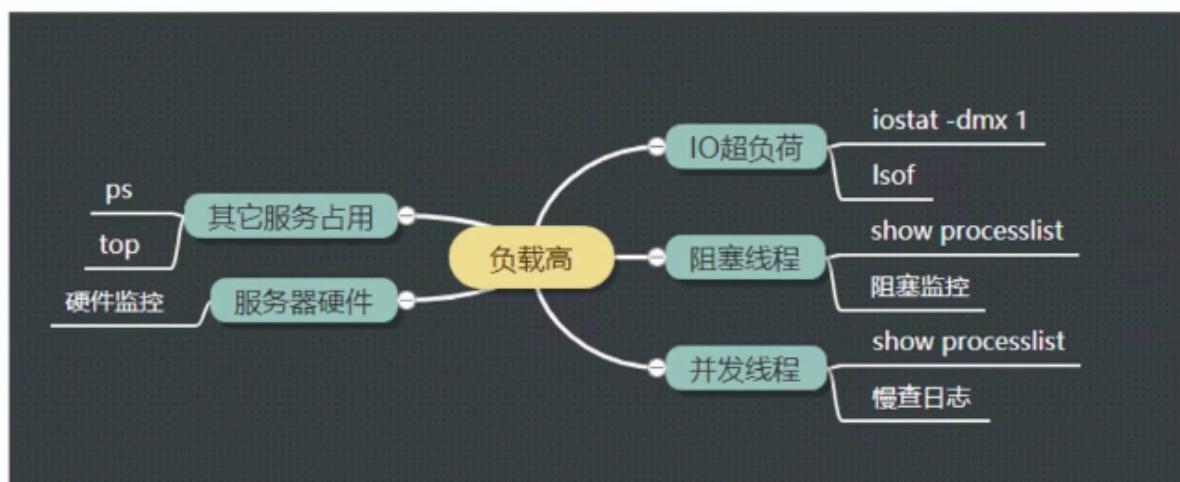
可能原因：

- 1) 、服务器磁盘IO超负载
- 2) 、存在大量阻塞线程
- 3) 、存在大量并发慢查询
- 4) 、存在其他占用CPU的服务
- 5) 、服务器硬件资源故障

排查思路：

27402-E 上 97 -

解决数据库服务器负载过大问题



2、慢查询造成的磁盘IO爆表

可能原因：

- 1) 、MySQL输出大量日志
- 2) 、MySQL正在进行大批量写
- 3) 、慢查询产生了大量的磁盘临时表

解决思路：

优化慢查询，减少使用磁盘临时表

增加tmp_table_size和max_heap_table_size参数的大小（会造成大量的内存消耗）

3、主从数据库不一致

现象：

主从数据延迟为0；

IO_THREAD和SQL_THREAD状态为YES

相同查询在主从服务器中查询结果不一致 (checksum table table_name获取的值不一致)

可能原因：

- 1)、对从服务器进行了写操作
- 2)、使用sql_slave_skip_counter或注入空事务的方式修复错误
- 3)、使用了statement格式的复制

解决思路：

设置read_only=on

设置super_read_only=on

使用row格式的复制

使用pt_table_sync修复数据

```
pt-table-sync --execute --charset=utf8 --database=stock --table=stock --sync-to-master  
h=172.16.188.99,u=root,p=123456
```

- 处理过哪些MySQL主从复制异常？

1、主服务器连接不上

排查思路：

- 1、主从服务间网络是否通畅； (ping)
- 2、是否存在防火墙，过滤了数据库端口； (telnet)
- 3、复制链路配置的用户名和密码是否正确，是否有相应权限。

2、主键冲突问题

可能原因：

- 1)、人为操作了slave数据库中的数据；
- 2)、slave服务器出现过宕机； (relay_log默认存放在文件中，重复执行relay_log)

解决思路：

跳过故障数据，检查主从数据一致性（使用pt-table-checksum工具）

直接删除从库主键冲突数据

3、数据行不存在

可能原因：

- 1)、人为操作了slave数据库中的数据；
- 2)、使用sql_slave_skip_counter或注入空事务的方式修复错误；

解决思路：

跳过故障数据

使用pt_table_sync修复数据

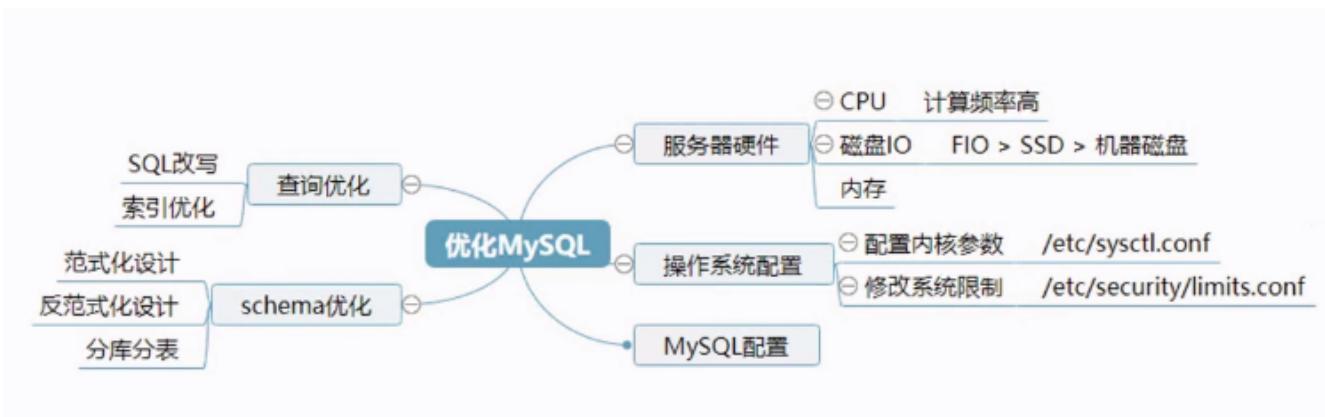
4、relay_log损坏

解决思路：

- 1)、找到已经正确同步的日志点； (show slave status)
- 2)、使用reset slave 删除relay_log

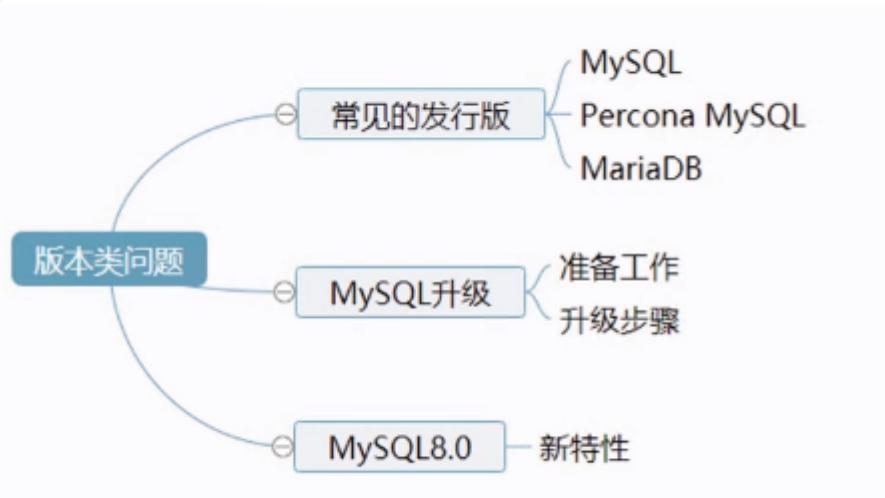
3) 在正确同步日志点后重新同步日志 (change_master)

- 你会从哪些方面对MySQL数据库进行优化?

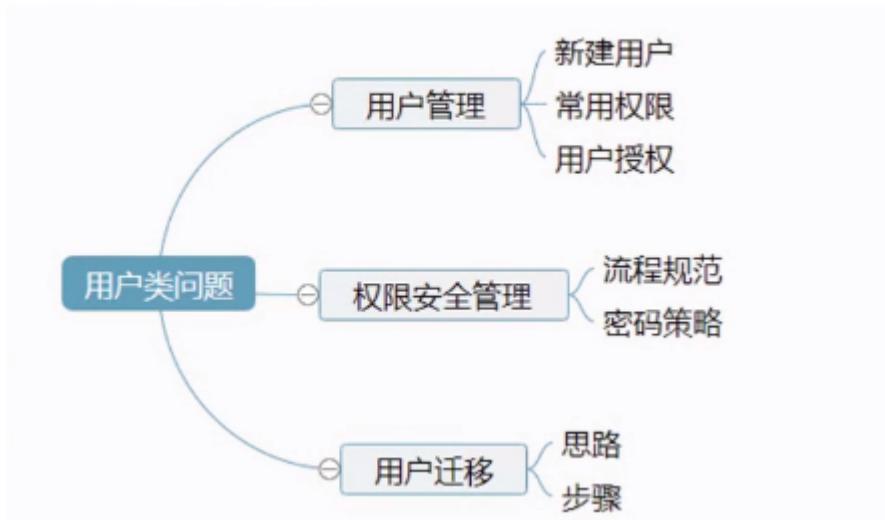


总结

MySQL版本类问题



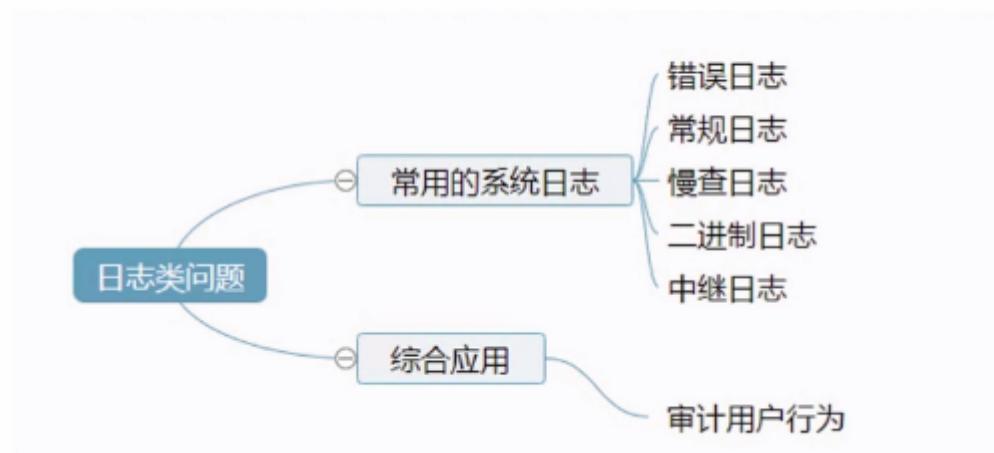
MySQL用户类问题



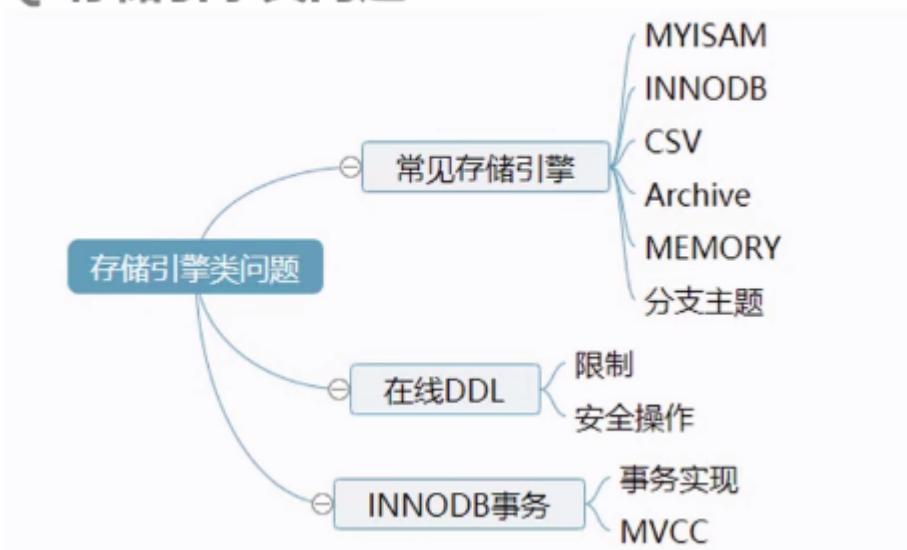
MySQL配置类问题



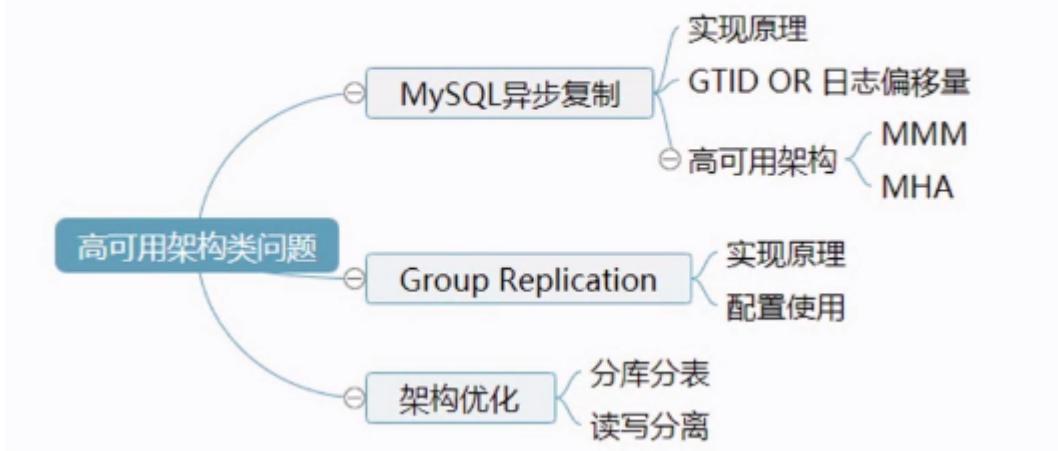
MySQL日志类问题



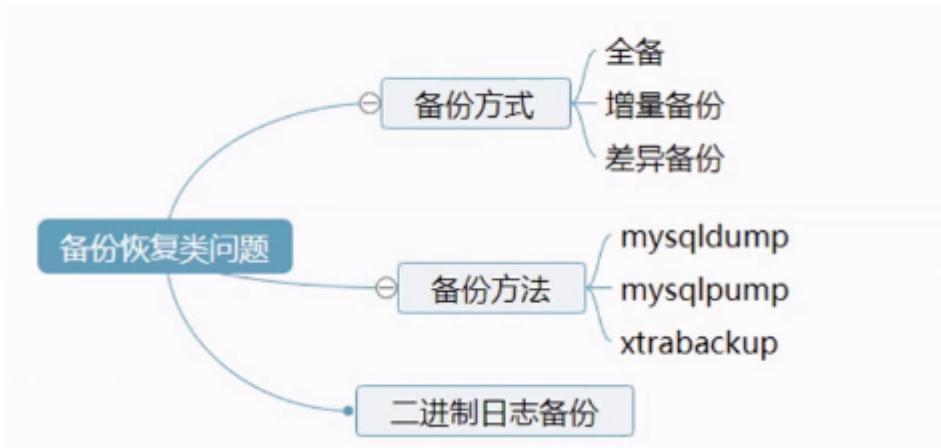
MySQL存储引擎类问题



↑ MySQL 架构类问题



↑ MySQL 备份恢复类问题



↑ MySQL 监控及管理类问题



↑ MySQL常见问题处理

