### 背景

1. **为什么要做GORM**

关系数据库扩容麻烦，性能较低。针对全区全服等数据扩容需求比较强的游戏大都使用Redis集群，成本较高，数据导入分析系统麻烦，不符合联运要求。

1. **GORM设计目标**

代码层面约束数据层使用方式。并针对游戏特性提供高性能、方便的API。

线上快速扩缩容。在线上数据层做到开发与运维解耦。

### **主要功能**

* 自动生成SQL建表语句
* SQL语句与内存结构(Protobuffer)的自动转换
* 针对游戏特性的数据操作接口(例如封装list,set,sort等常用接口)
* 基于版本号的数据一致性校验机制
* 数据缓存，数据自动冷热分离
* 数据层线上快速扩容
* 数据层不停机更新元数据

### **架构**

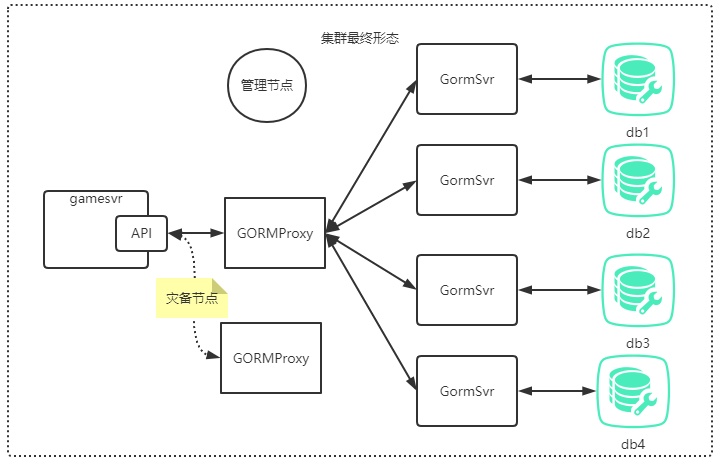
线上每张表的数据最多会被分为1024份，存储在不同的mysql中，例如8个MySql，每个MySql中存储128张表。

GORM涉及到如下5种实体节点：

* GormClient 游戏数据操作节点，调用GormClient API操作数据，暂时实现了如下8个接口，其中两个接口需要同时操作多条数据。
* GORM\_CMD\_INSERT = 3, // 增加记录
* GORM\_CMD\_REPLACE = 4, // 有则替换，没有则插入
* GORM\_CMD\_INCREASE = 5, // 增量更新请求
* GORM\_CMD\_GET = 6, // 单条查询请求
* GORM\_CMD\_DELETE = 7, // 删除请求
* GORM\_CMD\_BATCH\_GET = 8, // 批量查询请求
* GORM\_CMD\_GET\_BY\_PARTKEY = 9, // 批量请求，获取部分字段
* GORM\_CMD\_UPDATE = 10, // 更新请求
* GormPoxy 此节点会做数据路由，批量请求的拆分与响应的组装，在线上扩容期间对客户端屏蔽扩容过程。
* GormSvr 此节点为MySql操作与热点数据缓存节点。每个GORMSvr和对应的后端DB连接。和对应的后端DB处于一个可用区。
* 管理节点，管理GormSvr的节点生命周期，与路由等元数据管理
* MySql 节点，数据落地节点。

1. **集群最终形态**

下面为集群的最终形态，目前通过配置文件配置数据的路由。

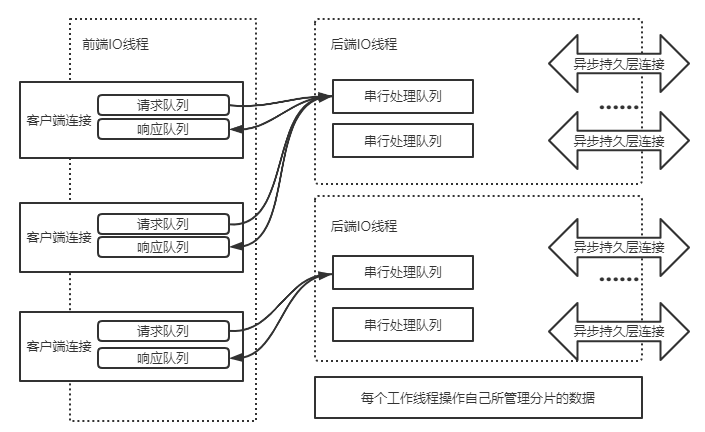


集群最终架构

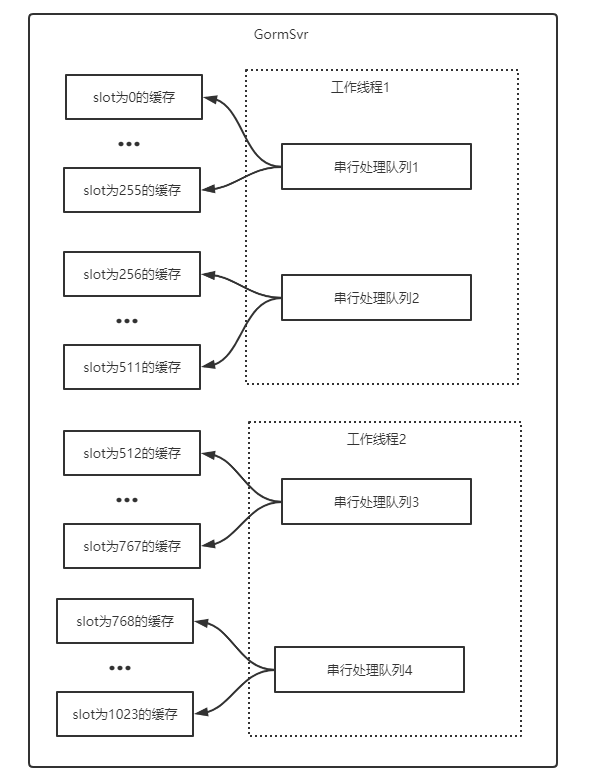
1. **GORM服务器内部视图**

GORM服务器中分为两种线程，接入线程与工作线程。接入线程负责网络IO与任务分发。工作线程负责数据存取操作。

GORM中的数据会根据路由字段hash之后分成1024份，GORM服务器端一共创建了1024个串行化处理队列，服务端启动n个工作线程，每个线程处理若干个请求队列。请求数据过来之后会根据请求的数据的hash值将请求放入后端串行化队列中。后端工作线程处理完之后将结果返回给接入线程，接入线程将结果发送给客户端。



请求处理示意图



缓存结构示意图

### **接口与协议**

1. **通信协议格式**
2. 握手过程

客户端连接上服务器之后需要先获取一个客户端ID，或者发生断线重连校验本地ID是否还有效，可用于断线重连之后获取断线之前发送的请求的结果（请求具有幂等性）。

请求协议格式：4字节 | 1字节操作类型(0，握手) |3字节请求编号 |1字节flag | 4字节ID(如果新客户端则为0) |串行ID

响应协议格式：4字节 | 3字节请求编号| 1字节错误码 |1字节flag| 4字节客户端ID | 串行ID

1. 正常通信协议

对于单请求，客户端填充路由信息，代理不需要解包就可以将请求路由到对应的GORMSvr。

请求协议格式：4字节长度 | 1字节操作类型 | 3字节请求编号 | 1字节flag | 2字节路由|请求内容

响应协议格式：4字节长度 |3字节请求编号 | 1字节错误码 | 1字节flag | 响应内容

1. **操作协议**

****

1. **接口**

目前支持8种操作接口，接口对应的协议见上面gorm\_pb\_proto.proto

GORM\_CMD\_INSERT = 3, // 增加记录

GORM\_CMD\_REPLACE = 4, // 有则替换，没有则插入

GORM\_CMD\_INCREASE = 5, // 增量更新请求

GORM\_CMD\_GET = 6, // 单条查询请求

GORM\_CMD\_DELETE = 7, // 删除请求

GORM\_CMD\_BATCH\_GET = 8, // 批量查询请求

GORM\_CMD\_GET\_BY\_PARTKEY = 9, // 部分key查询请求

GORM\_CMD\_UPDATE = 10, // 更新请求

### **使用方式**

1. 定义结构





1. 研发运维两个SDK版本

数据层SDK分为debug版本与release版本。

Debug版本以提高研发效率为主要目标。在没有增加表，只是增加字段的情况下，不需要重新编译SDK。具体方案为每个表中预留一个MySQL的JSON类型字段。动态增加的字段会被保存在JSON数据类型中。

Release版本主要考虑线上系统的性能与可靠性。MySQL的JSON操作效率较低，Release版本不支持JSON类型字段。

1. 生成关系映射代码与建表语句

$PWD/tools/gorm-conv/bin/gorm-conv -I=$PWD/conf/ -O=$PWD/conf/ -pb=true --cpppath=$PWD/src/tables/

$PWD/tools/bin/protoc --proto\_path=$PWD/conf/ --cpp\_out=$PWD/src/tables $PWD/conf/gorm-db.proto $PWD/conf/gorm\_pb\_tables\_inc.proto $PWD/conf/gorm\_pb\_proto.proto

1. 生成动态链接库

make lib

生成gorm-client.so gorm-tables.so

1. 使用api操作数据



代码示例



1. 配置

### **缓存实现与更新**

1. 无锁操作缓存

数据缓存会分为1024份，对应不同的处理队列，根据配置分给对应的工作线程(串行化处理队列)，GormSvr中每个工作线程操作自己的缓存。缓存更新与获取不涉及到锁。

1. 更新部分字段不需要重新编码

缓存编码为protobuff，为了提高内存的管理效率，缓存方案选择缓存protobuff编码之后的二进制数据，而不是protobuffer结构体本身。

标量只使用定长字段，例如fixed32，更新标量的时候不需要对缓存重新编码，只需要做覆盖更新。字符串通过二次编码加入容量与已使用两个字段，更新长度不超过已分配容量不需要对缓存重新编码，只需要做覆盖更新。

更新protobuff二进制数据需要自己实现对二进制数据的解析工作(由于数据只有一层，字符串为两层)。编解码工作可在一周左右完成。

1. 缓存更新与淘汰

将数据分为3种，运维人员可以在xml配置文件中标识这三种数据，1不需要缓存的数据，2需要缓存不重要的数据，3需要缓存并且需要实时落地的数据。

缓存更新与淘汰就是操作这三种数据。

* 不需要缓存的数据，使用不频繁的数据，直接操作后端db
* 需要缓存的不重要的数据，例如玩家的登陆时间等信息，异步刷入后端db
* 需要缓存的重要的数据，例如玩家的背包，存取比较频繁，并且重要性比较高，更新的时候需要实时落地

1. 缓存扩容

目前规划为离线扩容，在线扩容工作在做完代理之后，和代理配合做。

1. 缓存层灾备

在线热备需要做主从功能。目前规划为冷备，线上采用n+m的形式部署，n为在线提供服务的节点，m为提供灾备的节点，n中有一个down机，提供不了服务，则从m中选择一个出来顶替down的节点。

和etcd等第三方组件配合判断线上GormSvr是否不可达，实现节点的管理功能。

1. 避免缓存击穿

针对业务频繁存取不存在的数据（例如活动期间判断玩家是否满足活动条件，需要频繁判断玩家是否有某种数据），提供拦截服务，防止请求打到后端持久化层。

在系统长期使用之后bloom过滤器的过滤效果比较差。可以通过缓存全部主键信息，或者缓存对不存在的数据的查询来避免。

1. 避免缓存雪崩

缓存过期时间不设置为一样的，业务上导致的雪崩概率比较低。主要是避免缓存进程更新维护重启，缓存数据丢失。可以通过缓存配置主从两个节点，或者缓存放入共享内存，GormSvr重启缓存数据不丢失避免。此功能优先级比较低。

### **维护**

1. 处理能力扩容
2. 数据层灾备
3. 更新元数据

### **工作进度**

1. **项目地址**

<https://git.devcloud.ztgame.com/hongchao/gorm>

发布版本目录

<https://git.devcloud.ztgame.com/hongchao/gorm/-/tree/master/release>

目录中的example是使用示例

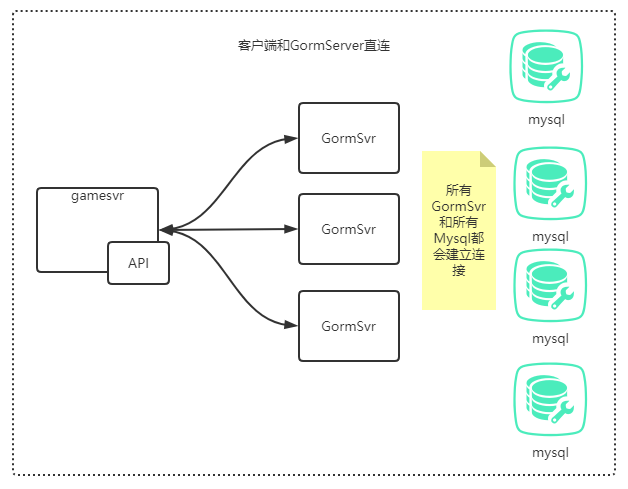
1. **开发进度**

* 目前已经完成的工作:

目前已实现GormSvr单节点，多线程服务(按配置的splitinfo中的信息路由)，客户端和单GormSvr直连，直接存取MySql。

实现数据的版本号校验机制，数据更新需要重新编译动态库，重启GormClient与GormSvr。

如果需要多个GORMSvr提供集群服务，目前需要客户端自己和多个GORMSvr建立连接。这种方式的缺点:批量请求需要客户端自己拆成单个请求，所以集群的情况下目前不支持批量请求(GORM\_CMD\_BATCH\_GET与GORM\_CMD\_GET\_BY\_PARTKEY两个接口)。



* **接下来工作计划**

**工作内容**

* 数据缓存与淘汰功能
* 缓存线上灾备
* 缓存层在线扩容功能
* 代理节点开发
* 冷数据扩容功能

**时间节点**

* 8月中下旬左右提供单节点Gormsvr缓存功能(主要工作量为protobuff二进制数据的解析，3种数据的落地策略，缓存自动淘汰，内存满淘汰，冷热数据比性能指标)。
* 9月上旬左右提供缓存扩容与灾备功能(主要工作量为不可用节点判断与剔除，选择一个备节点提供线上服务，并且所有客户端更新路由将数据路由到新的节点)

到9月这个节点可以提供线上比较安全的数据落地与缓存服务。可以根据实际需求调整开发方向，如果针对非全区全服只需要单GormSvr服务的业务，可以优先开发list,set,sort等游戏常用接口。针对扩容需求比较强的游戏可以优先开发代理，优先实现集群在线扩容功能。

* 10月初提供代理服务，可以在集群中使用批量请求。
* 10月底提供在线扩容热数据服务(扩容方案可以选择在线搬迁热数据，也可以不搬迁热数据，优先满足项目进度，选择难度较小的方案)。

后端冷数据存储，在线扩容目前需要选择tidb，离线扩容可以选择mysql。

### **Redis缓存方案**

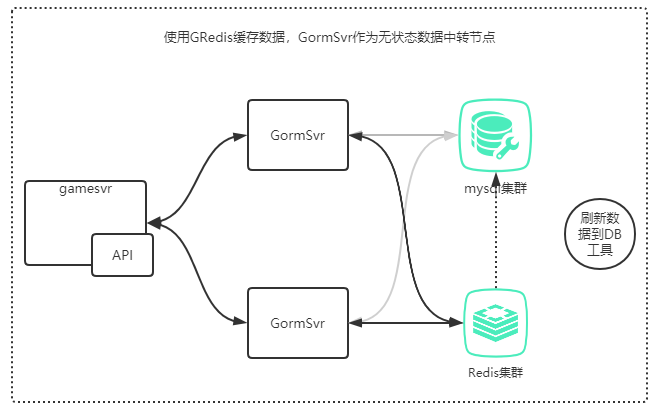
使用GRedis缓存数据与保证串行化操作，GormSvr和所有MySQL与Redis互通，取代代理节点作为无状态数据中转节点，省去代理、缓存层开发工作量，批量接口也可以使用。

使用Redis缓存则数据更新方式有两种。一种是数据只有被拉入Redis之后再使用，更新之后异步落地到MySQL，这种方案能提高数据的读写能力。第二种是数据被使用则拉去Redis，修改则实时写入MySQL并更新Redis，Redis只提供对数据读的缓冲服务，对写的性能没有帮助。

* 下面为针对第一种使用方式的一些说明：

内存淘汰策略可以选择Redis原生淘汰策略：内存满则随机淘汰设置了过期时间的数据。数据已经落地则设置一个过期时间，数据有更新则将数据的过期时间去掉防止被淘汰。

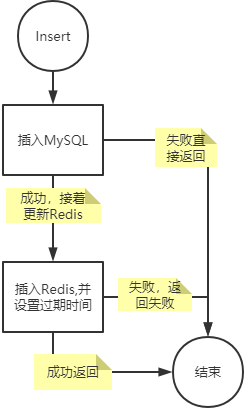
此方案需要做好Redis内存监控，如果热数据太多，来不及刷入MySQL容易造成Redis内存暴涨(如果内存用完Redis会出问题，有down机风险)。由于Redis一条命令不能提供条件判断并更新的功能，更新Redis缓存的时候需要使用LUA脚本保证操作的原子性与数据版本号检验，防止缓存覆盖问题。



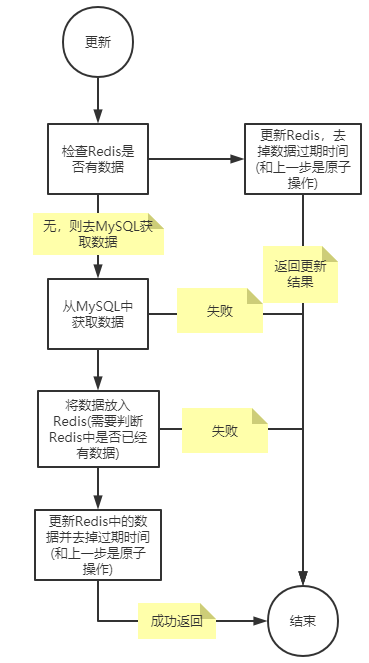
GRedis缓存数据方案

选择方案二为了保证数据的一致性，如果缓存不可用则系统不可用。下面为插入，更新，获取流程图。

1、插入操作



1. 更新操作



1. 获取操作

