[一、 Update List 2](#_Toc428)

[二、 背景 2](#_Toc13296)

[三、 主要功能 2](#_Toc12235)

[四、 架构 2](#_Toc20657)

[五、 接口与协议 4](#_Toc2890)

[六、 使用方式 6](#_Toc13197)

[七、 GormProxy实现方案 7](#_Toc25290)

[八、 缓存实现与更新 8](#_Toc3187)

[九、 维护 10](#_Toc6918)

[十、 GORM实践探索 10](#_Toc30613)

### Update List

##### 2020-10-08

增加章节 4-1， [GORM两种工作模式（单机与集群模式）](#_单机模式)

增加章节 6-7，[直接使用struct操作持久化数据接口例子](#_开发操作的接口)

增加章节8-8，[针对分区分服游戏数据缓存在数据操作层(GORM驱动中)](#_针对分区分服游戏数据缓存在数据操作层(GORM驱动中))

增加章节10，[GORM实践探索](#_GORM实践探索)

### 背景

##### 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。

##### 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。

### 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)，开发只需操作struct结构体，无需了解底层存储细节
* 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
* 支持全区全服与分区分服游戏架构，并提供方便的区服数据访问接口

### 架构

GORM有两种工作模式，单机模式与集群模式，单机模式与集群模式之间可以通过配置切换，切换工作模式对开发透明。

##### 单机模式

单机模式就是GORM客户端驱动直接连接MySQL。适合用于开发环境中。

##### 集群模式

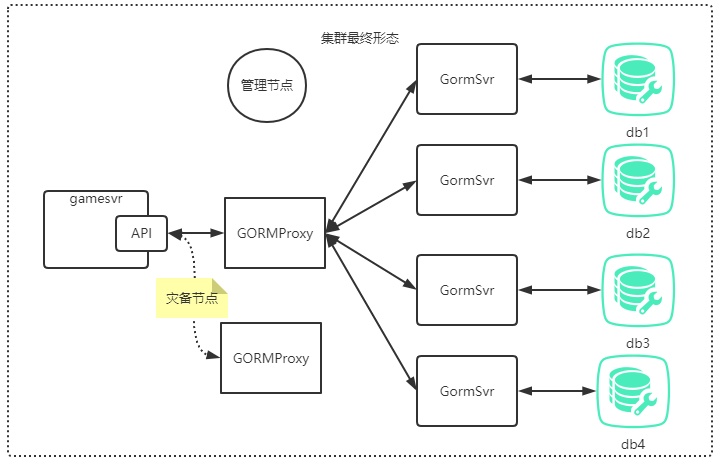
线上每张表的数据最多会被分为1024份，存储在不同的mysql中，例如8个MySql，每个MySql中存储128张表。

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，对游戏屏蔽集群细节，在扩容期间屏蔽扩容过程。对数据请求进行监控与异常流量的拦截，对后端DB起到保护作用。
* GormSvr 此节点作为单机MySql的抽象，和后端db直接连接。负责操作MySQL与热点数据缓存。和对应的后端DB处于一个可用区。MySql采用最新的8.0版本的驱动，使用异步操作接口，支持单进程并发访问MySQL。
* 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。

##### 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。

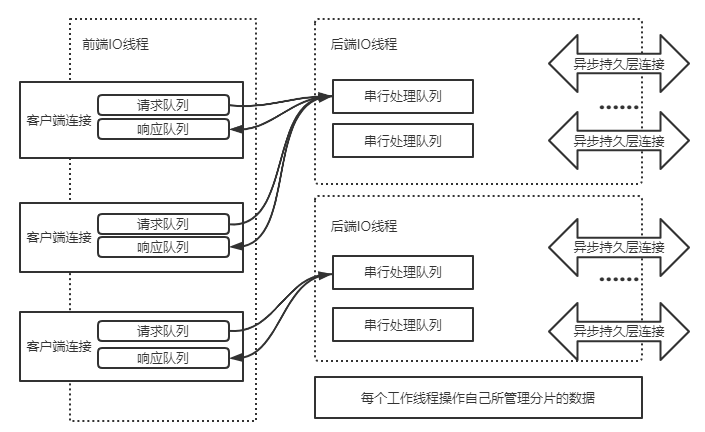


集群最终架构

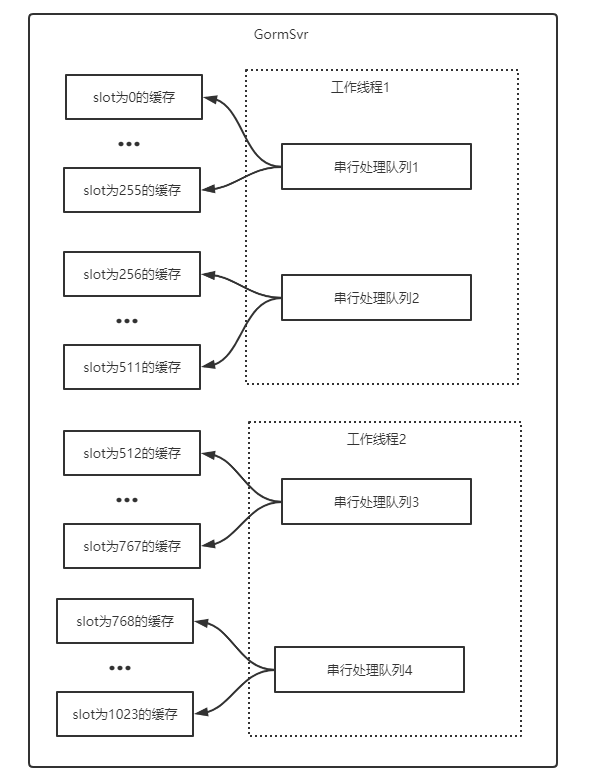
##### GORM服务器内部视图

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

##### 通信协议格式

1. 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）。

在握手期间确定客户端与服务器版本号是否一致。

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 | 请求内容

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|内容

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 | 1字节操作类型 | 3字节请求编号 | 1字节错误码 | 1字节flag | 响应内容

##### 操作协议

****

##### 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

##### 定义结构





##### 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。具体方案为增加表的时候使用alter table add column语句增加表的列。

Release版本主要考虑线上系统的性能与可靠性。线上修改表结构涉及到锁表等操作，消耗较大。Release版本需要手工修改表结构。

##### 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

##### 生成动态链接库

make lib

生成gorm-client.so gorm-tables.so

##### 底层操作接口



代码示例



##### 配置

##### 开发操作的接口

开发只需要操作相应的struct，数据可以自动持久化，或者从数据库中按struct的形式拉入进程。

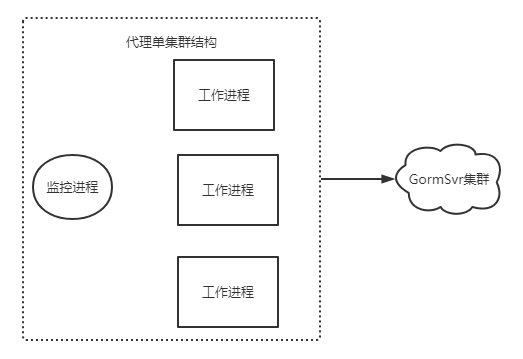
接口例子：刀疤兔currency表，表的主键与路由字段为roleid字段





### GormProxy实现方案

##### 进程模型



由于此节点功能对CPU消耗不大，采用管理进程加上多工作进程模式，在单机情况下可以动态增加工作进程。

工作进程down机通过监控进程实时拉起。

##### GormProxy线上提供多个节点，作为无状态对等节点，同时对外提供服务

线上至少提供两个不同可用区的对等节点提供灾备服务，如果单机GormProxy达到服务瓶颈，启动给一个对等节点即可提高服务能力。

##### 路由管理方式

目前通过配置文件配置路由，采用离线扩容，后端扩容之后将新的路由信息重新加载到代理中。

### 缓存实现与更新

##### 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。

##### 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。

##### 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

##### 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。

##### 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。

##### 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。

##### 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。

##### 单机模式缓存方案

目前游戏中的数据操作节点大都会缓存protobuff数据，特别是分区分区的游戏，会把热点数据全部缓存在数据操作节点中。而且一个区服对应一个数据操作节点。由于此数据节点只操作本区的数据，所以数据操作节点的影响范围只限于本区，down机的情况下影响较小。调研市面上的游戏，大都会有一个数据操作层节点，并且没有对这层节点做主备方案（部分游戏，腾讯系大部分游戏基本都采用把数据缓存在了共享内存中，以减少程序重启、down机等情况下热数据的损失）。

### 维护

##### 处理能力扩容

##### 数据层灾备

##### 更新元数据

### GORM实践探索

##### 分区分服

1. 三个概念

* 可用域(region): 放在一个机房中的数据为一个可用域

出于访问效率的考虑，分区分服游戏尽量使用本机房的数据。

* 逻辑分区(logic zone): 客户端玩家看到的分区概念

即通常意义的游戏区。通常包括战斗服等

* 物理分区(physics zone): 持久化层数据实际存储的位置(主要用来和region一起确定数据的存储位置)

物理分区主要是针对数据来说的，一个物理分区通常是一个DB，当然也可以是DB集群。在开服的时候逻辑分区和物理分区是一一对应的。合服之后一个逻辑分区会对应多个物理分区。每个区的数据尽量集中在一个DB实例中管理，方便运维与运营使用数据。

针对夸机房数据访问延时较大的问题，可以采用每个机房一个数据库集群，合区在本机房内的逻辑分区之间进行。如下图10-1所示。

1. 账号登陆系统、区服列表列系统

账号系统为玩家登陆的校验账号，在玩家通过区服列表选择区服之前就可以访问到，里面保存了玩家游戏数据所在的物理分区与逻辑分区等必要信息。

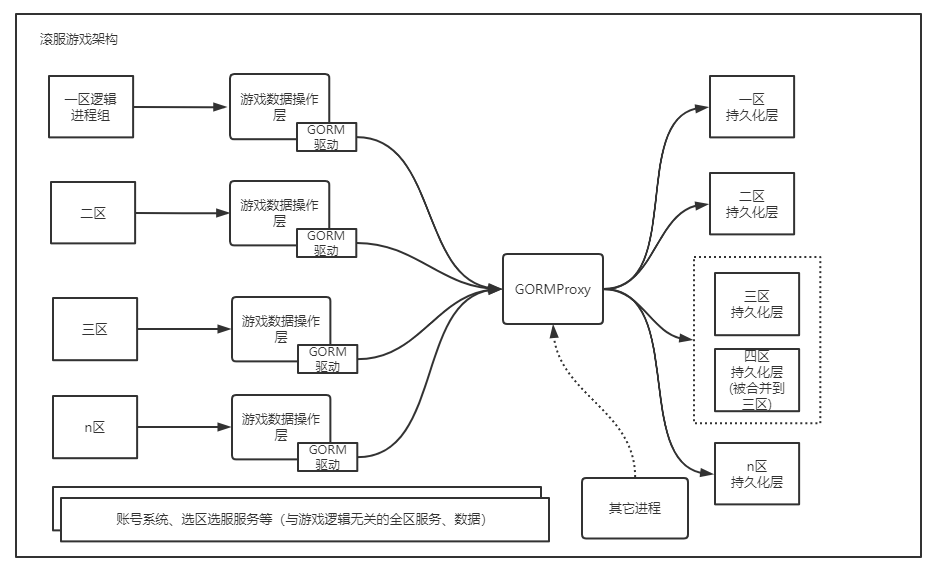
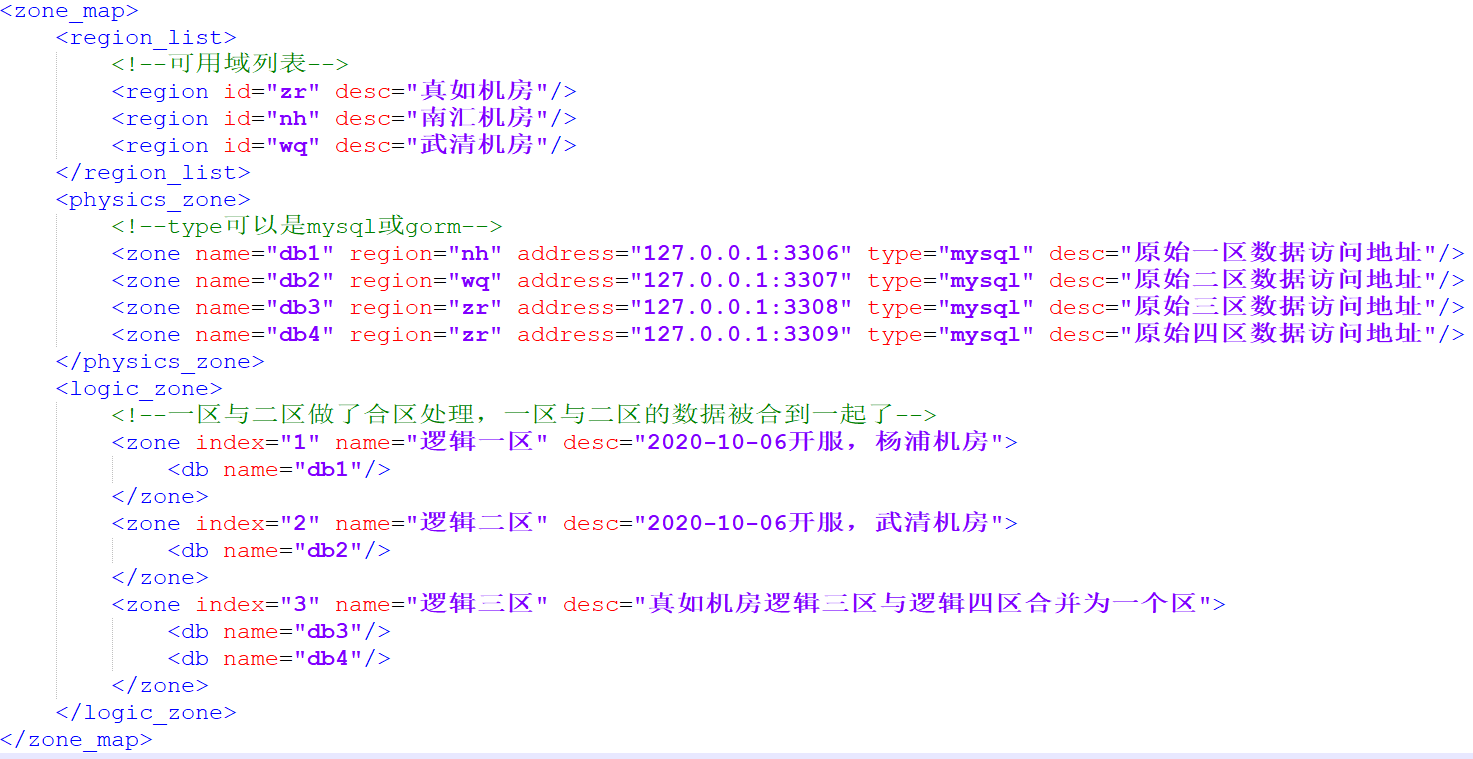


图10-1 分区分服架构图，一个机房一个数据库集群，每个区的数据集中管理，方便使用



GORM驱动层配置

##### 全区全服

没有分区分区的概念，后端数据直接集群化存储，并且根据默认路由，路由到所有物理库中，配置路由规则的时候，同一个玩家的数据尽量根据玩家id来路由，这样一个玩家的数据会被路由到一个可用域内（物理DB中），方便使用。