Video模块文档

接口分析

video视频微服务共提供了6个rpc，其具体实现位于`services/video/handler.go`文件中，下逐一分析。

putVideo

```
rpc putVideo(putVideoRequest) returns (putVideoResponse){};
```

- **功能**

  客户端通过调用`putVideo` rpc将一个本地的视频上传到tiktok服务器，供未来下载、查询使用

- **请求参数**

```
message putVideoRequest{
  string play_url = 1;            // 视频所在地址
  string cover_url = 2;           // 视频封面所在地址
  string title = 3;               // 视频标题
  int64 owner_id = 4;             // 视频发布者id
}

```

- **响应参数**

```
message putVideoResponse{
  bool state = 1;                 // 视频上传是否成功
  string title = 2;               // 视频标题
  int64 owner_id = 3;             // 视频发布者id
  string err_state = 4;           // 视频上传错误时的错误类型
}
```

deleteVideo

```
rpc deleteVideo(deleteVideoRequest) returns (deleteVideoResponse){};
```

- **功能**

  根据视频名称，从tiktok服务器中删除一个视频

- 请求参数

```
message deleteVideoRequest {
  string title = 1;               // 视频标题
  int64 deletor_id = 2;           // 删除者id，用于判断该用户是否有权限删除视频
```

```
4 }
```

- 响应参数

```
message deleteVideoResponse {
  bool state = 1;                    // 是否删除成功
  string delete_video_name = 2;      // 被删除的视频标题
  int64 deletor_id = 3;              // 删除者id
  int64 video_owner_id = 4;          // 视频发布者
  string err_state = 5;              //视频删除失败时的错误类型
}
```

getOneVideoInfo

```
rpc getOneVideoInfo(getOneVideoInfoRequest) returns (getOneVideoInfoResponse){};
```

- 功能

  获得一个指定视频的相关信息

- 请求参数

```
message getOneVideoInfoRequest {
  string video_name = 1;                // 视频标题
}
```

- 响应参数

```
message getOneVideoInfoResponse {
  bool state = 1;                    // 是否成功获取视频
  int64 video_id = 2;                // 视频ID
  string play_url = 3;               // 视频播放地址
  string cover_url = 4;              // 视频封面地址
  string video_title = 5;            // 视频标题
  int64 video_size = 6;              // 视频大小
  string video_mime_type = 7;        // 视频格式
  int64 owner_id = 8;                // 视频发布者id
  string err_state = 9;              // 获取视频失败时的错误类型
}
```

downloadOneVideo

```
rpc downloadOneVideo(downloadOneVideoRequest) returns (downloadOneVideoResponse)
```

- 功能

  用于下载一个指定视频，用户调用该rpc后会获得视频的url，在浏览器中打开该url即可获取视频

- 请求参数

```
message downloadOneVideoRequest {
  string video_name = 1;                // 视频标题
}
```

- 响应参数

```
message downloadOneVideoResponse {
  bool state = 1;                       // 下载视频状态
  string video_title = 2;               // 视频标题
  string video_url = 3;                 // 视频url地址
  int64 owner_id = 4;                   // 视频发布者id
}
```

downloadMultiVideo

```
rpc downloadMultiVideo(downloadMultiVideoRequest) returns (downloadMultiVideoRes
```

- 功能：

  批量下载多个视频，获得视频对应的url，在浏览器中打开该url即可获取视频

- 请求参数：

```
message downloadMultiVideoRequest {
  int64 video_number = 1;               // 视频数量
  int64 downloador_id = 2;              // 下载视频者id
}
```

- 响应参数

```
message downloadMultiVideoResponse {
  bool state = 1;                       // 下载视频状态
  int64 video_number = 2;               // 下载视频数量
  repeated string video_urls = 3;       // 存放视频url的数组
  string err_state = 4;                 // 下载视频失败时的错误类型
}
```

downloadMaxVideo

```
rpc downloadMaxVideo(downloadMaxVideoRequest) returns (downloadMaxVideoResponse)
```

- 功能

  批量下载指定上限数量的视频，该上限值由系统确定

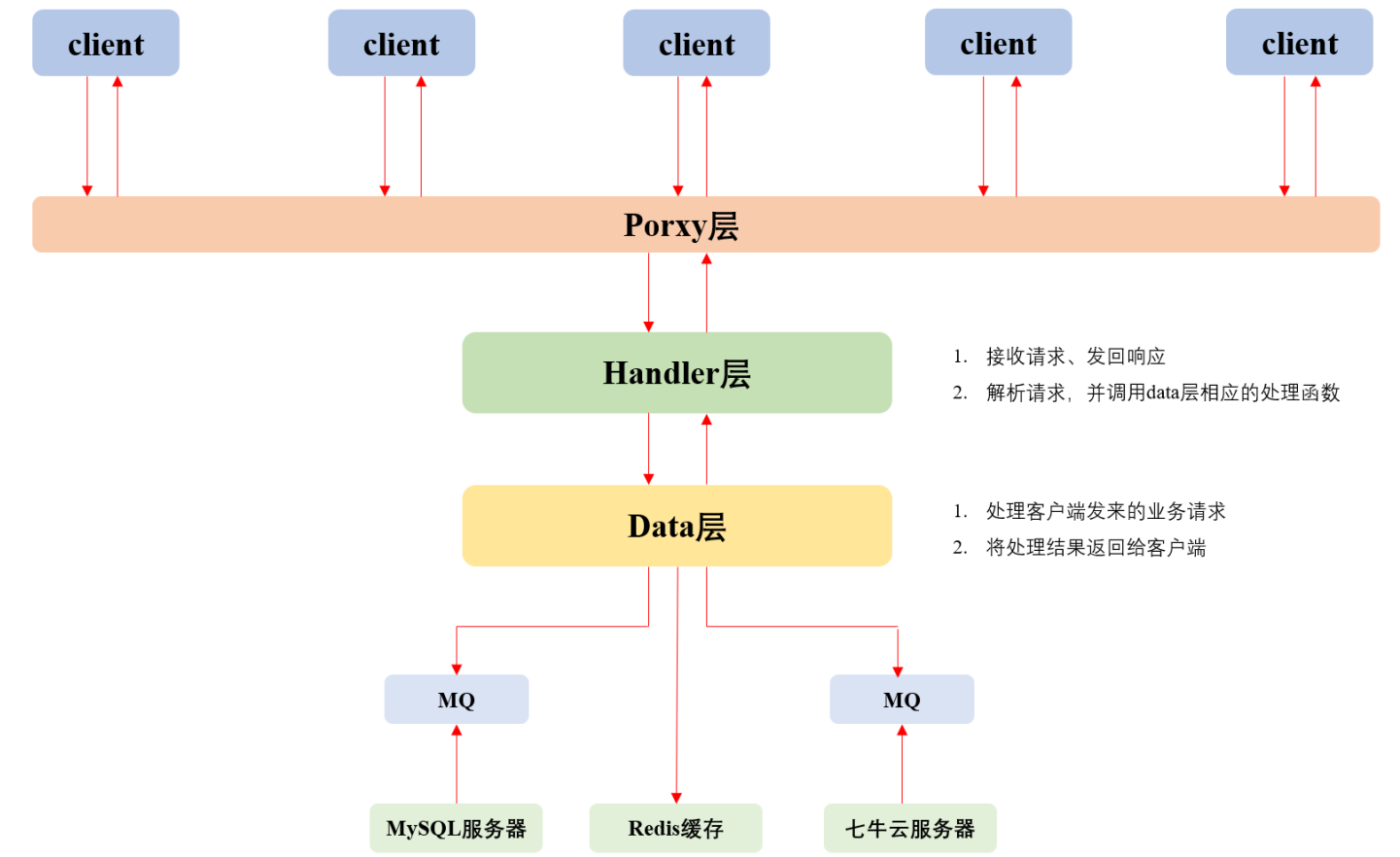
- 请求参数

```
message downloadMaxVideoRequest {
  int64 downloador_id = 1;                // 下载者id
}
```

- 响应参数

```
message downloadMaxVideoResponse {
  bool state = 1;                         // 下载视频状态
  string video_info = 2;                  // 下载视频信息，json字符串
  string err_state = 3;                   // 下载视频失败时的错误类型
}
```

架构设计



测试用例

video视频微服务模块共提供了5个测试用例，都位于`services/video/client`目录下，下逐一分析。

up_multi_video_info

- 功能

  将本地的多个视频上传到。

- 实现细节

其中视频来源为`video`目录下的mp4视频，封面图片来源于`picture`目录下的jpg图片。将要上传的视频、封面、视频标题、发布者id打包进请求消息后，调用`PutVideo`rpc远程函数，将请求发送给`Video微服务`，`Video微服务`模块会将视频保存在服务器中。

get_one_video_info

- 功能

  获取指定的视频的相关信息，包括视频id、视频播放url、截图url、视频大小、视频发布者id等信息。

- 实现细节

  将想要获取的视频的标题放入请求消息中，调用 `DownloadOneVideo` rpc远程函数，将请求发送给Video微服务，Video微服务会通过查询Mysql视频存储表得到视频的相关信息，，并将这些信息存储在响应消息中，返回给客户端。

down_one_video

- 功能

  下载一个指定的视频。

- 实现细节

  将想要下载的视频的标题放入请求消息中，调用 `DownloadOneVideo` rpc远程函数，将请求发送给Video微服务，Video微服务会通过查询Mysql视频存储表得到视频id，然后根据视频id查询七牛云存储库，得到视频对应的url链接，并将该链接存储在响应消息中，返回给客户端。

down_multi_video

- 功能

  下载多个视频。

- 实现细节

  大多数时候用户对视频内容并没有要求，因此客户端可以通过调用 `DownloadMultiVideo` rpc远程函数，将请求发送给Video微服务，Video微服务会在Redis缓存中随机获取若干视频id和视频url的键值对，并将这些视频信息存储在响应中，发回给客户端。

delete_one_video

- 功能

  删除一个指定的视频。

- 实现细节

  将要删除的视频id和删除者id放回请求消息中，然后客户端调用 `DeleteVideo` rpc远程函数，将请求发送给Video微服务，Video微服务负责在系统中删除该视频相关的内容

模块设计

接入层和数据处理层分离

将整个Video微服务系统分为Handler接入层和Data数据处理层，其中Data数据处理层包括MySQL微服务和七牛云存储库微服务。这样设计是为了将模块充分解耦：Handler层只负责接收请求，调用Data层提供的服务，并发回响应；Data层只负责处理具体的业务逻辑，管理MySQL数据库和七牛云存储库。

用户权限检查

以 `deleteVideo` rpc为例，只有视频的发布者才可以删除视频。因此当有用户要删除某视频时，需要先根据主键视频ID从MySQL数据库中检索得到视频的发布者，在确定视频删除者与发布者是

同一个人后，方可执行具体删除操作。用户权限检查流程可以提高Video模块的安全性，避免越权操作。

令牌桶限流器

限流器是提升服务稳定性的非常重要的组件，可以用来限制请求速率，保护服务，以免服务过载。限流器的实现方法有很多种，常见的限流算法有固定窗口、滑动窗口、漏桶、令牌桶。令牌桶算法可以支持突发流量，比较适合抖音出现热点事件的场景。

服务降级

由于爆炸性的流量冲击，对一些服务进行有策略的放弃，以此缓解系统压力，保证目前主要业务的正常运行。在本模块中，当Handler层失败调用次数达到一定阀值自动降级，使用异步机制探测回复情况启动服务降级，将请求存入Redis缓存中，停止将视频存储到MySQL数据库表和七牛云存储库中，待服务平稳之后再执行这些请求。

高并发

将Handler接入层和Data数据处理层分离后，Handler层以协程的方式并发调用Data层的服务，此时Handler层直接将响应消息发回给客户端，提高并发性。

Redis缓存

在MySQL服务器中使用Redis缓存若干视频信息，并使用定时器定期更新缓存中的视频。当客户端发来的请求中对视频内容没有要求时，直接从该缓存中获取视频，提高查询速度，同时减轻MySQL数据库的负载。针对以下Redis缓存常见的问题，我们也给出了适合的解决方案。同时当用户在MySQL查询一个

- 缓存穿透

  从缓存取不到的数据，在数据库中也没有取到，这时将要写入缓存的key-value对写为key-null，缓存有效时间设置为20秒。

- 缓存雪崩

  缓存数据的过期时间设置随机，防止同一时间大量数据过期现象发生。

- 缓存污染

  将缓存中的视频信息过期时间设置为20秒，这样会较快速过期，同时每检索一次缓存中的视频信息，将该条记录的过期时间增加10秒，同时设置过期上限60分钟，防止热点事件长期占用缓存。

MySQL主从复制机制

在开发工作中，有时候会遇见某个SQL语句需要锁表，导致暂时不能使用读的服务，这样就会影响现有业务，使用主从复制，让主库负责写，从库负责读，这样，即使主库出现了锁表的情景，通过读从库也可以保证业务的正常运作。

考虑到抖音业务场景下读操作数量远远大于写操作，因此本模块采用了一主一从的架构模型，其中所有写操作均发送给Master服务器，80%读操作发送给Slave服务器，20%读操作发送给Master服务器，从而降低Master服务器的负载。这一分流过程在Handler层处理，Handler层根据业务请求类型，将请求发送给对应的MySQL服务器。

异常场景处理

用户会发起很多越权操作，本系统在Data层进行异常检测，拦截用户的越权操作。如阻止用户删除不属于自己的视频、阻止用户获取其他用户的私密视频等。

性能分析

具体实现原理

video视频微服务共提供了6个rpc，其具体实现位于`services/video/handler.go`文件中，下逐一分析。

putVideo

- Handler层

使用protobuf规范rpc接口和请求响应参数，kitex框架构建rpc远程函数调用。从proxy中获取用户发送来的请求，从请求中解析得到视频标题等信息。

  - 令牌桶限流器限流

在handler层已使用Go官方的令牌桶算法限流器，若视频长度未超过10分钟则去消费令牌桶中的Token，若无剩余Token则阻塞10秒，超时则直接返回上传失败响应。

  - 雪花算法生成全局唯一ID

若成功消费Token，则使用雪花算法为视频生成唯一的视频id，并利用Go协程将视频相关信息发送给Data层的MySQL服务器和七牛云存储服务器进行进一步的存储，handler直接返回上传成功响应给客户端。

- Data层

Data层主要有MySQL微服务和七牛云微服务，其中MySQL服务器主要负责存储视频相关信息，如视频标题、发布者id等属性信息，七牛云服务器主要负责存储视频id和视频本身的键值对。这样设计是为了减轻MySQL数据库的负担。

  - MySQL微服务

    使用gorm框架打开video存储表，并调用`db.Create(&videostorageInfo)`将视频相关信息存储到该表中，将视频id作为主键进行存储。

  - 七牛云微服务

    基于七牛云GO SDK封装11个工具接口，利用这些接口即可方便的与七牛云存储库交互。通过服务端SDK生成上传凭证后，调用`formUploader.PutFile(context.Background(), &ret, upToken, key, localFile, &putExtra)`将(视频id，视频.mp4)的键值对上传到七牛云存储库中。

deleteVideo

- Handler层

使用protobuf规范rpc接口和请求响应参数，kitex框架构建rpc远程函数调用。从proxy中获取用户发送来的请求，从请求中解析得到要删除的视频id和删除者id信息。

  - 权限校验

    根据视频id调用MySQL微服务中的`DataBaseFindVideoIDByTitle`接口函数，获取该视频的 发布者id。若删除者id和发布者id不匹配，则无权删除，返回删除失败响应。否则利用Go协程将要删除的视频ID发送给Data层的MySQL服务器和七牛云存储服务器进行具体删除操作，Handler层直接返回删除成功响应。

- Data层

  - MySQL微服务

使用Gorm框架打开远程video数据库表，删除相关视频记录

  - 七牛云微服务

使用封装好的存储据工具接口，从七牛云存储库中删除相关记录。

getOneVideoInfo

- Handler层

使用protobuf规范rpc接口和请求响应参数，kitex框架构建rpc远程函数调用。从proxy中获取用户发送来的请求，从请求中解析得到视频id。由于所有视频都是公开的，因此不需要权限校验即可直接获取。Handler层将获取的视频ID发送给Data层的MySQL服务器进行具体查询操作，在查询结束后，将查询结果封装在封装在响应中发回给客户端。

- Data层
  - MySQL微服务

  使用Gorm框架打开远程video数据库表，根据主键视频id查询得到相关视频属性信息，返回给Handler层

downloadOneVideo

- Handler层

使用protobuf规范rpc接口和请求响应参数，kitex框架构建rpc远程函数调用。从proxy中获取用户发送来的请求，从请求中解析得到要下载的视频ID。由于所有视频都是公开的，因此不需要权限校验即可直接下载。

- Data层
  - 七牛云微服务

使用封装好的存储据工具接口，根据视频ID这一Key在七牛云存储库中检索得到视频的相关记录，使用七牛云SDK中的 `storage.MakePublicURL` 函数得到视频对应的公开url，并将该url返回给Handler层。

downloadMultiVideo

- Handler层

使用protobuf规范rpc接口和请求响应参数，kitex框架构建rpc远程函数调用。从proxy中获取用户发送来的请求，从请求中解析得到要下载的视频的数量。由于所有视频都是公开的，因此不需要权限校验即可直接下载。并将下载数量打包后发送给Data层MySQL服务器，在成功获取视频后，将视频信息打包后将响应发回给客户端。

- Data层
  - MySQL微服务

  在MySQL服务器中使用Redis建立缓存，用来存放若干视频ID和视频url的键值对。设置定时器每5ms更新一次缓存中的视频，从七牛云微服务中随机获取若干视频url，并写入缓存。因为大多数时候用户对视频内容并没有要求，因此只需要返回任意随机的视频即可，此时就可以从该Redis缓存中随机获取指定数量的视频，并返回给Handler层

downloadMaxVideo

与downloadMultiVideo实现类似。