TMSS(Tencent Media Stream Server)设计

# 背景和功能

TMSS（Tencent Media Stream Server）是高性能可扩展的流媒体server。

## 功能

### 做什么

行为：推流、回源、拉流、转推；

能力：收敛；

性能：协程化；多核特性

### 不做什么

应用层协议；转封装；转码；回调第三方接口

## 性能

**方案一**，基于多线程，线程之间传递的是fd，而不是实际的message。

**方案二**，基于多进程，进程之间共享数据

## 收敛性

确保同一个流（可根据url、param来唯一确定一个流，规则可配置）只回源一路。

## 扩展性

提供数据处理与回写接口，支持各种协议的扩展。

## 内存管理

内存分配和回收。

切片式数据存放在内存，提高读写性能。

### StreamCache

一个流的存储。

### FileCache

一个域名下的文件cache。

## 静态回源

支持回源和分发并行。

## 进程管理

热重启。

# 基本框架和结构定义

TMSS封装了线程/协程调度、连接管理、传输、cache、收敛等能力并隐藏实现细节。使用者通过接口实现协议、封装和流处理的工作。

方案一，通过编写模块代码，然后configure时加载。

方案二，通过动态库，实现接口。

## 框架

### 进程/线程模型

Proxy

Processor

### 接口分层

ServerLayer（流媒体服务层）

封装了线程/协程调度、连接管理、传输、cache、收敛等能力并隐藏实现细节。

TransportLayer（传输处理层）

应用层传输协议。如SRT、quic等。

ProtocolLayer（协议处理层）

和媒体封装层。包括rtmp、flv、hls、dash、webrtc等。

UserControlLayer（业务逻辑层）

定制化逻辑，主要是和客户逻辑相关。编码层。如转码、混流等。

## 关键数据结构或类定义

### ProxyModule

*{*

*}*

### ProcessorModule

*{*

*}*

*标识processor*

### Stream

*{*

*char key[16]; // hash\_key*

*std::vector<Consumer\*> consumers; // consumer集合*

*std::vector<Source\* > sources; // source集合（一般只有一个输入）*

*}*

标识一个流。

### Source

*{*

*Stream\* stream;*

*std::string origin\_host;*

*Connection connection;*

*std::string origin\_url;*

*std::string origin\_params;*

*int input\_type;*

*…*

*}*

标识一个输入信息，可以是推流或回源，一个Stream暂时只支持一个Source

### Consumer

*{*

*Stream\* stream;*

*Connection connection;*

*Session\* session;*

*…*

*}*

标识一个输出，可以是转推或者拉流。

### Connection

*{*

*Socket socket;*

*std::string ip;*

*int port;*

*int status;*

*}*

标识一个连接

### Address

*{*

*std::string ip;*

*int port;*

*}*

*标识一个地址*

### StaticFile

*{*

*std::string key;*

*int size;*

*void\* data;*

*}*

标识一个静态文件

### Request

*{*

*std::string vhost;*

*std::string path;*

*std::string streamid;*

*std::string ext;*

*std::map<std::string, std::string> params;*

*}*

标识一个请求

### Packet

*{*

*void\* data;*

*int size;*

*}*

标识一个封装后的数据包

### Frame

*{*

*void\* data;*

*int size;*

*}*

标识一个解封装后的音视频帧

### Work

*{*

*void \* req;*

*int fd;*

*int64\_t push\_time;*

*}*

标识一个任务

### EdgeClient

*{*

*Address addr;*

*std::string url;*

*ITransportCtrl\* transport\_ctrl;*

*}*

## 类成员函数

### ProxyModule接口

***int HandleConnect(Conection\* connection);***

连接建立处理函数，里面调用ConnCtrl->HandleConnect

params:

connection,

return:

0, success

*void* ***AddRouteCtrl****(IStreamCtrl\* stream\_control);*

添加路由控制实例

params:

return:

### ProcessorModule接口

隐藏了连接管理、数据传输的实现，调用者实现应用层的协议栈解析和封装。

*int* ***FetchOrCreateStream****(std::string hash\_key, Stream\*\* new\_stream);*

获取一个Stream，没有则创建一个

***int HandlePlay(Conection\* connection, Request\* request);***

播放请求处理函数

params:

request, 请求结构体指针

return:

0, success

api类型是否要单独一个接口

***int HandlePublish(Request\* request)***

输入请求处理函数。推流触发，开始推流时调用。

params:

request，输入url等信息

return:

0, success

***int HandlePacket(Source\* source, Packet\* packet);***

数据包处理函数

params:

packet，输入的一个数据包

return：

= 0，success

***int HandleUnpublish(Source\* source)***

输入结束函数。推流或回源断开。

params:

source，输入源信息

return:

0, success

***int ExitConsumer(Consumer\* consumer);***

consumer退出函数。和播放端的连接断开。

params:

consumer, 输出端信息

return:

0, success

*void* ***AddUserCtrl****(IUserCtrl\* user\_control, std::string vhost);*

添加业务控制接口

params:

return:

*void* ***AddProtocolCtrl****(IProtocolCtrl\* protocol\_control, std::string protocol);*

添加媒体封装协议解析接口

params:

return:

*void* ***AddTransportCtrl****(ITransportCtrl\* transport\_control, std::string vhost);*

添加传输协议接口

params:

return:

*void* ***AddStaticCtrl****(IStaticCtrl\* static\_control, std::string vhost);*

添加静态文件控制接口

params:

return:

*void* ***GetStaticCtrl****(IStaticCtrl\* static\_control, std::string vhost);*

静态文件控制接口

params:

return:

*void* ***GetUserCtrl****(IUserCtrl\* user\_control, std::string vhost);*

业务逻辑控制接口

params:

return:

*void* ***GetProtocolCtrl****(IProtocolCtrl\* protocol\_control, std::string protocol);*

协议控制接口

params:

return:

*void* ***GetTransportCtrl****(ITransportCtrl\* transport\_control, std::string transport);*

传输协议控制接口

params:

return:

### Stream接口

*int* ***fetchOrCreateSource****(std::string vhost, std::string url, Source\*\* new\_source);*

给Stream创建一个Source，针对唯一输入

*int add****Source****(std::string vhost, std::string url, Source\*\* new\_source);*

给Stream添加一个source，通常针对多source的情形

*int* ***fetchOrCreateConsumer****(std::string vhost, std::string url, Consumer\*\* new\_consumer);*

给Stream创建一个Consumer，针对唯一输出

*int* ***addConsumer****(Connection\* connection, Consumer\*\* new\_consumer);*

给Stream添加一个Consumer

*void* ***listConsumers****(list<Consumer\*> consumers);*

查询consumer列表

*int* ***sendToConsumer****(Consumer\* consumer, Packet\* packet);*

发送数据给某个consumer

*void* ***broadcastToConsumers****(Packet\* packet);*

发送所有数据给consumer，会调用consumer->***WriteMsg()***

*int* ***destroyConsumer****(Consumer \* consumer);*

强制断掉consumer。Consumer自己有超时机制，连接断开或超时一定时间后会清理。

*void* ***destroyAllConsumer****();*

强制断掉所有的 consumer

*int* ***initCache****();*

初始化cache，cache通常是header、gopcache等等，所以是一个动态的有序列表，类似于一个sorted<CommonMsg>。

*int* ***getCache****(sorted<CommonMsg\*> msgs);*

获取直播流缓存并更新，输出是sorted<CommonMsg>。

### Source接口

***int HandleIngest(Request \* request)***

回源处理函数，回源源站connect和play成功时调用。创建Stream时如果状态不是publish，则会触发on\_publish。失败状态则变回unpublish。

params:

request，输入url等信息

return:

0, success

*int Source*::***FetchStream(Buffer\* buffer, int &wanted\_size,*** *ITransportCtrl\* transport\_ctrl****)***

### Consumer接口

***int WriteMsg(CommonMsg\* msg);***

输出处理函数，播放端维度的控制

params:

return：

= 0，success

### StaticModule接口

***int HandleUpload(StaticFile\* file, Request\* request);***

上传处理接口

params:

file，完整的一个文件

return:

0, success

***int HandlePlay(Request\* request);***

播放请求接口

params:

return:

0, success

***int HandleIngest(StaticFile\* file, Request\* request);***

回源处理接口

params:

file，完整的一个文件

return:

0, success

### FileEdgeClient : EdgeClient接口

***int StartFetching(std::string address, StaticFile\* file,***

***IStaticCtrl\* static\_ctrl,*** *ITransportCtrl\* transport\_ctrl);*

***int FetchFile(std::string address, StaticFile\* file,***

***IStaticCtrl\* static\_ctrl,*** *ITransportCtrl\* transport\_ctrl****);***

### FileCache接口

静态回源。

以vhost作为key。

***接口***

*int* ***addFile****(std::string file\_key, CommonMsg\* msg);*

添加文件到内存。

*int* ***appendFile****(std::string file\_key, CommonMsg\* msg);*

追加写文件到内存。

*StaticFile\** ***getFile****(std::string file\_key);*

获取文件。

*int* ***deleteFile****(std::string file\_key);*

删除文件。

*void* ***clear****();*

清空存储。

### Buffer接口

*void ReadBytes(char\* data, int size);*

*void WriteBytes(const char \* data, int size);*

### Dispatch接口

int DispatchWork();

int GetWork();

## 枚举类型

*EdgeStatus*

*{*

*EdgeStatusInit,*

*EdgeStatusPublish,*

*EdgeStatusUnpublish,*

*}*

## 接口

### 框架接口

***int InitTransportLayer();***

注册传输相关Ctrl。

***int InitProtocolLayer();***

初始化框架，注册协议和封装相关Ctrl。

***int InitUserControlLayer();***

注册业务逻辑相关Ctrl。

***int InitStaticControlLayer();***

注册静态逻辑相关Ctrl。

### IUserCtrl接口

vhost维度

*int HandlePlay(Request\* request);* // *fetchOrCreateStream*，*fetchOrCreateSource*

*int HandlePublish(Request\* request);* // *fetchOrCreateStream*，*fetchOrCreateSource*

*int HandleFrame(Stream\* stream);*

### IProtocolCtrl接口

protocol维度

*int ReadFrame(Source\* source, Frame\*\* frame);*

*int WriteFrame(Consumer\* consumer, Frame\* frame);*

### ITransportCtrl接口

应用层传输协议维度

*int Connect(Connection\* conn, std::string address);*

*int Send(Connection\* conn,Buffer\* buffer);*

*int Recv(Connection\* conn,Buffer\* buffer);*

*int Disconnect(Connection\* conn);*

### IRouteCtrl接口

***int HandleRoute(Connection\* connection, Reuqest \* request);***

路由函数接口

params:

connection，连接

request，请求信息

return：

= 0，success

### IConnCtrl接口（非必选）

***int HandleConnect(Conection\* connection);***

连接建立处理函数，一般针对有白名单客户需求

params:

connection,

return:

0, success

### IStaticCtrl接口

***int CheckInput(Packet\* packet);***

包完整性检查，如在rtmp场景判断一个完整的tag。

params:

file, 已经存储的file

packet，输入的一个数据包

return：

> 0，receive size

= 0，not complete

*int HandlePlay(Request\* request);*

*int HandleUpload(Request\* request);*

*int HandleIngest(StaticFile\* file,Request\* request);*

## 类成员函数

提供协程封装、通用库等工具类api。

### 基于协程的func

*co\_connect*

*co\_send*

*co\_recv*

*co\_sendAndRecv*

*co\_sleep*

*co\_thread\_create*

*co\_thread\_join*

*co\_thread\_exit*

*co\_cond\_new*

*co\_cond\_signal*

*co\_cond\_wait*

*co\_cond\_destroy*

*co\_mutex\_new*

*co\_mutex\_destroy*

*co\_mutex\_lock*

*co\_mutex\_unlock*

## 类图



## 例子

### flv回源和拉流**（用户）**

**框架**

***int InitUserControlLayer()***

*{*

*ProcessorModule::Instance()->****AddUserCtrl****(new UserCtrl,vhost);*

*ProcessorModule::Instance()->****AddUserCtrl****(new UserCtrl2,vhost2);*

*ProcessorModule::Instance()->****AddUserCtrl****(new UserCtrl2,default\_vhost);*

*}*

***int InitProtocolLayer()***

*{*

*ProcessorModule::Instance()->****AddProtocolCtrl****(new StreamCtrl,”rtmp”);*

*ProcessorModule::Instance()->****AddProtocolCtrl****(new StreamCtrl,”flv”);*

*}*

***int InitTransportLayer()***

*{*

*ProcessorModule::Instance()->****InitTransportLayer****(new TransportCtrl,”srt”);*

*ProcessorModule::Instance()->****InitTransportLayer****(new TransportCtrl,”quic”);*

*}*

**控制接口**

*int UserCtrl::HandlePlay(Request\* request)*

*{*

*std::string stream\_key =*

*request->vhost + request->path + request->streamid****;*** *// 这里用户自定义*

*Stream =*

*Server->fetchOrCreateStream(stream\_key,…);*

*Consumer\* consumer = NULL;*

*Stream->addConsumer(connection, &consumer);*

*sorted<CommonMsg> cache = Stream->getCache();*

*Stream->sendToConsumer();*

*std::string origin\_url =*

*origin\_host + request->path + request->streami;**// 这里用户自定义*

*// connect to origin*

*connection;*

*Stream->fetchOrCreateSource(); // 播放触发回源*

*}*

*int UserCtrl::HandlePublish(Request\* request)*

*{*

*}*

*int UserCtrl::HandleFrame(Stream\* stream)*

*{*

*rtmpProtocolCtrl = ProcessorModule::Instance()->GetProtocolCtrl(“rtmp”);*

*srtTransportCtrl = ProcessorModule::Instance()->GetTransportCtrl(transport);*

*Frame\* frame;*

*rtmpProtocolCtrl->ReadFrame(source, frame, srtTransportCtrl);*

*sorted<Frame> cache = Stream->getCache();*

*// 这里用户自定义*

*if(/\*this is metadata\*/)*

*{*

*cache[METADATA] = flv\_tag;*

*}*

*else if(/\*this is avc\_header\*/)*

*{*

*cache[VIDEO\_HEADER] = flv\_tag;*

*}*

*else if(/\*this is aac\_header\*/)*

*{*

*cache[AUDIO\_HEADER] = flv\_tag;*

*}*

*…*

*protocolCtrl = ProcessorModule::Instance()->GetProtocolCtrl(“flv”);*

*Packet \* packet;*

*protocolCtrl->BroadcastFrame(frame, packet);*

*// 如果是单个consumer*

*protocolCtrl->WriteFrame(consumer, frame);*

*}*

*int RouteCtrl*::*HandleRoute(Connection\* connection, Reuqest \* request)*

*{*

*// route work to processor*

*}*

*int RTMPProtocolCtrl::ReadFrame(Source\* source, Frame\*\* frame, ITransportCtrl\* transport\_ctrl)*

*{*

*Buffer\* buffer;*

*while(true)*

*{*

*Packet \* packet = buffer->pos();*

*source->****FetchStream(****packet->buffer****, size, transport\_ctrl);***

*// if this is a complete tag, check buffer*

*frame = Decode(packet); // 伪代码*

*break;*

***}***

*}*

*int FLVProtocolCtrl::WriteFrame(Consumer\* consumer, Frame\*\* frame)*

*{*

*// encode*

*Packet\* packet = Encode(frame); // 伪代码*

*// 如果需要针对某个****consumer****做定制化需求，也可以自己根据****consumer****信息做****session****存储*

*consumer->stream->* ***sendToConsumer(packet);***

*consumer->stream->****broadcastToConsumers(packet);***

*}*

*int SRTTransportCtrl::Recv(Connection\* conn, char \* buffer, int size)*

*{*

*// use libsrt*

*srt\_read(conn-fd, buffer ,size);*

*}*

### rtmp推流和转推**（用户）**

**控制接口**

*int UserCtrl::HandlePlay(Request\* request)*

*{*

*}*

*int UserCtrl::HandlePublish(Request\* request)*

*{*

*std::string stream\_key =*

*request->vhost + request->path + request->streamid****;*** *// 这里用户自定义*

*Stream =*

*Server->fetchOrCreateStream(stream\_key,…);*

*sorted<CommonMsg> cache = Stream->getCache();*

*Stream->sendToConsumer();*

*std::string output\_url =*

*output\_host + request->path + request->streami;**// 这里用户自定义*

*// connect to output*

*connection;*

*Consumer\* consumer = NULL;*

*Stream->****fetchOrCreateConsumer****(connection, &consumer);*

*Stream-> fetchOrCreateSource();*

*}*

### hls回源和转推

**控制接口**

*int StaticCtrl::HandlePlay(Request\* request)*

*{// example-relay*

*while(true)*

*{*

*std::string file\_key =* ***generateFileKey(request);*** *// 这里用户自定义*

*// check if the file is in cache*

*FileCache->GetFile(file\_key);*

*std::string origin\_url =* ***GenerateOriginUrl(request);*** *// 这里用户自定义*

*file\_edge\_client->****StartFetching(address, file, this, TransportCtrl);***

*// wait and get*

*// output to client*

*// decode m3u8 and get next file*

*}*

*}*

*int StaticCtrl::HandleUpload(Request\* request)*

*{*

*}*

*int StaticCtrl::HandleIngest(StaticFile\* file, Request\* request)*

*{*

*std::string file\_key = StaticCtrl->****generateFileKey(request);*** *// 这里用户自定义*

*file->key = file\_key;*

*FileCache->addFile(file);*

*}*

### 静态回源或上传

**框架**

***int InitStaticControlLayer()***

*{*

*ProcessorModule::Instance()->****AddStaticCtrl****(new StaticCtrl, vhost);*

*}*

**控制接口**

*int CheckInput(Packet\* packet)*

*{*

*// check if the file is complete*

*}*

*std::string GenerateFileKey(Request\* request)*

*{*

*return request->vhost + request->path + request->streamid;*

*}*

*std::string GenerateOriginUrl(Request\* request)*

*{*

*}*

*std::string GenerateOutputUrl(Request\* request)*

*{*

*}*

*int HandlePlay(Request\* request)*

*{*

*std::string file\_key = StaticCtrl->****generateFileKey(request);*** *// 这里用户自定义*

*// check if the file is in cache*

*FileCache->GetFile(file\_key);*

*std::string origin\_url = StaticCtrl->****GenerateOriginUrl(request);*** *// 这里用户自定义*

*file\_edge\_client->****StartFetching(address, file, this, TransportCtrl);***

*// wait and get*

*}*

*int StaticCtrl::HandleUpload(Request\* request)*

*{*

*std::string file\_key = StaticCtrl->****generateFileKey(request);*** *// 这里用户自定义*

*file->key = file\_key;*

*FileCache->addFile(file);*

*std::string origin\_url = StaticCtrl->****GenerateOutputUrl(request);*** *// 这里用户自定义*

*// connect to client*

*connection;*

*// send to client*

*}*

*int HandleIngest(StaticFile\* file, Request\* request)*

*{*

*std::string file\_key = StaticCtrl->****generateFileKey(request);*** *// 这里用户自定义*

*file->key = file\_key;*

*FileCache->addFile(file);*

*}*

### caster推流（如rtp-ts）

**框架**

***int HandleInit()***

*{*

*ProcessorModule::Instance()->****AddProtocolCtrl****(new ProtocolCtrl,”ts”);*

*ProcessorModule::Instance()->****AddTransportCtrl****(new TransportCtrl,”rtp”);*

*ProcessorModule::Instance()->****AddUserCtrl****(new UserCtrl,”default\_vhost”);*

*}*

**控制接口**

*int UserCtrl::HandlePublish(Request\* request)*

*{*

*std::string stream\_key =*

*request->vhost + request->path + request->streamid****;*** *// 这里用户自定义*

*Stream =*

*Server->fetchOrCreateStream(stream\_key,…);*

*sorted<CommonMsg> cache = Stream->getCache();*

*Stream->sendToConsumer();*

*std::string output\_url =*

*output\_host + request->path + request->streami;**// 这里用户自定义*

*// connect to output*

*connection;*

*Consumer\* consumer = NULL;*

*Stream->addConsumer(connection, &consumer);*

*Stream-> fetchOrCreateSource();*

*}*

*int CasterCtrl::HandleFrame(Stream\* stream)*

*{*

*tsProtocolCtrl = ProcessorModule::Instance()->GetProtocolCtrl(“ts”);*

*rtpTransportCtrl = ProcessorModule::Instance()->GetTransportCtrl(“rtp”);*

*Frame\* frame;*

*tsProtocolCtrl->ReadFrame(source->stream, frame, rtpTransportCtrl);*

*sorted<Frame> cache = Stream->getCache();*

*// 这里用户自定义*

*if(/\*this is metadata\*/)*

*{*

*cache[METADATA] = flv\_tag;*

*}*

*else if(/\*this is avc\_header\*/)*

*{*

*cache[VIDEO\_HEADER] = flv\_tag;*

*}*

*else if(/\*this is aac\_header\*/)*

*{*

*cache[AUDIO\_HEADER] = flv\_tag;*

*}*

*…*

*}*

*int RTPTransportCtrl::Recv(Connection\* conn,char \* buffer, int size)*

*{*

*// use libsrt*

*srt\_read();*

*}*

### 多路输入混流

*int UserCtrl::HandlePlay(Request\* request)*

*{*

*std::string stream\_key =*

*request->vhost + request->path + request->streamid****;*** *// 这里用户自定义*

*Stream =*

*Server->fetchOrCreateStream(stream\_key,…);*

*Consumer\* consumer = NULL;*

*Stream->addConsumer(connection, &consumer);*

*sorted<CommonMsg> cache = Stream->getCache();*

*Stream->sendToConsumer();*

*std::string origin\_url =*

*origin\_host + request->path + request->streami;**// 这里用户自定义*

*// connect to origin*

*connection;*

*Stream->fetchOrCreateSource(); // 播放触发回源*

*Stream->addSource(); // 添加第二个输入*

*}*

*int UserCtrl::HandleFrame(Stream\* stream)*

*{*

*rtmpProtocolCtrl = ProcessorModule::Instance()->GetProtocolCtrl(“rtmp”);*

*srtTransportCtrl = ProcessorModule::Instance()->GetTransportCtrl(transport);*

*Frame\* frame0, frame1;*

*rtmpProtocolCtrl->ReadFrame(stream->sources[0], frame0, srtTransportCtrl);*

*rtmpProtocolCtrl->ReadFrame(stream->sources[1], frame1, srtTransportCtrl);*

*Frame\* frame;*

*frame = mix(frame0, frame1);*

*sorted<Frame> cache = Stream->getCache();*

*// 这里用户自定义*

*if(/\*this is metadata\*/)*

*{*

*cache[METADATA] = flv\_tag;*

*}*

*else if(/\*this is avc\_header\*/)*

*{*

*cache[VIDEO\_HEADER] = flv\_tag;*

*}*

*else if(/\*this is aac\_header\*/)*

*{*

*cache[AUDIO\_HEADER] = flv\_tag;*

*}*

*…*

*protocolCtrl = ProcessorModule::Instance()->GetProtocolCtrl(“flv”);*

*Packet \* packet;*

*protocolCtrl->BroadcastFrame(frame, packet);*

*// 如果是单个consumer*

*protocolCtrl->WriteFrame(consumer, frame);*

*}*

### 内部实现主要函数（框架）

*int ProxyModule*::*HandleConnect****(Conection\* connection)***

*{*

*// check ip*

*ConnCheckCtrl*->***on\_connect(connection);***

*}*

*int ProcessorModule::HandlePlay****(Connection\* connection, Request\* request)***

*{*

*userCtrl = ProcessorModule::Instance()->GetUserCtrl(request->vhost);*

*userCtrl->HandlePlay();*

*}*

*int ProcessorModule*::*HandlePublish(Request\* request)*

*{*

*userCtrl = ProcessorModule::Instance()->GetUserCtrl(request->vhost);*

*userCtrl->HandlePublish();*

*}*

***int*** *ProcessorModule****::HandlePacket(Source\* source)***

***{***

*userCtrl = ProcessorModule::Instance()->GetUserCtrl(request->vhost);*

*userCtrl->HandleFrame(source->stream);*

***}***

*int ProcessorModule*::*HandleUnpublish****(Source\* source)***

*{*

*}*

*int ProcessorModule*::*ExitConsumer****(Consumer\* consumer)***

*{*

*}*

*int Source*::***FetchStream(Buffer\* buffer, int &wanted\_size,*** *ITransportCtrl\* transport\_ctrl****)***

*{*

*if(buffer->size() < wanted\_size)*

*{*

*buffer->grow();*

*}*

*transport\_ctrl->Recv(source->connection, buffer->pos(), wanted\_size);*

*}*

*int StaticModule::HandleUpload(StaticFile\* file,* ***Request\* request****)*

*{*

*StaticCtrl = ProcessorModule::Instance()->GetStaticCtrl(vhost);*

*StaticCtrl->HandleUpload();*

*}*

*int StaticModule::HandlePlay(Request\* request)*

*{*

*StaticCtrl = ProcessorModule::Instance()->GetStaticCtrl(vhost);*

*StaticCtrl->HandlePlay();*

*}*

*int StaticModule::HandleIngest(StaticFile\* file, Request\* request)*

*{*

*StaticCtrl = ProcessorModule::Instance()->GetStaticCtrl(vhost);*

*StaticCtrl->HandleIngest();*

*}*

***int StartFetching(std::string address, StaticFile\* file,***

***IStaticCtrl\* static\_ctrl,*** *ITransportCtrl\* transport\_ctrl)*

*{*

*StartCoThread(****FetchFile);***

*}*

***int*** *FileEdgeClient::****FetchFile(std::string address, StaticFile\* file,***

***IStaticCtrl\* static\_ctrl,*** *ITransportCtrl\* transport\_ctrl****)***

*{*

*// connect*

*Packet \* packet;*

*while(static\_ctrl->****CheckInput(packet))***

*{*

*transport\_ctrl->Recv(connection, packet->buffer, size);*

*}*

***HandleIngest****()*

*}*

## 配置

可配置进程/线程数，配置分为跟配置和域名(vhost)配置。

## 部署

单机只用部署一个实例，不用做单机内的调度。

# 代码结构

## ServerLayer

框架层。

common

配置管理，更新、缓存、查询等。基础库及第三方库的封装。

session

会话信息存储和管理。包括stream、source、consumer、等。

proxy

处理接收请求，分配fd到processor

processor

工作线程，接受、处理和发送数据包。

sched

线程和微线程管理。任务调度功能。

cache

静态文件、切片等。cache存储、管理。

## TransportLayer

传输协议处理层。

## ProtocolLayer

流媒体封装层。

protocol

## UserControlLayer

user

用户业务逻辑。

transcode

音视频编解码。