**服务器微服务架构接入手册**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 日期 | 修订人 | 备注 |
| V1.0.0 | 2019/10/01 | 杨锐 | 初版 |
| V1.0.1 | 2019/11/05 | 杨锐 | 1. rtmpfw节点增加输入码率参数 2. rtmpfw节点增加透传参数（兼容海外url上使用） 3. 所有功能节点增加Script数据的传输开关 |
|  |  |  |  |

目录

[一. 文档目的 2](#_Toc21437431)

[二. 微服务架构简介 3](#_Toc21437432)

[三. SDK接入标准 4](#_Toc21437433)

[3.1 HUYA私有协议上行 4](#_Toc21437434)

[四. 老架构向新架构快速迁移 5](#_Toc21437435)

[4.1 新增任务Json 5](#_Toc21437436)

[4.1.1 任务协议 5](#_Toc21437437)

[4.1.2 协议字段说明 6](#_Toc21437438)

[4.1.3 Json详细解析 7](#_Toc21437439)

[4.2 转推CDN的RtmpForward功能 7](#_Toc21437440)

[4.2.1 功能简介 7](#_Toc21437441)

[4.2.2 接入方式 8](#_Toc21437442)

[4.3 混画功能的Mix\_Lua功能 8](#_Toc21437443)

[4.3.1 功能简介 8](#_Toc21437444)

[4.3.2 接入方式 8](#_Toc21437445)

[4.3.3 补充说明 9](#_Toc21437446)

[4.4 数据流的流向控制介绍 9](#_Toc21437447)

[4.4.1 功能简介 9](#_Toc21437448)

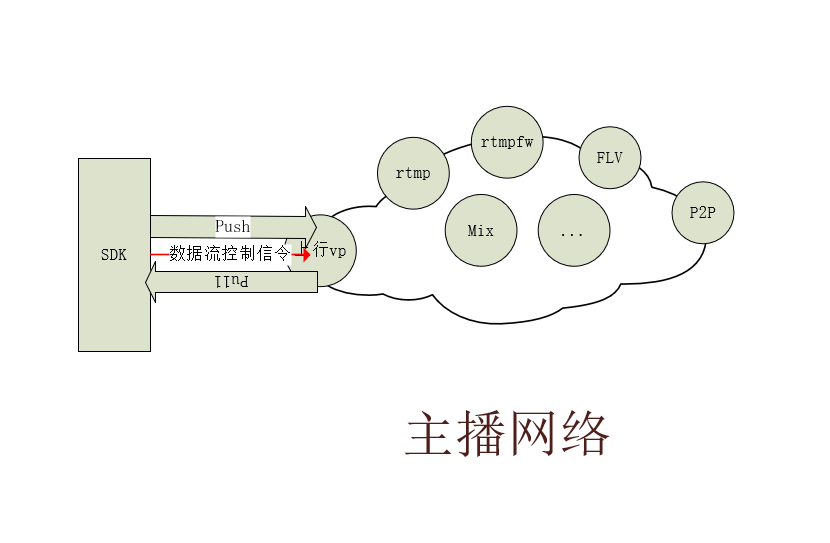
[4.5 APP,SDK相关说明 10](#_Toc21437449)

[4.6 信令时序图 11](#_Toc21437450)

# 一. 文档目的

由于音视频服务器端已经完成初步的微服务架构改造，相比于老架构模式，SDK和一些其他的服务器接入方式都有一些改变。这里整理一个较为详细的说明文档，为以后各端在接入服务器微服务架构的时候提供一个接入标准。

# 二. 微服务架构简介



在普通的直推HUYA场景中，SDK不仅仅只推流（连麦时还要拉交互流），还需要将数据流控制信令发送上来，按照下面的协议，利用主播网络里面的各个微服务模块来进行数据流的处理，最终自己拉流或者将数据推到cdn，以完成需求。目前我们直接对外cdn的节点包括有rtmp,flv,p2p节点。Sdk将数据流和信令推到上行vp。

# 三. SDK接入标准

目前已有的业务场景中，SDK接入HUYA音视频网络主要的方式是HUYA私有协议上行。

## 3.1 HUYA私有协议上行

HUYA私有协议上行，即SDK使用HUYA的私有YY协议格式与音视频网络进行交互，这里面的流程主要包括有：请求分配VP(区分于主播端和观众端)，登录VP，推流，发流通知等等内容。

相关协议说明：（这里只是简单诺列）

|  |  |  |
| --- | --- | --- |
| 协议名（协议号） | 功能说明 | 备注 |
| PLoginVideoProxyUniformly | 登录 |  |
| PStreamData3 | 数据流传输 |  |
| PNotifyCurrentStream4 | 流通知 |  |

暂略，后续有时间再补充….

# 四. 老架构向新架构快速迁移

由于已经有老架构的存在，很多SDK已经有关于老架构的逻辑，在此基础上向微服务的新架构支持的时候，需要额外关注的点在这里单独说明一下，便于这部分的快速接入微服务架构。

相比于老架构来说，需要额外增加以下的逻辑：

## 4.1 新增任务Json

SDK需要在推流的同时，利用与VP的Tcp通道，发送任务Json到VP服务器，服务器会根据任务Json来完成SDK需要服务器对这条流进行的所有操作。

### 4.1.1 任务协议

struct PCloudStreamTaskReq : public sox::Marshallable

{

    enum { uri = (30117 << 8 | protocol::SESSION\_SVID) };

    uint32\_t     appid;             //业务id

    string       streamname;        //主播流名称

    uint32\_t     version;           //任务版本号

    uint32\_t     stmpc;             //时间戳

    map<string, string> stgJson;    //任务细节Json

    virtual void marshal(sox::Pack &pk) const

    {

        pk << appid << streamname << version << stmpc;

        marshal\_container(pk, stgJson);

    }

    virtual void unmarshal(const Unpack &up)

    {

        up >> appid >> straemname >> version >> stmpc;

        unmarshal\_container(up, std::inserter(stgJson, stgJson.end()));

    }

}

struct PCloudStreamTaskRsq : public sox::Marshallable

{

    enum { uri = (30118 << 8 | protocol::SESSION\_SVID) };

    uint32\_t     appid;             //回显

    string       streamname;        //回显

    uint32\_t     version;           //回显

    uint32\_t     stmpc;             //回显

    map<string, string>  resJson;   //任务执行状态返回

    virtual void marshal(sox::Pack &pk) const

    {

        pk << appid << streamname << version << stmpc;

        marshal\_container(pk, resJson);

    }

    virtual void unmarshal(const Unpack &up)

    {

        up >> appid >> straemname >> version >> stmpc;

        unmarshal\_container(up, std::inserter(resJson, resJson.end()));

    }

}

### 4.1.2 协议字段说明

appid: 表明此业务类别，目前HUYA线上环境值为66，测试环境为67

streamname: 主播账号对应的streamname，也即推流的streamname

version: 该任务请求对应的版本号，一般当json内容无变化时，version值不变，json内容变化时，version值+2。version起始值最好与当前时间相关，可用当前秒时间的Linux时间戳。

stmpc: 整个任务创建时的时间戳，用当前秒时间的Linux时间戳。

stgJson: 实际每一项任务对应的json字符串。key值为当前所有任务中的一个唯一字符串标识，具体值可以由端上自定义，但必须保证局部的唯一性。value值为当前key对应的实际json字符串。Json总体格式如下：

{

    "node\_type": 107,

    "input\_stream\_list": [

        {

            "stream\_name": "1234567890-1234567890-1234567890987654321-1234567890-10057-A-0-1-src"

        }

    ],

    "output\_stream": {

        "stream\_name": "1234567890-1234567890-1234567890987654321-1234567890-10057-A-1"

    }

}

resJson: 服务端接收到任务之后的任务启动情况，以此map来返回任务执行状态。key值和stgJson的key值相对应，表示每一个任务的执行状态。value为json字符串，Json格式暂时是预留的，后续进行扩展，前期只需判断json字符串不为空就认为任务正确被执行：{ }

### 4.1.3 Json详细解析

node\_type 表示该任务的任务类型，目前已有的任务类型如下:

enum node\_type

{

    rtmpforward = 107,     //转推

    mix\_lua = 108,         //混画

}

input\_stream\_list 表示该项任务所有的输入列表，支持多输入，每个输入流必带的属性为stream\_name。

output\_stream 表示该项任务的唯一输出流，目前不支持多输出，一个任务只能有一个输出。输出流必带的属性为stream\_name。

关于输入输出的其他属性项需要针对具体不同的任务类型，向其提供定制化的流属性，具体请参考以下node\_type的Json专项说明。

## 4.2 转推CDN的RtmpForward功能

### 4.2.1 功能简介

由于HUYA目前并没能提供CDN功能让观众都直接到HUYA拉流，所以HUYA上行的流，最后都是必须转推到其他家CDN，观众才能到CDN拉流观看。所以我们微服务架构下，提供一个将指定流以指定的方式转推到指定的CDN。目前该服务仅支持rtmp的方式将流推到cdn。

### 4.2.2 接入方式

使用上面介绍的json控制方式，按以下格式填写json即可：

{

    "node\_type": 107,

    "input\_stream\_list": [

        {

            "stream\_name": "1234567890-1234567890-1234567890987654321-1234567890-10057-A-0-1-src",

            "video":

            {

                "switch\_conf":true,

                "code\_rato": 40000,

                "code\_type": "h264"

            },

            "audio":

            {

                "switch\_conf":true

            },

            "other":

            {

                "scriptag":

                {

                }

            },

            "video\_code\_rato":4000

        }

    ],

    "output\_stream": {

        "stream\_name": "1234567890-1234567890-1234567890987654321-1234567890-10057-A-1",

        "rtmp\_transfer\_params": "一些参数字符串"

    }

}

### 4.2.3 参数字段说明

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 一级属性 | 二级属性 | 三级属性 | 四级属性 | 属性值 | 备注 |
| node\_type |  |  |  | 节点属性 | 不可缺少 |
| input\_stream\_list | stream\_name | video | switch\_conf  code\_rato  code\_type | 输入视频的开关  输入视频的码率  输入视频的编码 | Boolean字段，缺省则表示开 四级缺省则表示源码档位  三级缺省则表示源码档位 |
| input\_stream\_list | stream\_name | audio | switch\_conf | 输入音频的开关 | Boolean字段，缺省则表示开  四级缺省则表示源码档位  三级缺省则表示源码档位 |
| input\_stream\_list | stream\_name | other | scriptag | 输入光圈数据 | 四级缺省则表示不输入  三级缺省则表示不输入 |
| output\_stream | stream\_name |  |  | 输出流名称 | 不可缺少 |
| output\_stream | rtmp\_transfer\_params |  |  | url的透传参数 | 缺省则不补充 |

## 4.3 混画功能的Mix\_Lua功能

### 4.3.1 功能简介

当数据流在HUYA音视频网络之后，我们可以对这些流进行拼接处理以完成类似如pk，连麦等业务中的主播画面拼接的功能。

### 4.3.2 接入方式

使用上面介绍的json控制方式，按混画节点约定的json属性来使用即可。

参考《lua混画json.doc》文档



### 4.3.3 补充说明

由于该节点功能较为复杂，所以约定的流属性较多，有单独的文档进行说明。请参考详细说明文档。

## 4.4 数据流的流向控制介绍

### 4.4.1 功能简介

按照上面的描述，每个json只控制单个功能，但是没有说明音视频数据流的一个具体流向，这里做一个简单的说明。

**4.4.2 使用方式**

我们的Json总体格式中规定了，必须要有的属性项就是输入输出流名称（无论是多输入还是单输入），在构造Json的时候，将各个项目的json的输入流名称和输出流名称串联起来，就可以自动完成数据流的定向流动。例如：

rtmp:

{

    "node\_type": 107,

    "input\_stream\_list":

    [

        {

            "stream\_name": "A-smooth-src"

        }

    ],

    "output\_stream":

    {

        "stream\_name": "A"

    }

}

mix:

{

    "node\_type": 108,

    "input\_stream\_list":

    [

        {

            "stream\_name": "A-src"

        }

    ],

    "output\_stream":

    {

        "stream\_name": "A-smooth-src"

    }

}

如上这两个json，就可以让上行流（A-src）先经过混画节点进行处理，然后输出到转推节点，然后转推节点将数据以流名称A推给CDN。

## 4.5 APP,SDK相关说明

由于已经有部分版本的APP接入到了微服务架构中来，这里大致说一下接入中各部分负责的内容，仅供参考。

由APP调用SDK提供的接口发起/修改/停止任务，由于我们采用的是每次都带全量Json信息过来，这里只提供一个接口给APP使用，当Json的map不为空时，任务将会被创建或者修改，当Json的map为空时，服务器将会停掉所有已存在的任务。

另外针对map<string, string> stgJson; 这个做一个说明，这个map的内容实际上由APP端进行组织，SDK缓存并透传到服务器，SDK需要缓存然后用来当作心跳来发送。

几点说明：

1.当APP每次调用修改接口， SDK将需要将version自增2。

2.当SDK和VP的链接断开重连的时候，也需要将version自增2。

3.当需要在原来的json上直接进行修改的时候，请尽量将stgJson对应项的key保持不变，只需要修改Json中的对应内容即可，这样对画面的切换和服务器任务更新比较友好。

## 4.6 信令时序图

