****

**版本号 1.00**

**共 册/共 页**

**杭州登虹科技有限公司**

**转码基础库**

**StreamConvert详细设计说明书**

**编 写 人：Closeli**

**编写时间：2018.06.06**

**部 门 名：ATR**

**审 核 人：**

**审核时间：**

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **变更时间** | **版本** | **变更人** | **审批人** | **变更说明** |
| 1 | 2018/6/5 | V0.1 | 田野 |  | 新建，StreamConvert库详细设计文档 |
| 2 | 2018/6/7 | V0.2 | 田野 |  | 根据评审意见，进行格式修改 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[**1引言** 4](#_Toc516163325)

[**1.1编写目的** 4](#_Toc516163326)

[**1.2背景** 4](#_Toc516163327)

[**1.3定义** 4](#_Toc516163328)

[**1.4参考资料** 4](#_Toc516163329)

[**2模块说明** 5](#_Toc516163330)

[**2.1模块设计** 5](#_Toc516163331)

[**3接口定义** 6](#_Toc516163332)

[**3.1模块外部接口说明** 6](#_Toc516163333)

[**4模块详细设计** 8](#_Toc516163334)

[**4.1** **线程池模块** 8](#_Toc516163335)

[**4.2** **任务创建模块** 11](#_Toc516163336)

[**4.3** **任务队列模块** 14](#_Toc516163337)

[**4.4** **数据缓存模块** 16](#_Toc516163338)

[**4.5** **转封装模块** 16](#_Toc516163339)

[**5 数据结构设计** 17](#_Toc516163340)

[**5.1逻辑结构** 17](#_Toc516163341)

[**5.2物理结构** 20](#_Toc516163342)

[**5.3数据结构和程序关系** 20](#_Toc516163343)

[**6 特定需求** 21](#_Toc516163344)

[**6.1安全** 21](#_Toc516163345)

[**6.2 性能** 21](#_Toc516163346)

[**6.3 可用性** 21](#_Toc516163347)

[**6.4易用性** 21](#_Toc516163348)

[**6.4.1 异常处理** 21](#_Toc516163349)

[**6.5可测试性** 22](#_Toc516163350)

[**6.6可移植性** 22](#_Toc516163351)

[**6.7 可维护性** 22](#_Toc516163352)

[**6.8 兼容性** 22](#_Toc516163353)

[**6.9 系统** 22](#_Toc516163354)

[**7 外部依赖** 22](#_Toc516163355)

[**8 改动/影响** 22](#_Toc516163356)

[**9 测试建议** 23](#_Toc516163357)

[**9.1**功能测试 23](#_Toc516163358)

[**9**.2 整合测试 23](#_Toc516163359)

[**10 当前问题** 23](#_Toc516163360)

[**11 潜在问题** 23](#_Toc516163361)

**1引言**

**1.1编写目的**

本文主要对转封装StreamConvert库的各个模块进行详细设计，定义了各个模块内部的接口和模块间的交互过程以及组件内部和其它模块集成的处理流程，主要用来指导本模块的代码编写和测试验证工作。

本文档预期读者包括：产品经理，项目经理，软件开发工程师，测试工程师等项目相关人员

**1.2背景**

转码基础库StreamConvert作为媒体业务的基础库，主要用于视频数据的封装格式转换。该基础库对外提供相关接口，可以用于相关媒体服务和公司重大项目的快速开发。目前基础库用在转码服务StreamTranscode和OneNet项目中。

该基础库目前由ATR团队负责开发和维护，相关的开发包和成果物可以联系ATR团队提供。

**1.3定义**

本文档中会出现一些缩略语和技术用语，为了简化描述和避免歧义，下面表格将给出详细解释说明。后续在本文档遇到相关词汇，不特殊说明情况下，一切以表格给出的解释为准。

**表 1‑1：技术用于/缩略语**

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 缩写 | 原词 | 详细说明 |
| 1 | RAW |  | 公司内部定义的私有码流封装格式 |
| 2 | TS | TransportStream | 是一种视频封装格式MPEG2-TS，特点是可以再视频流的任一一段都可以解码播放 |
| 3 | FLV | Flash Video | FLV流媒体格式是随着Flash MX的推出发展而来的视频格式. |
| 4 | MP4 | MPEG-4 | MP4是一套用于音频、视频信息的压缩编码标准 |
| 5 | FFmpeg |  | FFmpeg是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序. |

**1.4参考资料**

**参考文档:**

1.《转码服务StreamTranscode产品用户使用说明书》；

2.《转码基础库StreamConvert需求说明书》；

3.《转码基础库StreamConvert概要设计说明书》；

**2模块说明**

**2.1模块设计**

**2.1.1．关键点分析**

这部分主要描述整个基础库里面最关键的一些设计，其中包括网络通信，线程模型，关键数据结构等内容，是整个模块最关键、最重要的设计部分；

**2.1.1.1 线程池模型**

本基础库里面的为了并发快速处理用于创建的任务，线程模型比较简单。主要是在

基础库加载后，通过接口设置初始化线程个数，这样就是一个数目固定的线程池。线程池中的每个线程维护一个任务队列，实质就是一个双向链表。该双向链表中持有分配给该线程的所有任务。

目前初始化线程数目建议是8倍的服务器CPU核心数。

新添加的任务如何分配给不同线程的策略是分配给线程池中目前负载最轻的线程，也就是线程持有任务队列数目最少的队列。

**2.1.1.2 任务调度策略**

为了让各个任务公平的被当前线程调度，同时为了进一步提高系统的并发性能和吞吐量。目前基础库提供两种调度策略。

一种是普通的调度策略，就是线程执行函数顺序轮询任务队列的每一个任务，这样保证了公平性，但是比较消耗CPU资源。

另外一种调度策略是：用一个整形值的32个bit位来标记是否当前数据缓存里面有数据，如果没有数据则当达到一定数目后，则再下次循环中少调度一次。这样做的目前是主要减少无效循环，提高CPU的使用率。

**2.1.2．设计约束**

这部分主要描述基础库里面的一些设计限制和能力集，在使用时需要注意程序本身设计上带来的限制。主要是为了在实际使用和测试时，能够做到心中有数。

**2.1.2.1 不同格式的码流格式转换**

目前除了RAW封装格式的流只外，其它格式的流目前都要转成raw之后再转从raw转成目标格式的码流。目前支持flv2ts、flv2flv flv2raw flv2mp4几种组合格式的转流。

**2.1.3．模块划分**

本封装库从层次上划分：

1. 主要分为接口层，该层主要用来用户调用，提供对外接口。
2. 中间层为任务处理和调度业务核心层，主要用来管理用户的创建的任务队列和线程调度执行该任务。任务执行时能够将用户的缓存数据读取进来，传输到最下层的转封装层进行数据的实际的转封装处理。
3. 最底层就是实际完成转封装的各个模块，该层将中间层的数据处理后，再回调给用

户上层。



图<1>:转封装库模块框架图

**2.3资源消耗**目前基础库消耗比较大的资源是CPU和内存资源，其它资源消耗有限；

**3接口定义**

**3.1模块外部接口说明**

此部分接口主要是用来外部调用的接口，外部使用者可以通过下面的接口完成一个转封装任务的创建。

**3.1.1 初始化**

|  |  |
| --- | --- |
| 函数名 | static IavStreamConvert\* Create(int threadnum = 16); |
| 功能描述 | 初始化转封装基础库，设置线程池的线程数量 |
| 输入参数 | Threadnum 可以由外部调用者指定线程数目 |
| 输出参数 | 无 |
| 返回值 | IavStreamConvert类对象指针 |

**3.1.2 任务创建**

|  |  |
| --- | --- |
| 函数名 | virtual intprtx jobCreate() = 0; |
| 功能描述 | 创建一路转封装任务 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 会话任务编号id |

本库重载了三种任务创建的接口，可以利用其它接口创建任务，省略后面的一些调用。具体请参考头文件和接口说明文档进行使用。

**3.1.3设置任务参数**

|  |  |
| --- | --- |
| 函数名 | virtual int jobConfigure(const intprtx jobID, trancodeConfigure\* pTransParam) = 0; |
| 功能描述 | 给该任务设置任务配置参数 |
| 输入参数 | 任务id：jobID  配置参数：pTransParam |
| 输出参数 | 暂无 |
| 返回值 | 0成功  -1失败 |

**3.1.4设置回调函数**

|  |  |
| --- | --- |
| 函数名 | virtual int jobRegisterAvCb(const intprtx jobID, ItransCallback\* cbImpl, void \*opaque) = 0; |
| 功能描述 | 给转封装模块设置回调函数，用于将目标数据回调给调用者 |
| 输入参数 | 任务id：jobID  回调函数指针：cbImpl  用户自定义数据：opaque |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**3.1.5 开始任务**

|  |  |
| --- | --- |
| 函数名 | virtual int jobStart(const intprtx jobID) = 0; |
| 功能描述 | 启动任务 |
| 输入参数 | 任务id：jobID |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**3.1.6 任务停止**

|  |  |
| --- | --- |
| 函数名 | virtual int jobStop(const intprtx jobID) = 0; |
| 功能描述 | 停止任务 |
| 输入参数 | 任务id：jobID |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**3.1.7 销毁任务**

|  |  |
| --- | --- |
| 函数名 | virtual int jobDelete(const intprtx jobID) = 0; |
| 功能描述 | 销毁任务，释放当前任务所占有的资源 |
| 输入参数 | 任务id：jobID |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**3.1.8输入码流**

|  |  |
| --- | --- |
| 函数名 | virtual int jobWrtieData(const intprtx jobID, const AV\_Buffer\* inData, CONVERT\_BUFFER\_INFO\* info) = 0; |
| 功能描述 | 将用户的码流输入进来，用于转封装模块进行转封装 |
| 输入参数 | 任务id：jobID  原始码流指针：inData  码流信息的描述：CONVERT\_BUFFER\_INFO |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

本基础库对外利用函数重载方式提供了三种输入，外部针对不同类型的原始码流利用不同的接口输入数据。具体请参考头文件和接口说明文档进行使用。

**4模块详细设计**

* 1. **线程池模块**

**4.1.1 模块类设计**

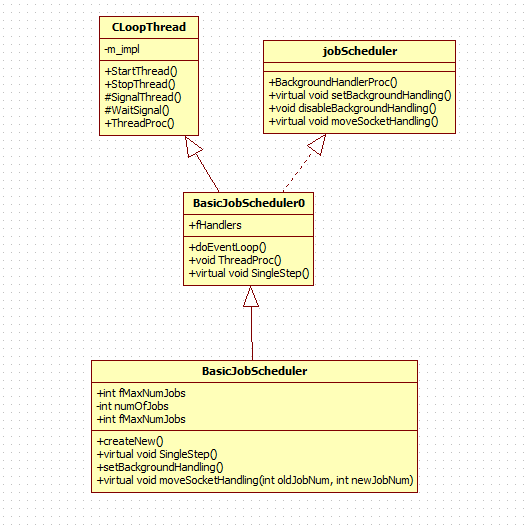
根据设计中模块的划分，设计类图如下:

CLoopThread类是线程类，定义了一个线程的所有操作，其中包括线程创建，销毁等操作；

JobScheduler类为调度基类，主要定义了任务的添加，任务删除，和任务编号替换等操作；

BasicJobSchedule0类同时继承了线程基类和调度基类，里面定义了调度算法的接口，可以在该层的线程函数里面，指定任务遍历的调度策略；

BasicJobScheduler类是真实调度策略和线程函数的实现类，在这里完成该线程对属于自己的任务队列的遍历和执行；



**图<4>:线程池和调度器类图**

**CLoopThread类中:**

m\_impl 结构体是所有线程信息，其中包括线程名称，线程栈大小，线程优先级，线程锁，循环处理函数，线程锁等信息；

StartThread() 启动线程函数

StopThread() 停止线程操作

ThreadProc() 线程执行函数

**jobScheduler类中:**

setBackgroundHandling()接口纯虚函数，定义了如何添加一个任务到任务队列；

disableBackgroundHandling()接口纯虚函数，定义了如何从任务队列里面删除一个具体任务；

moveSocketHandling()修改已经存在任务队列的任务编号，将老编号换成新编号；

**BasicJobScheduler0类中:**  
fHandlers 后台任务队列

fLastHandledJobNum 记录了上次任务发生时该任务在链表中的位置；

SingleStep()调度策略函数，其中通过函数重载方式，定义了两种调度策略，用于执行属于该线程的任务队列中的任务；

ThreadProc()线程执行函数，在该函数中执行调度策略函数，不同的传参，可以决定不同的调度策略，从而影响了线程真实执行任务队列中的操作行为；

**BasicJobScheduler类中**

Index 当前调度器编号;

numOfJobs 当前调度器的任务所有个数;  
fMaxNumJobs 目前调度器的任务队列里面最大的任务编号;

SingleStep() 调度策略函数的真实实现；

setBackgroundHandling() 添加任务到任务队列；

**4.1.2 模块接口说明**

**CLoopThread类中:**

**4.1.2.1线程基类构造函数**

|  |  |
| --- | --- |
| 函数名 | CLoopThread(int idx, int stacksize = 10 \* 1024 \* 1024, int priority = -1); |
| 功能描述 | 主要用来初始化线程的结构体成员，主要包括线程名，线程编号，线程锁，同步量，线程栈的大小； |
| 输入参数 | 线程编号id：idx  线程栈大小：stacksize  线程优先级：priority |
| 输出参数 | 无 |
| 返回值 | 无 |

**4.1.2.2线程启动函数**

|  |  |
| --- | --- |
| 函数名 | bool StartThread(); |
| 功能描述 | 创建线程，在线程中执行CLoopThread::Impl::ThreadProc函数，由实际线程对象进行实现线程函数 |
| 输入参数 | 线程编号id：idx  线程栈大小：stacksize  线程优先级：priority |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.1.2.3线程停止函数**

|  |  |
| --- | --- |
| 函数名 | bool StopThread (); |
| 功能描述 | 结束线程，等待线程退出，释放线程资源； |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**BasicJobScheduler0类中:**

**4.1.2.4线程执行函数**

|  |  |
| --- | --- |
| 函数名 | void ThreadProc() |
| 功能描述 | 该函数中调用实际调度器里面的执行策略函数，完成对任务的调度； |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**BasicJobScheduler类中：**

**4.1.2.5调度函数**

|  |  |
| --- | --- |
| 函数名 | Void SingleStep(unsigned maxDelayTime) |
| 功能描述 | 该函数实际完成对调度器的遍历，从而决定线程对调度队列的执行策略 |
| 输入参数 | 延时时间：maxDelayTime |
| 输出参数 | 无 |
| 返回值 | 无 |

**4.1.2.6调度函数**

|  |  |
| --- | --- |
| 函数名 | void setBackgroundHandling(int jobNum, int bEnable,BackgroundHandlerProc\* handlerProc, void\* clientData) |
| 功能描述 | 添加任务到任务队列 |
| 输入参数 | 任务编号：jobNum  使能开关：bEnable  任务处理函数：handlerProc  任务对象：clientData |
| 输出参数 | 无 |
| 返回值 | 无 |

**4.1.3 模块关键函数实现**

**任务调度算法分析：**

算法目的这边主要是如何减少无效循环次数，所以用一个int每一个bit位来标记是否读到数据，如果没有读到数据则用bit位来比较，则下次循环中调过对该任务的遍历，则降低了无效循环的次数。

**任务调度策略执行流程：**



**图<5>:任务调度策略流程图**

**4.1.4 示例代码**

N/A

* 1. **任务创建模块**
     1. **模块类设计**

根据设计中模块的划分，设计类图如下:

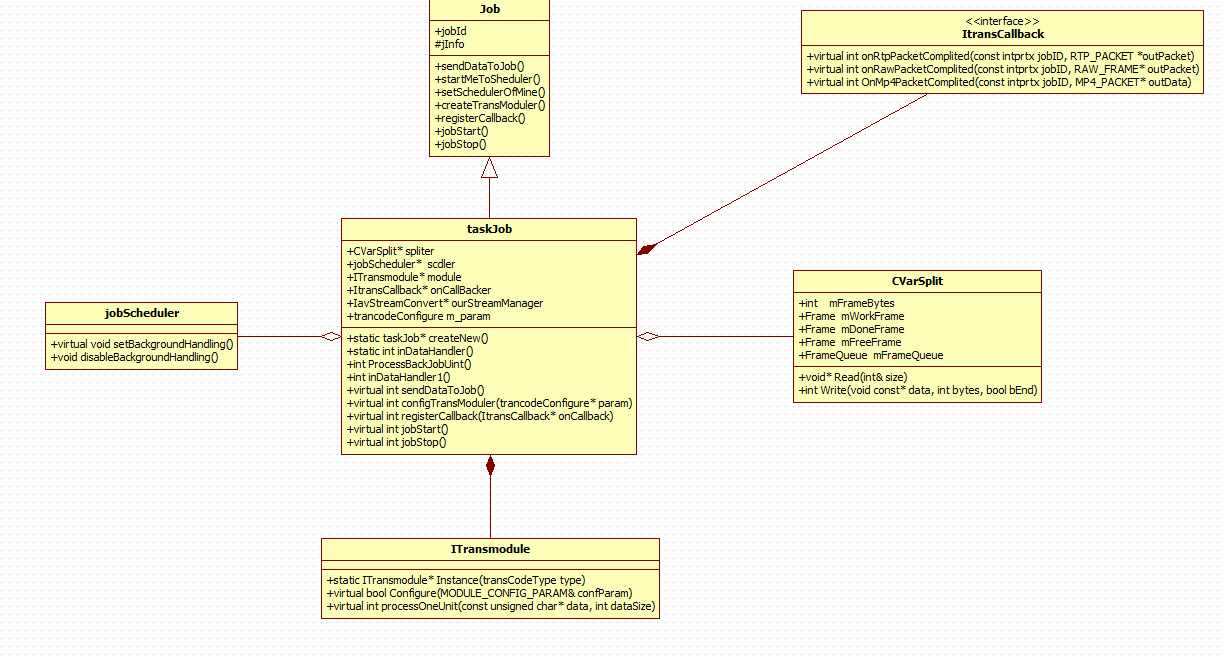
Job类是所有任务的基类，该类定义了一个转封装任务的所有公共属性和任务的工作操作，其中包括任务的创建，任务的生成，信息配置，任务的销毁等动作；

taskJob类是实际的任务类，包含了转封装的特殊信息，是对任务的具体化。

JobScheduler类为调度基类，主要定义了任务的添加，任务删除，和任务编号替换等操作，主要用来任务添加到任务队列里面；

ITransmodule类是下面各个转封装库的基类，针对上面不同的任务实例化成和该任务匹配的转封装模块；

CVarSplit类是数据缓存低于队列，任务源数据写进缓存，然后任务被线程调度后，将输入到转封装模块里面进行转换；

  
**图<6>:任务类图**

**Job类中：**

jobId 任务id，调用者创建任务都会生成一个任务编号；

jInfo 任务的结构体信息，保存当前任务id、任务优先级、任务执行时间、任务状态等；

* + 1. **模块接口说明**

**Job类中：**

**4.2.1.1写原始码流函数接口**

|  |  |
| --- | --- |
| 函数名 | virtual int sendDataToJob(const unsigned char\* exterData, int exterSize, int offSet,const unsigned char\* insideData, int insideSize, CONVERT\_BUFFER\_INFO\* bInfo) = 0 |
| 功能描述 | 任务接受调用者的源数据，然后调用该接口输入码流缓存； |
| 输入参数 | 待输入数据指针：exterData  待输入长度：exterSize  码流描述信息：bInfo |
| 输出参数 | 无 |
| 返回值 | 实际写进缓存的大小 |

**4.2.1.2实例化转封装对象接口**

|  |  |
| --- | --- |
| 函数名 | virtual int createTransModuler(trancodeConfigure\* param) = 0; |
| 功能描述 | 实例化转封装模块对象 |
| 输入参数 | 转码类型和该转码类型的配置参数：param |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.2.1.3实例化转封装对象接口**

|  |  |
| --- | --- |
| 函数名 | virtual int registerCallback(ItransCallback\* onCallback)= 0; |
| 功能描述 | 设置数据回调类，用于转封装模块将数据和消息回调给使用者 |
| 输入参数 | 回调函数类：onCallback |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.2.1.4开始任务**

|  |  |
| --- | --- |
| 函数名 | virtual int jobStart() = 0; |
| 功能描述 | 启动转封装任务 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.2.1.5停止任务**

|  |  |
| --- | --- |
| 函数名 | virtual int jobStop() = 0; |
| 功能描述 | 停止转封装任务； |
| 输入参数 | 回调函数类：onCallback |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**TaskJob类中：**

**4.2.1.6 任务执行函数**

|  |  |
| --- | --- |
| 函数名 | int inDataHandler1() |
| 功能描述 | 该函数是任务的具体执行动作，主要包括读数缓存原始码流数据和塞进转封装模块，进行处理 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 无数据或者转封装模块转失败：-1  成功：0 |

**4.2.1.7写原始码流到数据缓存**

|  |  |
| --- | --- |
| 函数名 | int sendDataToJob(const unsigned char\* exterData, int exterSize, int offSet,const unsigned char\* insideData, int insideSize, CONVERT\_BUFFER\_INFO\* bInfo) |
| 功能描述 | 将原始码流写进数据缓存 |
| 输入参数 | 待写数据指针：exterData  待写数据长度：exterSize  数据信息：bInfo |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.2.1.8实例化转封装模块对象**

|  |  |
| --- | --- |
| 函数名 | int createTransModuler(trancodeConfigure\* param) |
| 功能描述 | 实例生成转封装对象 |
| 输入参数 | 转码参数：param |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

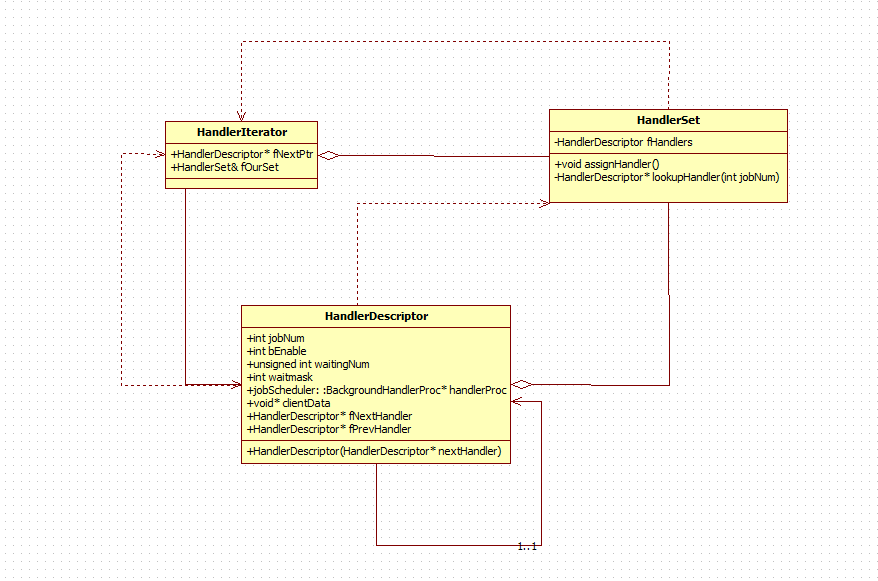
**4.2.1.8注册回调类**

|  |  |
| --- | --- |
| 函数名 | int registerCallback(ItransCallback\* onCallback) |
| 功能描述 | 向转封转模块注册回调类对象，便于装封装模块回调码流到调用者 |
| 输入参数 | 转码参数：param |
| 输出参数 | 无 |
| 返回值 | 0成功  -1失败 |

**4.2.3 模块关键函数实现  
4.2.4 示例代码**

* 1. **任务队列模块**
     1. **模块类设计**

根据设计中模块的划分，设计类图如下:



**图<7>:任务队列类图**

**HandlerDescriptor类中:**

jobNum 任务编号

bEnable 任务使能开关

waitingNum 记录目前多少次没有读到原始数据

waitmask 等待掩码，用整形32bit位上位0的个数标记那次没有读到数据

handlerProc 任务执行函数

clientData 任务实体

fNextHandler 任务节点后向指针

fPrevHandler 任务节点前向指针

**HandlerSet类中：**

fHandlers 任务实体节点，这是整个链表的头结点;

assignHandler() 添加任务到双向链表里面

clearHandler() 清除所有任务节点

lookupHandler() 根据任务编号找任务节点

**HandlerIterator类中：**

fOurSet 任务双向队列地址引用

fNextPtr 指向任务双向链表指针

next() 取得双向链表的下一个任务节点

reset() 复位，指针指向双向链表起始位置

* + 1. **模块接口说明**

**HandlerSet类中：**

**4.3.2.1添加任务**

|  |  |
| --- | --- |
| 函数名 | void HandlerSet::assignHandler(int jobNum, int bEnable,  jobScheduler::BackgroundHandlerProc\* handlerProc, void\* clientData) |
| 功能描述 | 将当前任务添加到任务队列 |
| 输入参数 | 任务编号：jobNum  任务使能开关：bEnable  任务执行动作：handlerProc  任务对象：clientData |
| 输出参数 | 无 |
| 返回值 | 无 |

**4.3.2.2删除任务**

|  |  |
| --- | --- |
| 函数名 | void clearHandler(int jobNum) |
| 功能描述 | 从任务队列中删除一个任务 |
| 输入参数 | 任务编号：jobNum |
| 输出参数 | 无 |
| 返回值 | 无 |

**HandlerDescriptor类中:**

**4.3.2.3查找任务**

|  |  |
| --- | --- |
| 函数名 | HandlerDescriptor\* lookupHandler(int jobNum) |
| 功能描述 | 根据任务编号查找一个任务 |
| 输入参数 | 任务编号：jobNum |
| 输出参数 | 无 |
| 返回值 | 任务节点指针 |

**HandlerIterator类中：**

**4.3.2.4遍历任务队列**

|  |  |
| --- | --- |
| 函数名 | HandlerDescriptor\* next() |
| 功能描述 | 找当前任务队列的下一个节点指针 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 当前任务节点的下一个节点指针 |

**4.3.2.5重置任务队列**

|  |  |
| --- | --- |
| 函数名 | HandlerDescriptor\* reset() |
| 功能描述 | 将任务队列复位，指针指向任务队列起始位置 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 返回值 | 任务队列第一个起始节点指针 |

**4.3.3 模块关键函数实现  
4.3.4 示例代码**

* 1. **数据缓存模块**
     1. **模块类设计**
     2. **模块接口说明**

**CVarSplit类中：**

**4.4.2.1写码流到数据缓存函数**

|  |  |
| --- | --- |
| 函数名 | int Write(void const\* data, int bytes, bool bEnd); |
| 功能描述 | 输入码流写入缓存； |
| 输入参数 | 待输入数据指针：data  待输入长度：bytes  是否写入数据结束：bEnd |
| 输出参数 | 无 |
| 返回值 | 实际写进缓存的大小 |

**4.4.2.2 从缓存读数据**

|  |  |
| --- | --- |
| 函数名 | void\* Read(int& size); |
| 功能描述 | 从缓存读取数据 |
| 输入参数 | 无 |
| 输出参数 | 实际读出数据的长度 |
| 返回值 |  |

* + 1. **模块关键函数实现**
    2. **示例代码**
  1. **转封装模块**
     1. **模块类设计**
     2. **模块接口说明**
     3. **模块关键函数实现**
     4. **示例代码**

**5 数据结构设计**

**5.1逻辑结构**

**5.1.1．jobStatus数据结构**

/\*任务状态\*/

enum jobStatus {

/\*任务未开始\*/

kJobNotStart = 1,

/\*任务已创建\*/

kJobCreated = 2,

/\*任务已配置\*/

kJobConfiged = 3,

/\*任务处理中\*/

kJobRuning = 4,

/\*任务已结束\*/

kJobStoped = 5,

/\*任务已删除\*/

kJobDeleted = 6,

/\*任务失败\*/

kJobFailed

};

**5.1.2．trancodeConfigure转码配置参数**

\*针对特定任务的转码配置参数\*/

typedef struct \_trancodeConfigure\_

{

transCodeType tranType; /\*转码类型\*/

RAW\_RTP\_PARAM raw2rtpParam; /\*raw转rtp配置参数\*/

RTP\_RAW\_PARAM rtp2RawParam; /\*rtp转raw配置参数\*/

RAW\_MP4\_PARAM raw2mp4Param; /\*raw转mp4配置参数\*/

HLS\_PARAM hlsMuxParam; /\*flv转hls配置参数\*/

RAW\_TIMEALBUM\_PARAM raw2timealbumParam; /\*raw转时光相册文件\*/

RAW\_TS\_PARAM raw2tsParam; /\*raw转ts文件配置参数\*/

RAW\_FLV\_PARAM raw2flvParam; /\*raw转flv配置参数\*/

TS\_MP4\_PARAM ts2mp4Param; /\*ts转mp4配置参数\*/

}trancodeConfigure;

其中 transType是转码类型，本库支持的转码类型有：

/\*转码类型\*/

enum transCodeType{

kFlv2raw = 0,

kHeavc2raw = 1,

kAvc2raw = 2,

kHevc2raw = 3,

kTs2raw = 4,

kPs2raw = 5,

kRtp2raw = 6,

kRaw2mp4 = 7,

kRaw2ts = 8,

kRaw2ps = 9,

kRaw2hls = 10,

kRaw2dash = 11,

kRaw2jpeg = 12,

kRaw2rtp = 13,

kRaw2TimeAlbum = 14, //raw转时光相册

kFlv2hls = 15,

kTs2Mp4 = 16,

kFlv2Flv = 17, //flv编码转换

kRaw2flv = 18,

kUndefined = 0xff

};

其中不同的转码任务配置参数如下：  
 typedef struct \_RAW\_MP4\_PARAM\_

{

MP4\_MODE mode; /\*转码模式\*/

unsigned int highLightRate; /\*用于抽帧模式，抽帧率，即隔几帧编码一帧的比率\*/

OSD\_LANGUGE osdLanguage; /\*用于OSD模式，OSD显示的语言\*/

unsigned long fileStartTime; /\*用于OSD模式，文件开始时间，单位ms\*/

VideoCodec videoCodecType; /\*视频编码类型\*/

AuidoCodec audioCodecType; /\*音频编码参数\*/

bool bPackAudio;

}RAW\_MP4\_PARAM;

typedef struct \_FLV\_HLS\_PARAM\_

{

unsigned int tsSliceDuration; /\*目标切片时长,单位秒\*/

AuidoCodec dstCodecType; /\*目标音频类型，默认为aac\*/

unsigned int liveM3u8listNum; /\*m3u8列表中live直播ts数目\*/

// unsigned int bSurpportVod; /\*是否支持点播\*/

char\* tsPreName; /\*ts文件前缀名，用于m3u8输出\*/

char\* tsPostName; /\*ts文件后缀名，用于m3u8输出\*/

}HLS\_PARAM;

typedef struct \_RAW\_TIMEALBUM\_PARAM\_

{

/\*时光相册播放速度: 0: 按帧本身时间播放，其它:帧之间毫秒时间间隔\*/

unsigned int playSpeed;

unsigned int fileType; /\*视频格式: 0,mp4 1,fmp4\*/

VideoCodec vCodec; /\*视频codec类型\*/

}RAW\_TIMEALBUM\_PARAM;

typedef struct \_VIDEO\_ENCODE\_PARAM\_

{

unsigned int Bitrate; /\*目标码率:bps\*/

unsigned int Qualit; /\*调整图像质量\*/

unsigned int IFrameInterval; /\*I帧间隔\*/

unsigned int Fps; /\*目标帧率\*/

unsigned int Width; /\*宽度\*/

unsigned int Height; /\*高度\*/

VideoCodec VideoType; /\*目标视频编码类型\*/

}VIDEO\_ENCODE\_PARAM;

typedef struct \_AUDIO\_ENCODE\_PARAM\_

{

unsigned int aBitrate; /\*目标音频码率: bps\*/

unsigned int aSamplerate; /\*目标音频采样率\*/

AuidoCodec aEncodeType; /\*目标音频编码类型\*/

}AUDIO\_ENCODE\_PARAM;

typedef struct \_REENCODE\_PARAM\_

{

bool bAudioRecode;

AUDIO\_ENCODE\_PARAM aEncodePram;

bool bVideoRecode;

VIDEO\_ENCODE\_PARAM vEncodePram;

}REENCODE\_PARAM;

typedef struct \_TS\_MP4\_PARAM\_

{

bool bReencode; /\*是否重新编码\*/

REENCODE\_PARAM rencParam; /\*如果需要重新编码，则设置编码参数\*/

RAW\_MP4\_PARAM modeParam; /\*MP4转码模式\*/

}TS\_MP4\_PARAM;

typedef struct \_TS\_MUXER\_PARAM\_

{

bool bSegmented;

int segmentLength;

}TS\_MUXER\_PARAM;

typedef struct \_RAW\_TS\_PARAM\_

{

TS\_MUXER\_PARAM param;

}RAW\_TS\_PARAM;

typedef struct \_RAW\_FLV\_PARAM\_

{

int reserved;

VideoCodec videoCodecType; /\*视频编码类型\*/

AuidoCodec audioCodecType; /\*音频编码参数\*/

bool bPackAudio;

}RAW\_FLV\_PARAM;

**5.1.3．trancodeConfigure转码配置参数**

typedef struct AV\_Buffer {

const unsigned char \*data;

/\*\*\* Size of data in bytes.\*/

int dataSize;

/\*\*\* Size of buffer lenth in bytes.\*/

int bufSize;

} AV\_Buffer;

typedef struct

{

int talSzie;

int usedSize;

int leftSize;

} CONVERT\_BUFFER\_INFO;

**5.2物理结构**

**N/A**

**5.3数据结构和程序关系**

在5.1定义的数据结构分别在下面的模块中用到，如下面表格所示：

**表1-1:数据结构和模块引用对照表**

|  |  |  |
| --- | --- | --- |
| **编号** | **数据结构** | **相关模块** |
| 5.1.1 | jobStatus | 任务模块 |
| 5.1.2 | trancodeConfigure | 任务模块、转封装模块 |
| 5.1.3 | AV\_Buffe | 缓存模块、任务模块 |
| 5.1.4 | CONVERT\_BUFFER\_INFO | 缓存模块、任务模块 |

**6 特定需求**

**6.1安全**

**6.2 性能**

**6.3 可用性**

**6.4易用性**

**6.4.1 异常处理**

单个转封装任务的处理异常不会导致整个模块出现问题，当基础库捕获到这个异常后，则会打印相应的日志进行记录。

**6.5可测试性**

**6.6可移植性**

**6.7 可维护性**

**6.8 兼容性**

**6.9 系统**

**6.9.1 Linux**

目前支持linux64位版本

**6.9.2 Windows**

N/A

**7 外部依赖**

**8 改动/影响**

本次改动影响如下：

1. 以前线程池的任务调度策略有失公平性，导致所有任务不能公平调度。现在增加了一种调度策略，后续需要分析，进一步提高性能和吞吐量。

**9 测试建议**

**9.1**功能测试

**9**.2 整合测试

**10 当前问题**

**11 潜在问题**

1. 目前基础库缺少日志支持，给问题排查和代码调试带来困难；

2. 支持转封装能力还不是很强，有些转换组合目前不支持；

3. 缺少全链路ID支持，出现问题后，不便于跟踪程序执行路径；

4. 线程池调度策略还可以继续优化，有浪费CPU的情况；

5. ts原始码流转时，下面会针对每一路任务都创建线程，会造成线程膨胀，可以用线程池机制来优化；

6.代码中还存在大量的无效代码和冗余代码，有些变量和函数命名不规范，可以去掉和重新命名，给可读性带来困难；

7.缺少错误码，对于出现问题，上层无法感知到，应该统一全局错误码，出现问题应该返回具体的错误码