SDK集成文档

iOS集成文档

一、SDK导入

1. 将AlgorithmModel.zip解压后拽到工程中，勾选“Copy items if needed”和“Create Groups”，选择对应的Target。
2. 工程中展开“AlgorithmModel->lib”，右键lib，show in founder，把lib中所有的文件拖拽到工程配置BuildPhase->Link Binary中。
3. Build Phase->Link Binary额外添加以下系统库：

libbz2.tbd

libiconv.tbd

VideoToolbox.framework

AudioToolbox.framework

CoreMedia.framework

1. 工程中BuildSettings->Other C++ Flags添加 -std=c++14
2. 将工程目录“AlgorithmModel->Resource”下的“rec\_int8.mnn”文件拖拽到BuildPhase->Copy Bundle Resources下。

二、SDK接入

参考EdfuVideoHandler工程->ViewControoler->Process方法。

1. import头文件

#import "video\_processor\_interface.h"

#import "common.h"

1. 将引入头文件的.m文件改为.mm，使其能编译C++，其他步骤参考ViewControoler->Process方法。

Android集成文档

使用方式

1. 初始化 ProcessTask

|  |
| --- |
| ProcessTask task = new ProcessTask(context, config, callback); |

|  |
| --- |
| public ProcessTask(@NonNull Context context, @NonNull ProcessConfig config, @NonNull ProcessCallback callback) {  this.context = context;  this.config = config;  this.callback = callback;  File outputDir = new File(config.outputPath);  if (!outputDir.exists()) {  outputDir.mkdirs();  }  } |

1. context：上下文
2. config：
   1. inputVideoPath，待处理视频文件路径
   2. outputPath，输出路径，如果是SD卡记得申请权限
   3. taskId，任务id，由前端Uri 传递过来
   4. outputWidth，输出视频分辨率宽，由前端Uri 传递过来
   5. outputHeight，输出视频分辨率高，由前端Uri 传递过来
   6. idx，传空字符串即可
   7. layoutType，由前端Uri 传递过来
   8. source，传空字符串即可
3. 开启线程执行task

|  |
| --- |
| Executors.newSingleThreadExecutor().execute(task); |

1. 监听任务执行过程

|  |
| --- |
| public interface ProcessCallback {  /\*\*  \* @param frameFinished 已处理完的帧数  \* @param progress 处理进度  \*/  void progress(int frameFinished, float progress);  /\*\*  \* @param success 处理结果  \*/  void finish(boolean success);  } |

C++ SDK接口文档

一、接口定义

**common.h**

|  |
| --- |
| #pragma once  #include <string>  namespace vak {  enum Status { kSuccess = 0, kFailed };  // frame\_num: number of frames that has been processed  // percent: procedure percent of current task  using Callback = std::function<void(int /\*frame\_num\*/, float /\*percent\*/)>;  } // namespace vak |

**video\_processor\_interface.h**

|  |
| --- |
| #pragma once  #include <string>  #include "common.h"  namespace vak {  class \_\_attribute\_\_((visibility("default"))) VideoProcessorInterface {  public:  static VideoProcessorInterface \*getInstance();  virtual ~VideoProcessorInterface() {}  // model\_path: path of model file  // input\_video\_path: path of input .mp4 video file  // output\_path: directory to store output files  // uuid: a unique task id  // path of output mp4 is: ${output\_path}/${uuid}.mp4  // path of output zip is: ${output\_path}/${uuid}.zip  // uuid is composed by ${user\_id}\_${timestamp}  // user\_id: a unique id of user  // timestamp: a program generated timestamp  // output\_width: width of output video  // output\_height: height of output video  // idx:  // layout\_type: [1-8]  // source:  virtual Status open(const std::string &model\_path, const std::string &input\_video\_path,  const std::string &output\_path, const std::string &uuid,  int output\_width, int output\_height, const std::string &idx,  int layout\_type, const std::string &source) = 0;  virtual Status process() = 0;  // cb: a callback function that report the progress of current task  virtual Status process(Callback cb) = 0;  // close must be called after the call of process is finish  virtual Status close() = 0;  };  } // namespace vak |

二、使用示例

|  |
| --- |
| void simple\_cb(int frame\_num, float percent) {  printf("processed: %d frames, percent: %.2f%%\n", frame\_num, percent);  }  std::string user\_id = "123";  auto timestamp = std::time(nullptr);  std::string uuid = user\_id + "\_" + std::to\_string(timestamp);  int output\_width = 370, output\_height = 800;  std::string idx = "1";  int layout\_type = 2;  std::string source = "test";  auto processor = std::shared\_ptr<vak::VideoProcessorInterface>(  vak::VideoProcessorInterface::getInstance());  if (processor->open(model\_path, input\_video\_file, output\_dir, uuid, output\_width,  output\_height, idx, layout\_type, source) != vak::kSuccess) {  printf("failed at open\n");  return 1;  }  if (processor->process(simple\_cb) != vak::kSuccess) {  printf("failed at process\n");  return 2;  }  if (processor->close() != vak::kSuccess) {  printf("failed at close\n");  return 3;  } |

注：

1. uuid由用户ID与时间戳拼接而成
2. idx, layout\_type以及source均是从服务器下发的信息中获取
3. output\_width固定为370，output\_height根据输入的视频的真实宽高等比缩放而来，output\_width和output\_height都需要是2的整数倍
4. 对于每个任务，依次调用open, process和close这三个方法，当某个方法的返回值不为vak::kSuccess时，需要做错误处理，并终止此任务的执行
5. 当有多个任务时，应将所有任务放进一个队列中，依次串行执行（即执行完某个任务再开始下一个任务），任务应在创建的线程中执行，以免阻塞主线程
6. process方法接受一个回调函数，它会在每处理完一帧的时候被执行一次，它的两个参数分别是已经处理的视频帧数和已经执行完的百分比