Technology Tarena 达内科技

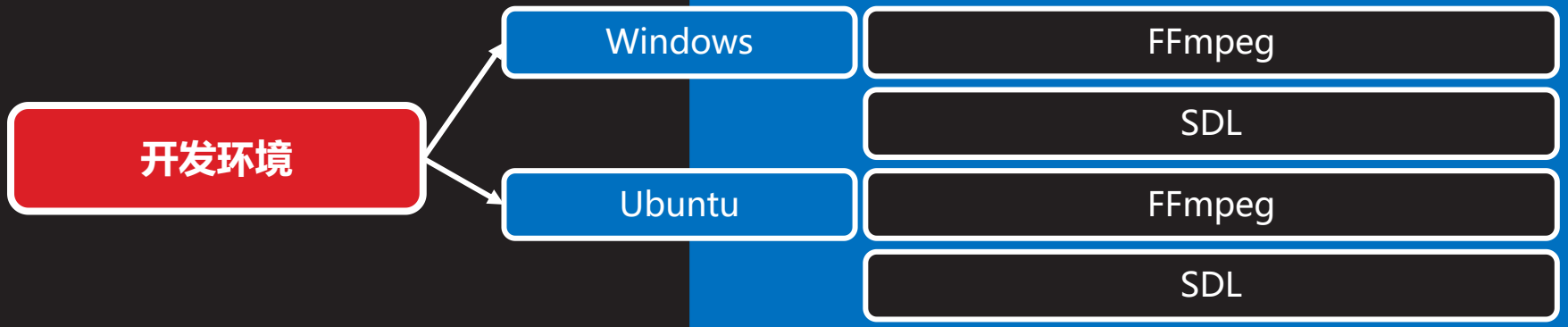
流媒体高级编程

STREAMING MEDIA DAY02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	开发环境
	10:30 ~ 11:20	
	11:30 ~ 12:20	流媒体应用
下午	14:00 ~ 14:50	视频流播放
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

开发环境



Technology Tarena 达内科技

Windows

FFmpeg

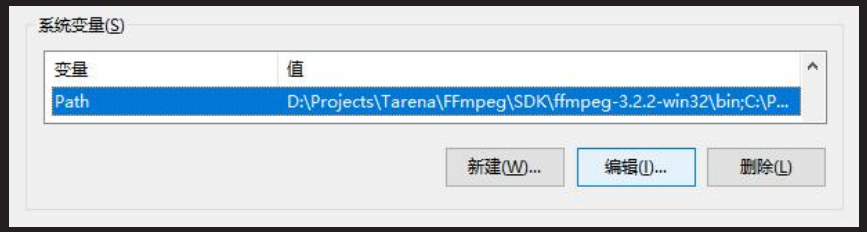
- 下载开发包
  - http://ffmpeg.zeranoe.com/builds/win32/dev/ffmpeg-3.2.2-win32-dev.zip
  - 解压到ffmpeg-3.2.2-win32目录下
- 下载运行库
  - http://ffmpeg.zeranoe.com/builds/win32/shared/ffmpeg-3.2.2-win32-shared.zip
  - 解压并将其中的bin和presets子目录复制到ffmpeg-3.2.2-win32目录下

FFmpeg(续1)

Technology Tarena 达内科技

知识讲解

- ffmpeg-3.2.2-win32目录结构
  - bin/ // 动态库及可执行程序
    - +- avdevice-57.dll // 音视频设备函数动态库
    - +- avcodec-57.dll // 音视频编解码函数动态库
    - +- avfilter-6.dll // 音视频过滤器函数动态库
    - +- avformat-57.dll // 音视频格式函数动态库
    - +- avutil-55.dll // 音视频工具函数动态库
    - +- swresample-2.dll // 音频重采样及采样格式转换函数动态库
    - +- swscale-4.dll // 图像缩放及颜色空间转换函数动态库
    - +- postproc-54.dll // 后期效果处理函数动态库
    - +- ffmpeg.exe // 流媒体采集、转换、传输、存储工具
    - +- ffplay.exe // 流媒体播放器
    - +- ffprobe.exe // 流媒体格式查看器
  - 将bin目录添加到系统PATH环境变量中

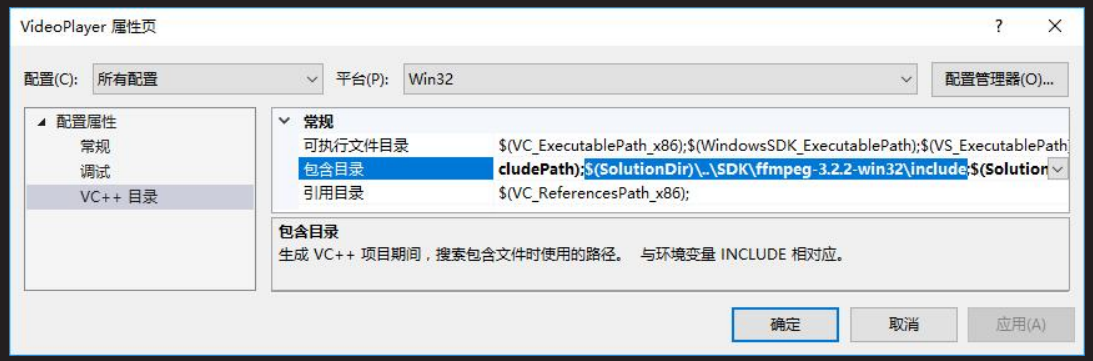


FFmpeg(续2)

Technology Tarena 达内科技

知识讲解

- ffmpeg-3.2.2-win32目录结构
  - include/ // 头文件
    - +- libavdevice/ // 音视频设备函数头文件
    - +- libavcodec/ // 音视频编解码函数头文件
    - +- libavfilter/ // 音视频过滤器函数头文件
    - +- libavformat/ // 音视频格式函数头文件
    - +- libavutil/ // 音视频工具函数头文件
    - +- libswresample/ // 音频重采样及采样格式转换函数头文件
    - +- libswscale/ // 图像缩放及颜色空间转换函数头文件
    - +- libpostproc/ // 后期效果处理函数头文件
  - 将include目录添加到"工程属性/VC++目录/包含目录"中

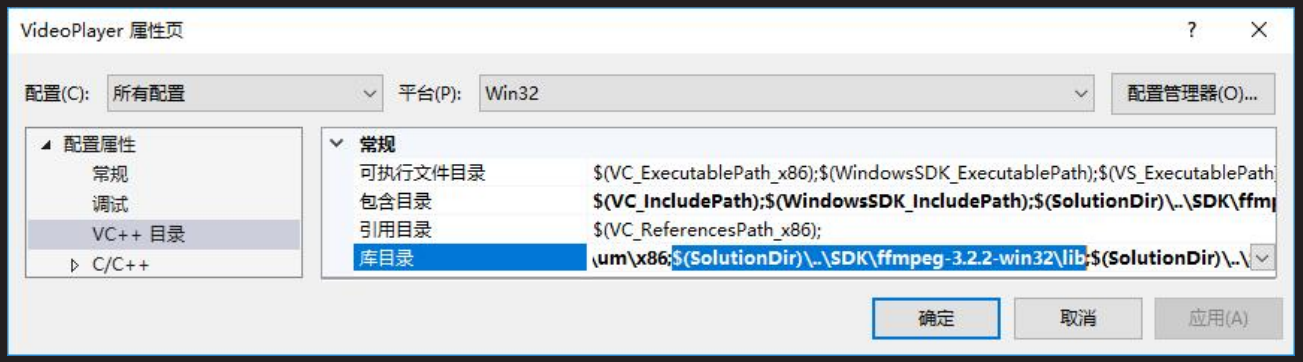


FFmpeg(续3)

Technology Tarena 达内科技

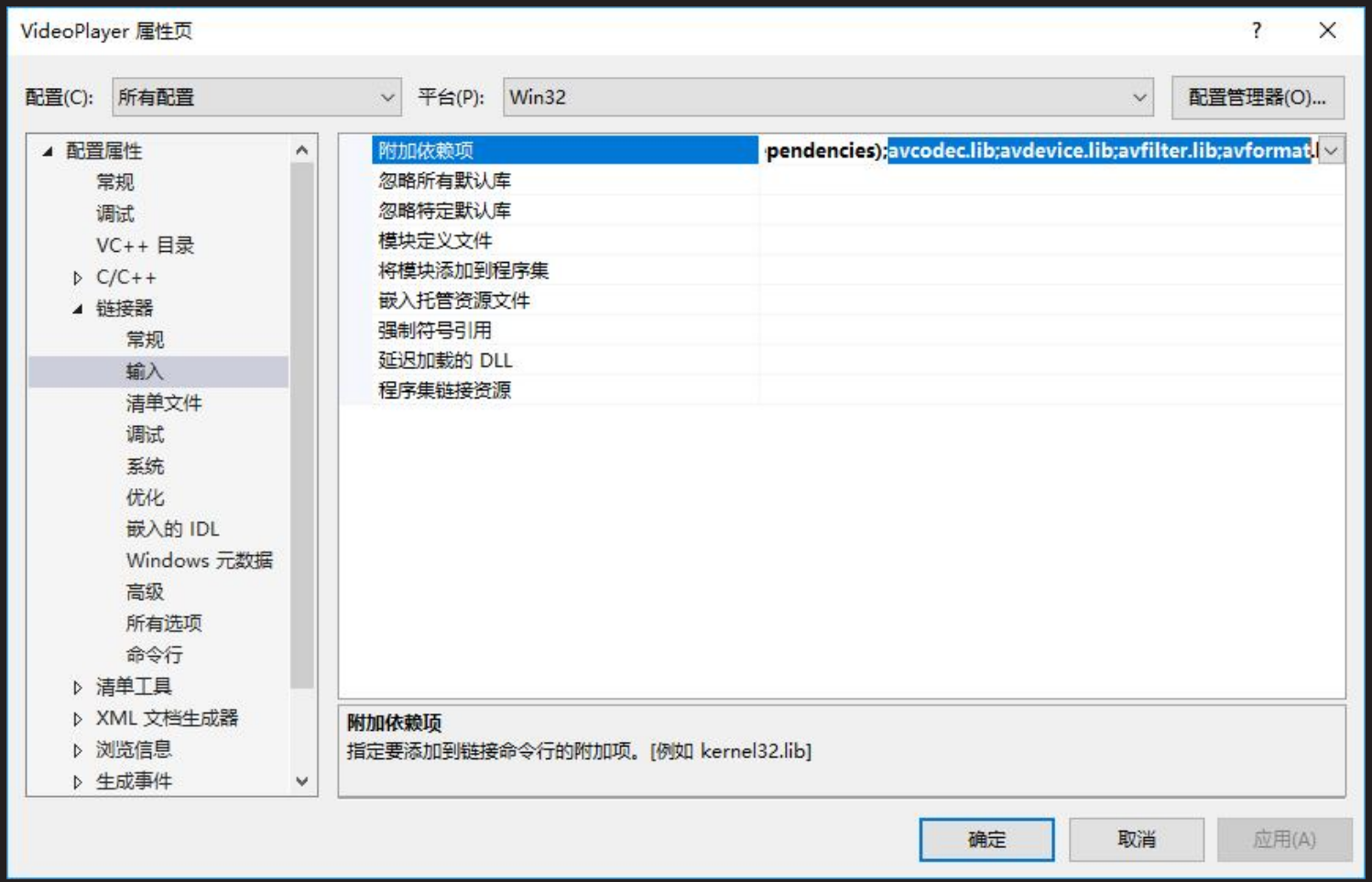
知识讲解

- ffmpeg-3.2.2-win32目录结构
  - lib/ // 导入库
    - +- avdevice.lib // 音视频设备函数导入库
    - +- avcodec.lib // 音视频编解码函数导入库
    - +- avfilter.lib // 音视频过滤器函数导入库
    - +- avformat.lib // 音视频格式函数导入库
    - +- avutil.lib // 音视频工具函数导入库
    - +- swresample.lib // 音频重采样及采样格式转换函数导入库
    - +- swscale.lib // 图像缩放及颜色空间转换函数导入库
    - +- postproc.lib // 后期效果处理函数导入库
  - 将lib目录添加到"工程属性/VC++目录/库目录"中



FFmpeg(续4)

- ffmpeg-3.2.2-win32目录结构
  - 将以上库添加到"工程属性/链接器/输入/附加依赖项"中



知识讲解

SDL

- SDL (Simple DirectMedia Layer)是一个开源、跨平台的多媒体开发库，提供了多种与音视频处理有关的功能
- 下载开发包和运行库
  - http://www.libsdl.org/release/SDL2-devel-2.0.5-VC.zip
  - 解压到SDL2-2.0.5目录下
- 配置环境变量和工程属性
  - 将SDL2-2.0.5/lib/x86目录添加到系统PATH环境变量中
  - 将SDL2-2.0.5/include目录添加到"工程属性/VC++目录/包含目录"中
  - 将SDL2-2.0.5/lib/x86目录添加到"工程属性/VC++目录/库目录"中
  - 将SDL2.lib和SDL2main.lib添加到"工程属性/链接器/输入/附加依赖项"中

Technology Tarena 达内科技

Ubuntu

FFmpeg

- 安装ffmpeg依赖
  - sudo apt-get install yasm
    sudo apt-get install libx264-dev
    sudo apt-get install libfaac-dev
    sudo apt-get install libmp3lame-dev
    sudo apt-get install libtheora-dev
    sudo apt-get install libvorbis-dev
    sudo apt-get install libxvidcore-dev
    sudo apt-get install libxext-dev
    sudo apt-get install libxfixes-dev

FFmpeg(续1)

Tarena Technology 达内科技

- 下载源代码
  - https://github.com/FFmpeg/FFmpeg/archive/master.zip
  - 解压到FFmpeg-master目录下
- 编译源代码
  - 在FFmpeg-master目录下执行：
    ./configure --prefix=/usr/local/ffmpeg --enable-shared --enable-gpl --enable-version3 --enable-nonfree --enable-postproc --enable-pthreads --enable-libmp3lame --enable-libtheora --enable-libx264 --enable-libxvid --enable-libvorbis --enable-indev=alsa --enable-outdev=alsa
  - make

知识讲解

FFmpeg(续2)

- 安装开发包和运行库
  - sudo make install
- 配置共享库路径
  - sudo vi /etc/ld.so.conf.d/ffmpeg.conf
    加入：/usr/local/ffmpeg/lib
    sudo ldconfig
- Makefile中的编译和链接选项
  - -I/usr/local/ffmpeg/include
  - -L/usr/local/ffmpeg/lib
  - -lavdevice -lavformat -lavcodec -lswscale -lswresample -lavfilter -lpostproc -lavutil

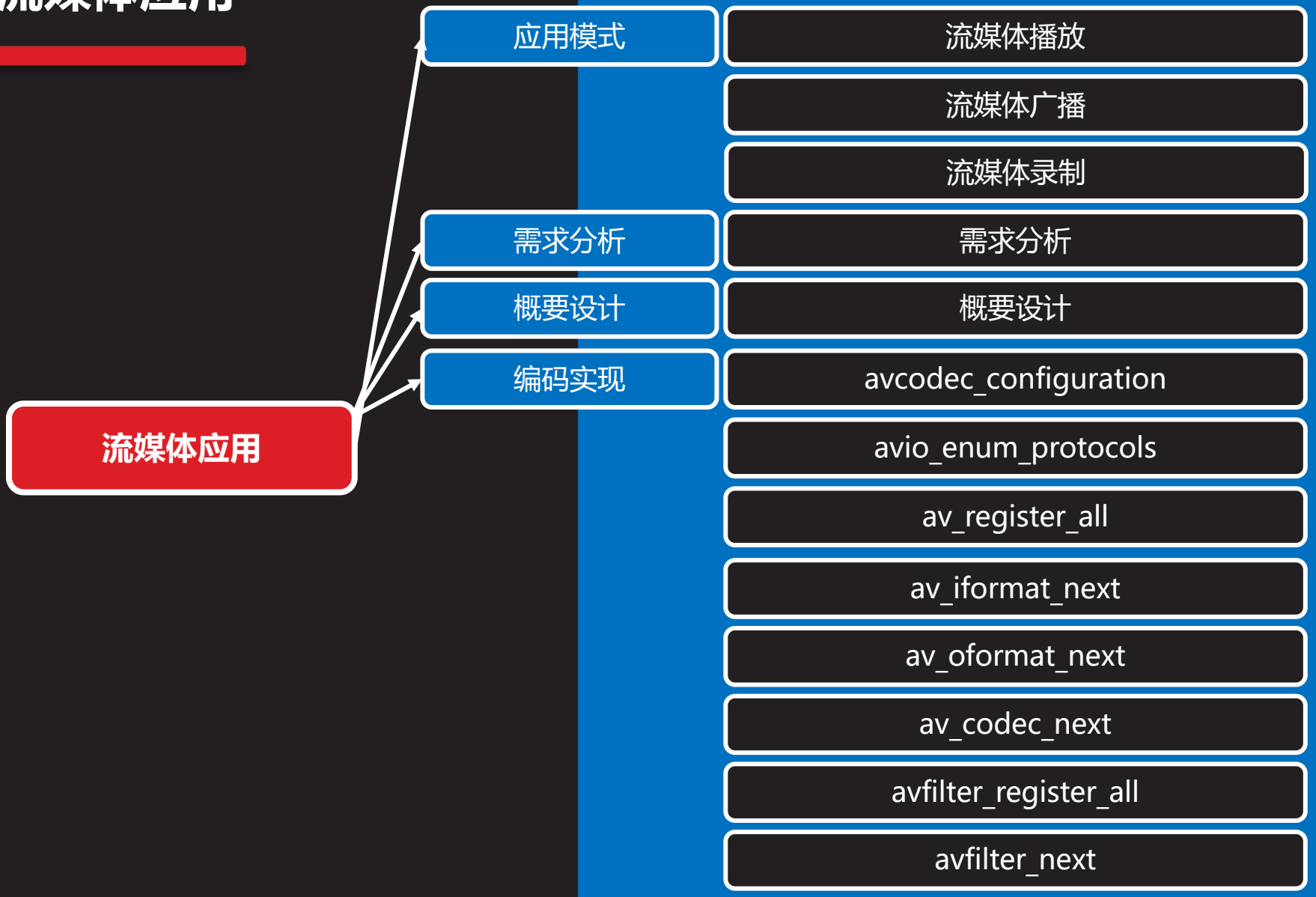
知识讲解

SDL

- 安装开发包和运行库
  - sudo apt-get install libsdl2-dev
- Makefile中的编译和链接选项
  - -I/usr/include/SDL2
  - -lSDL2 -lSDL2main -ldl -lpthread

知识讲解

流媒体应用


流媒体应用
应用模式
需求分析
概要设计
编码实现
流媒体播放
流媒体广播
流媒体录制
需求分析
概要设计
avcodec_configuration
avio_enum_protocols
av_register_all
av_iformat_next
av_oformat_next
av_codec_next
avfilter_register_all
avfilter_next

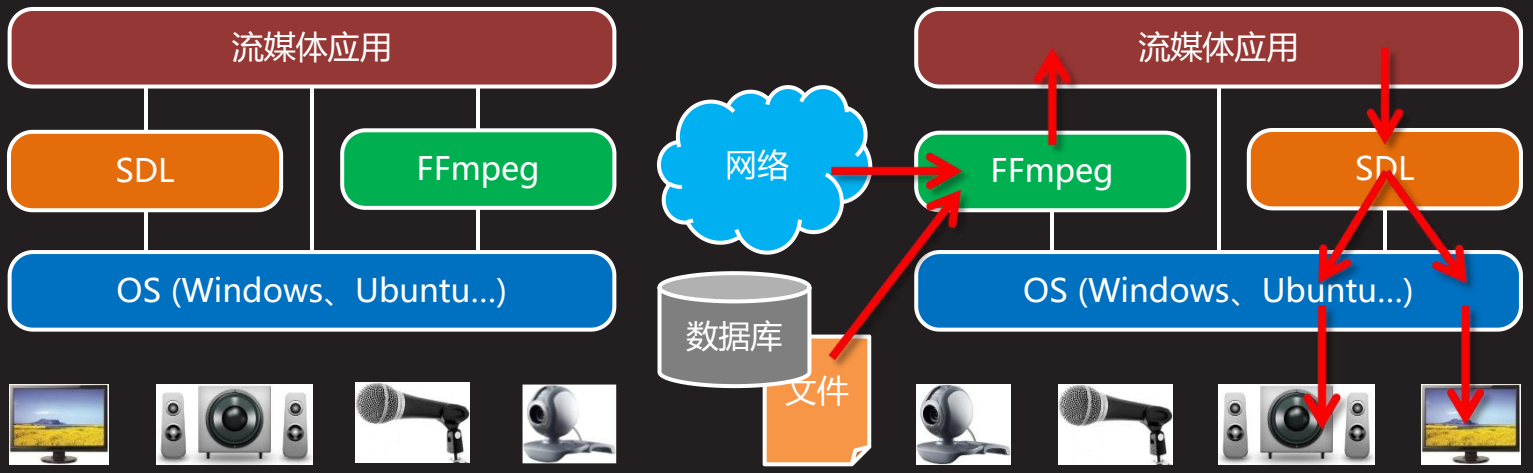
Technology Tarena 达内科技

应用模式

流媒体播放

- 流媒体应用通过FFmpeg从网络或本地数据库及文件中读取编码流，经解码后交由SDL渲染和回放
  - 媒体播放器
  - 点播客户端
  - 直播粉丝端

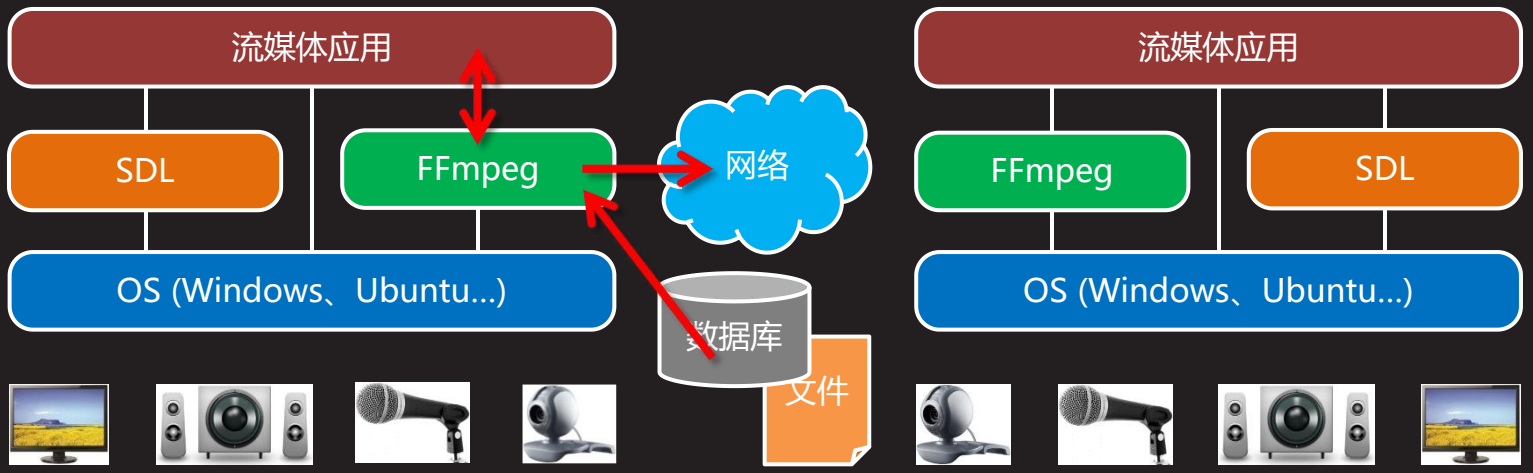
知识讲解



流媒体广播

- 流媒体应用通过FFmpeg读取本地数据库或文件中的编码流，经解码并重编码后发送到网络
  - 网络电视台
  - 格式转码器
  - 广告插播器

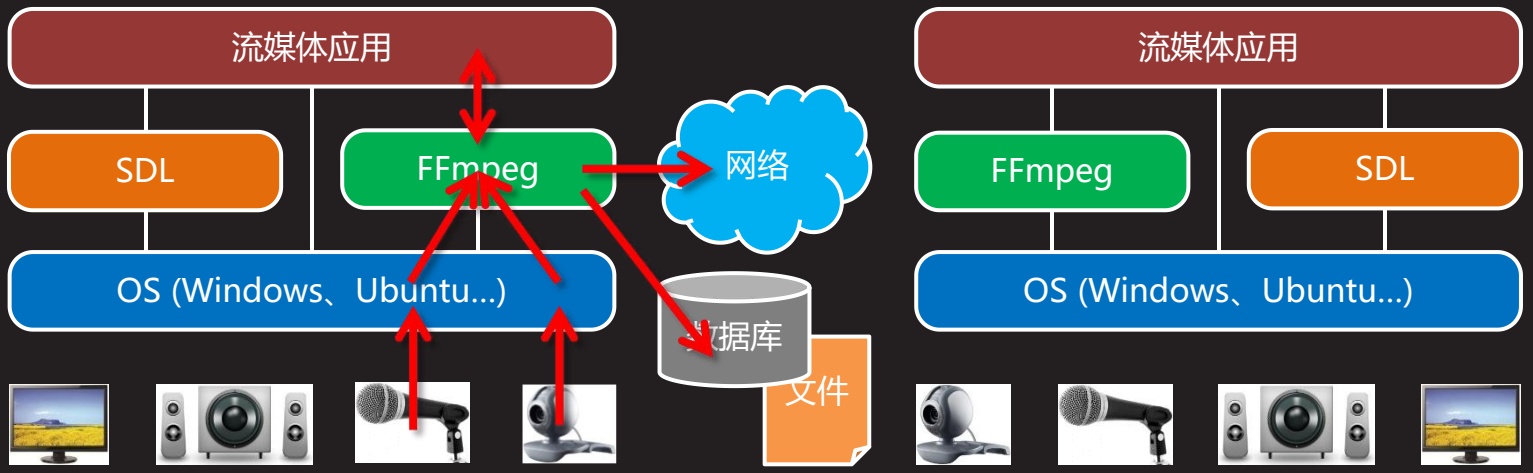
知识讲解



流媒体录制

- 流媒体应用通过FFmpeg读取来自摄像头和麦克风的音视频编码流，经解码并重编码后保存到本地数据库或文件中，也可以通过网络发送至远程
  - 数字录像机
  - 监控采集端
  - 直播主播端

知识讲解



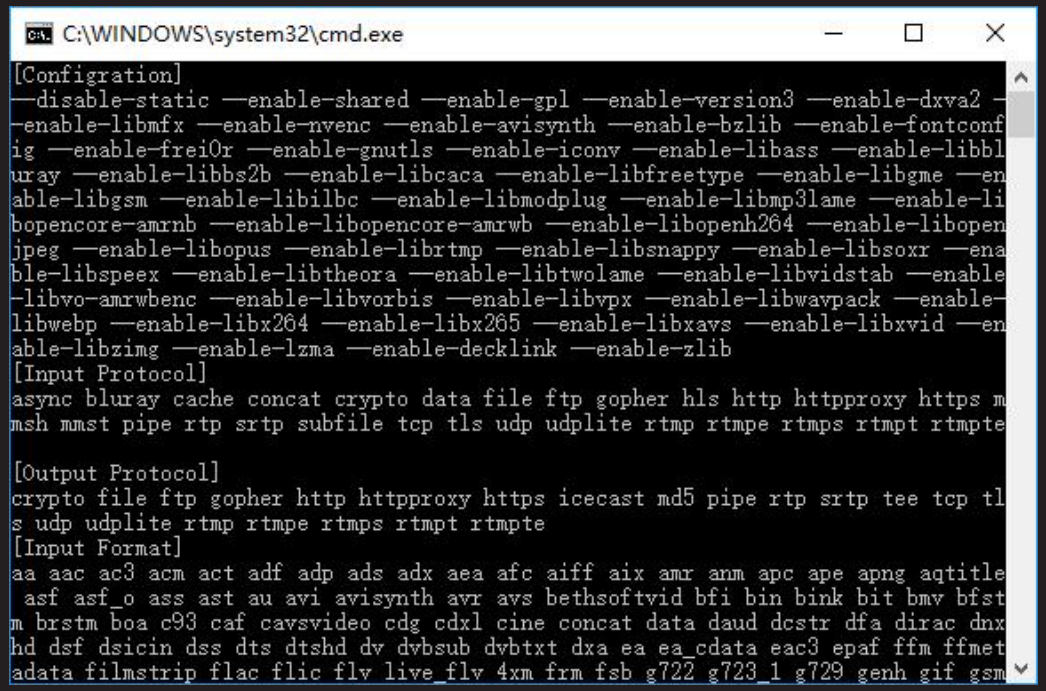
Technology Tarena 达内科技

需求分析

需求分析

- 打印FFmpeg如下五个方面的信息：
  - 当前配置
  - 输入/输出协议
  - 输入/输出格式
  - 音/视频编/解码器
  - 过滤器

知识讲解

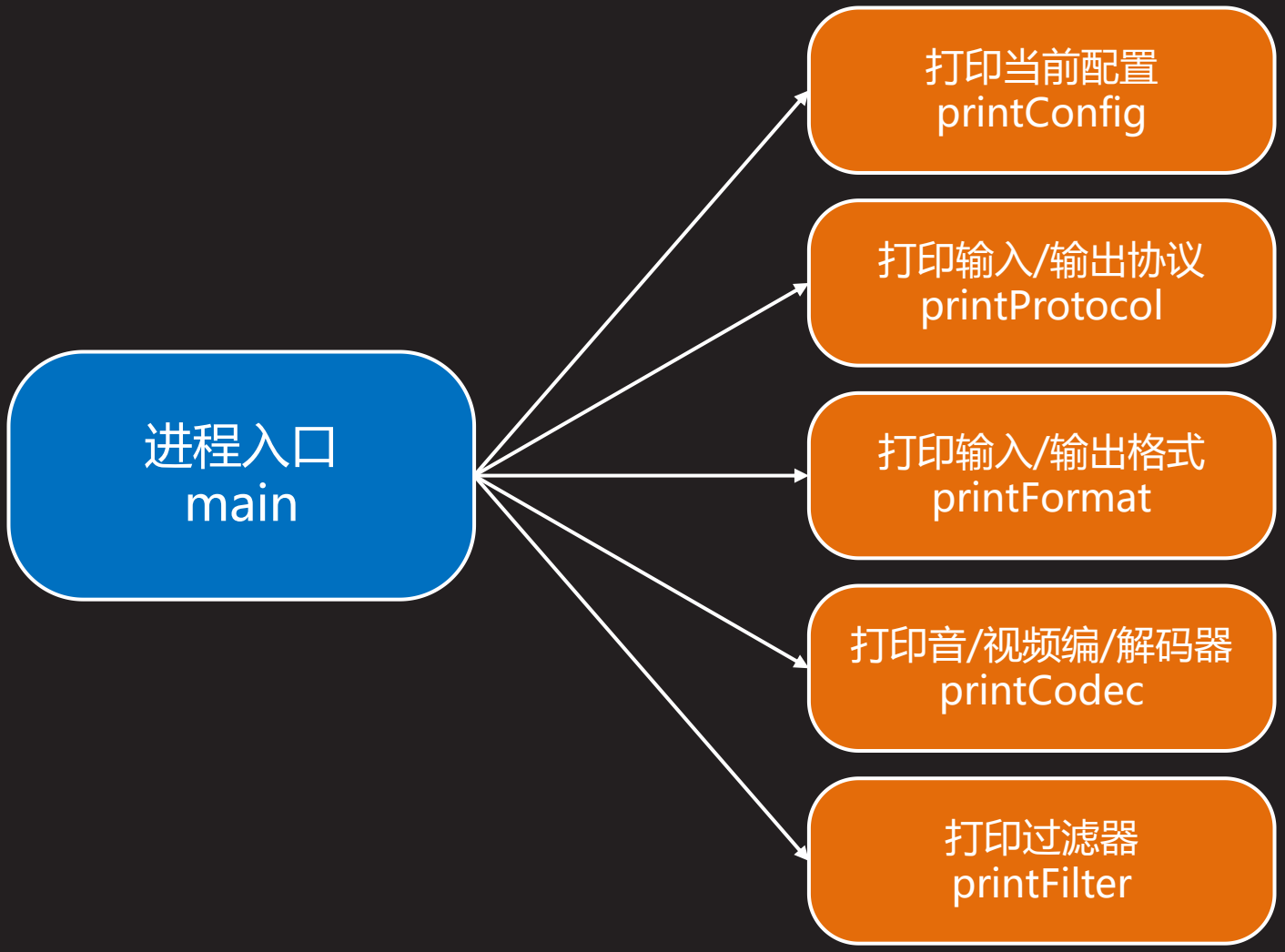


Technology Tarena 达内科技

概要设计

概要设计

知识讲解



Technology Tarena 达内科技

编码实现

avcodec_configuration

- 获取libavcodec库的构建配置
  - `#include <libavcodec/avcodec.h>`
    `const char* avcodec_configuration (void);`
  - 返回libavcodec库的构建配置字符串，如：
    --disable-static --enable-shared --enable-gpl ...
  - 例如：
    `printf ("%s\n", avcodec_configuration ());`
    或
    `cout << avcodec_configuration () << endl;`

知识讲解

avio_enum_protocols

- 迭代当前可用协议
  - `#include <libavformat/avio.h>`

```
const char* avio_enum_protocols (
    void** opaque,   // URLProtocol类型的二级
                     // 指针，首次调用传入NULL，
                     // 随迭代更新，当其再度为
                     // NULL时，终止迭代
    int    output);  // 0 - 输入协议，1 - 输出协议
```

  - 返回协议名字符串
  - 例如：

```
for (struct URLProtocol* up = NULL;;) {
  char const* name = avio_enum_protocols ((void**)&up, 0);
  if (!up) break;
  cout << name << endl; }
```


av_register_all

- 初始化libavformat库并注册所有打包器、解包器和协议
  - `#include <libavformat/avformat.h>`

    `void av_register_all (void);`
  - 在使用任何打包器、解包器、编码器、解码器之前，必须先调用此函数，否则将导致失败

知识讲解

av_iformat_next

- 获取第一个或下一个输入格式
  - `#include <libavformat/avformat.h>`
    `AVInputFormat* av_iformat_next (`
    `  const AVInputFormat* f); //` 输入格式指针
  - 若参数f为NULL，则返回第一个输入格式，否则返回f的下一个输入格式，若f指向最后一个输入格式，则返回NULL
  - 例如：

```
for (AVInputFormat* fmt =
  av_iformat_next (NULL); fmt; fmt =
  av_iformat_next (fmt))
  cout << fmt->name << endl;
```

知识讲解

av_oformat_next

- 获取第一个或下一个输出格式
  - `#include <libavformat/avformat.h>`
    `AVOutputFormat* av_oformat_next (`
    `const AVOutputFormat* f); //` 输出格式指针
  - 若参数f为NULL，则返回第一个输出格式，否则返回f的下一个输出格式，若f指向最后一个输出格式，则返回NULL
  - 例如：

```
for (AVOutputFormat* fmt =
  av_oformat_next (NULL); fmt; fmt =
  av_oformat_next (fmt))
  cout << fmt->name << endl;
```

知识讲解

av_codec_next

- 获取第一个或下一个编解码器
  - `#include <libavcodec/avcodec.h>`

    `AVCodec* av_codec_next (`

    `const AVCodec* c); // 编解码器指针`
  - 若参数c为NULL，则返回第一个编解码器，否则返回c的下一个编解码器，若c指向最后一个编解码器，则返回NULL
  - 可以根据AVCodec中的type成员判断是何种编解码器，根据decode成员判断是编码器还是解码器
  - 例如：

```
for (AVCodec* codec = av_codec_next (NULL); codec;
  codec = av_codec_next (codec))
  if (codec->type == AVMEDIA_TYPE_AUDIO && !codec->decode)
    cout << "Audio Encoder: " << codec->name << endl;
```

知识讲解

avfilter_register_all

- 初始化过滤器系统，注册所有内置过滤器
  - `#include <libavfilter/avfilter.h>`
    `void avfilter_register_all (void);`
  - 在使用任何内置过滤器之前，必须先调用此函数，否则将导致失败

知识讲解

avfilter_next

- 获取第一个或下一个过滤器
  - `#include <libavfilter/avfilter.h>`

    ```
    const AVFilter* avfilter_next (
      const AVFilter* prev); // 过滤器指针
    ```
  - 若参数prev为NULL，则返回第一个过滤器，否则返回prev的下一个过滤器，若prev指向最后一个过滤器，则返回NULL
  - 例如：

    ```
    for (AVFilter const* filter =
      avfilter_next (NULL); filter; filter =
      avfilter_next (filter))
      cout << filter->name << endl;
    ```

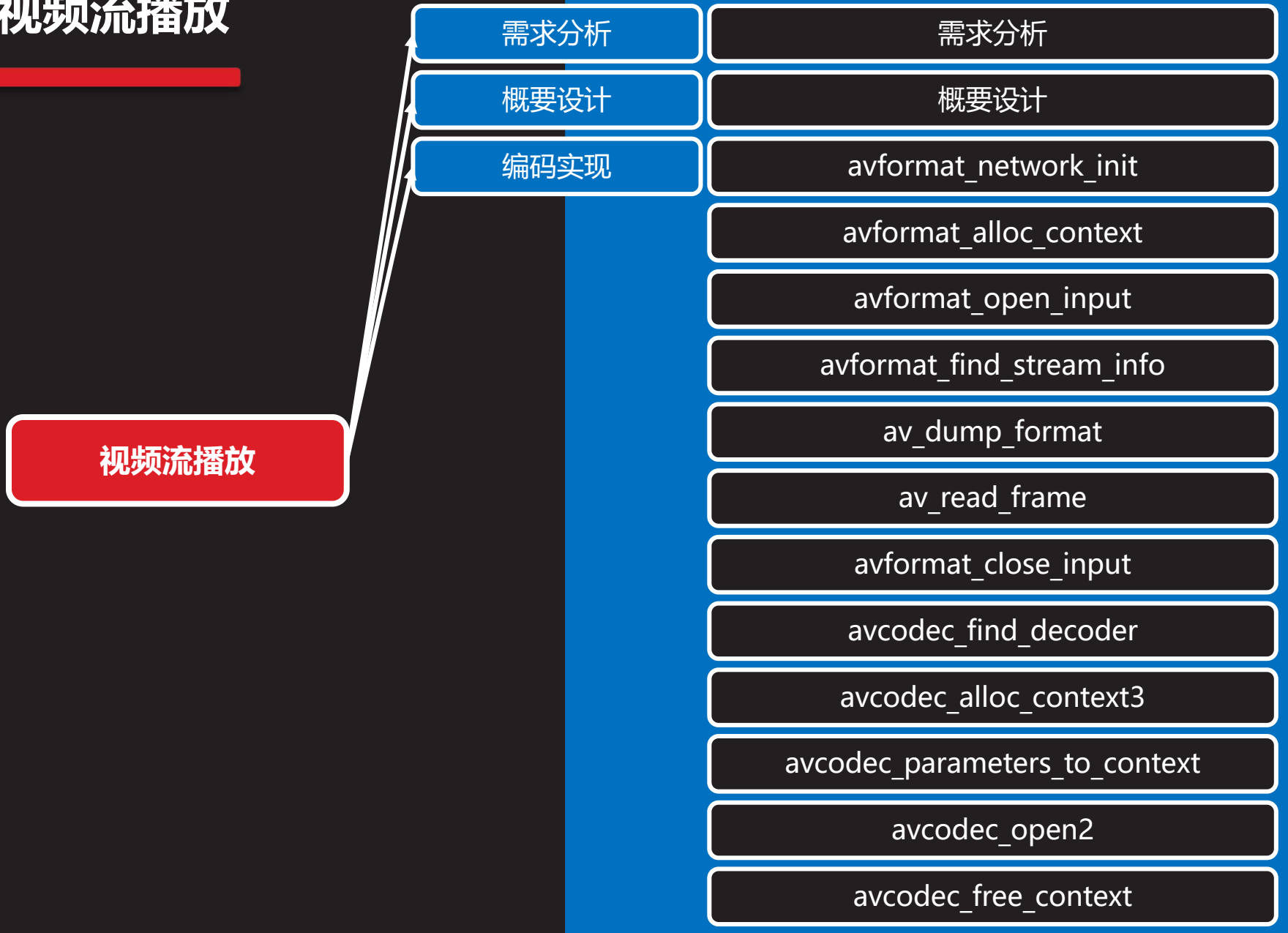

HelloFFmpeg

【参见：FFmpeg/Primer/HelloFFmpeg】

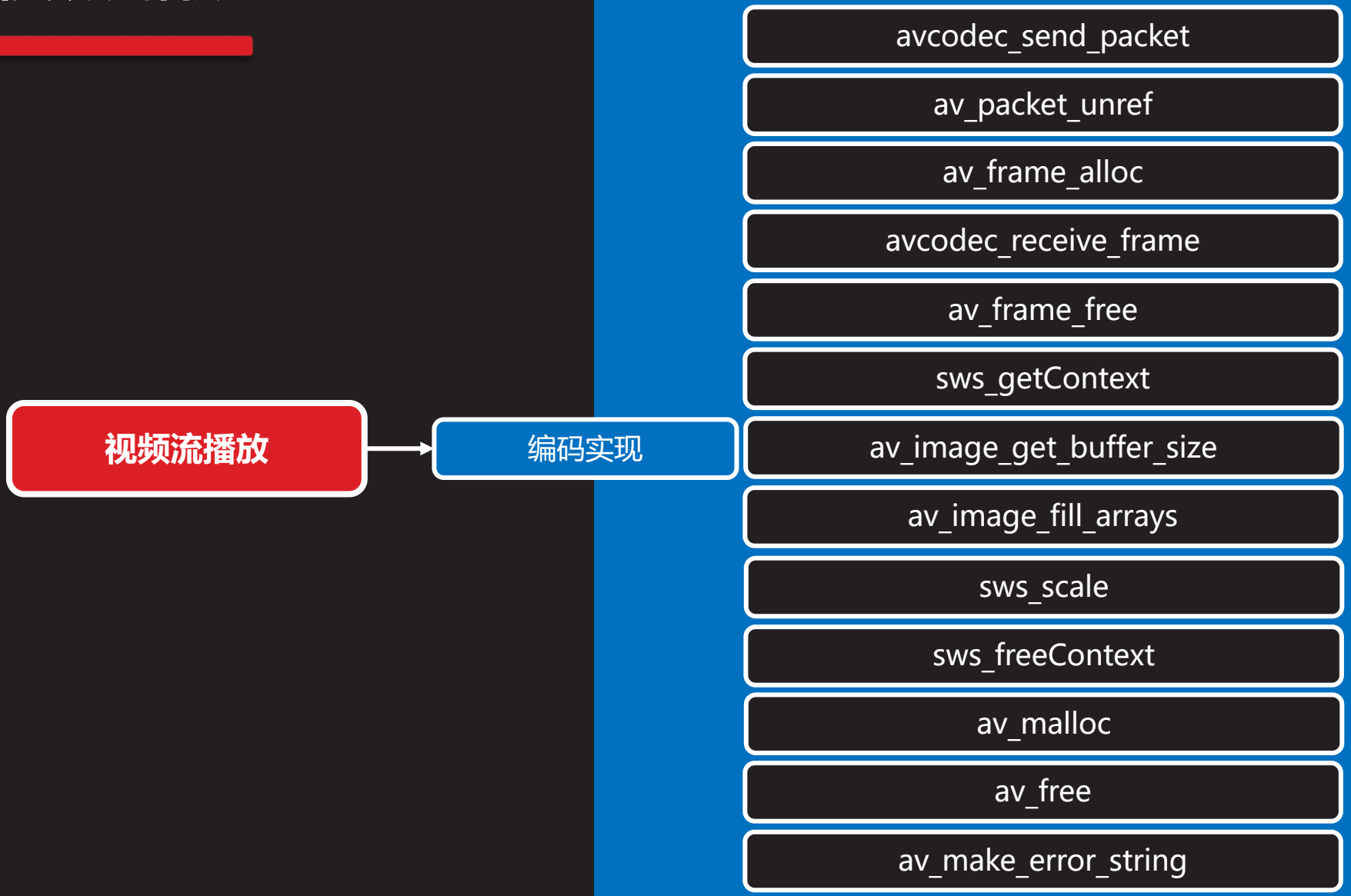
课堂练习

- 打印FFmpeg如下五个方面的信息：
  - 当前配置
  - 输入/输出协议
  - 输入/输出格式
  - 音/视频编/解码器
  - 过滤器

视频流播放


视频流播放
需求分析
概要设计
编码实现
需求分析
概要设计
avformat_network_init
avformat_alloc_context
avformat_open_input
avformat_find_stream_info
av_dump_format
av_read_frame
avformat_close_input
avcodec_find_decoder
avcodec_alloc_context3
avcodec_parameters_to_context
avcodec_open2
avcodec_free_context

视频流播放


视频流播放
编码实现
avcodec_send_packet
av_packet_unref
av_frame_alloc
avcodec_receive_frame
av_frame_free
sws_getContext
av_image_get_buffer_size
av_image_fill_arrays
sws_scale
sws_freeContext
av_malloc
av_free
av_make_error_string

视频流播放

- SDL_Init
- SDL_Quit
- SDL_CreateWindow
- SDL_DestroyWindow
- SDL_CreateRenderer
- SDL_RenderCopy
- SDL_RenderPresent
- SDL_DestroyRenderer
- SDL_CreateTexture
- SDL_UpdateYUVTexture
- SDL_DestroyTexture
- SDL_WaitEventTimeout
- SDL_Delay
- SDL_GetError

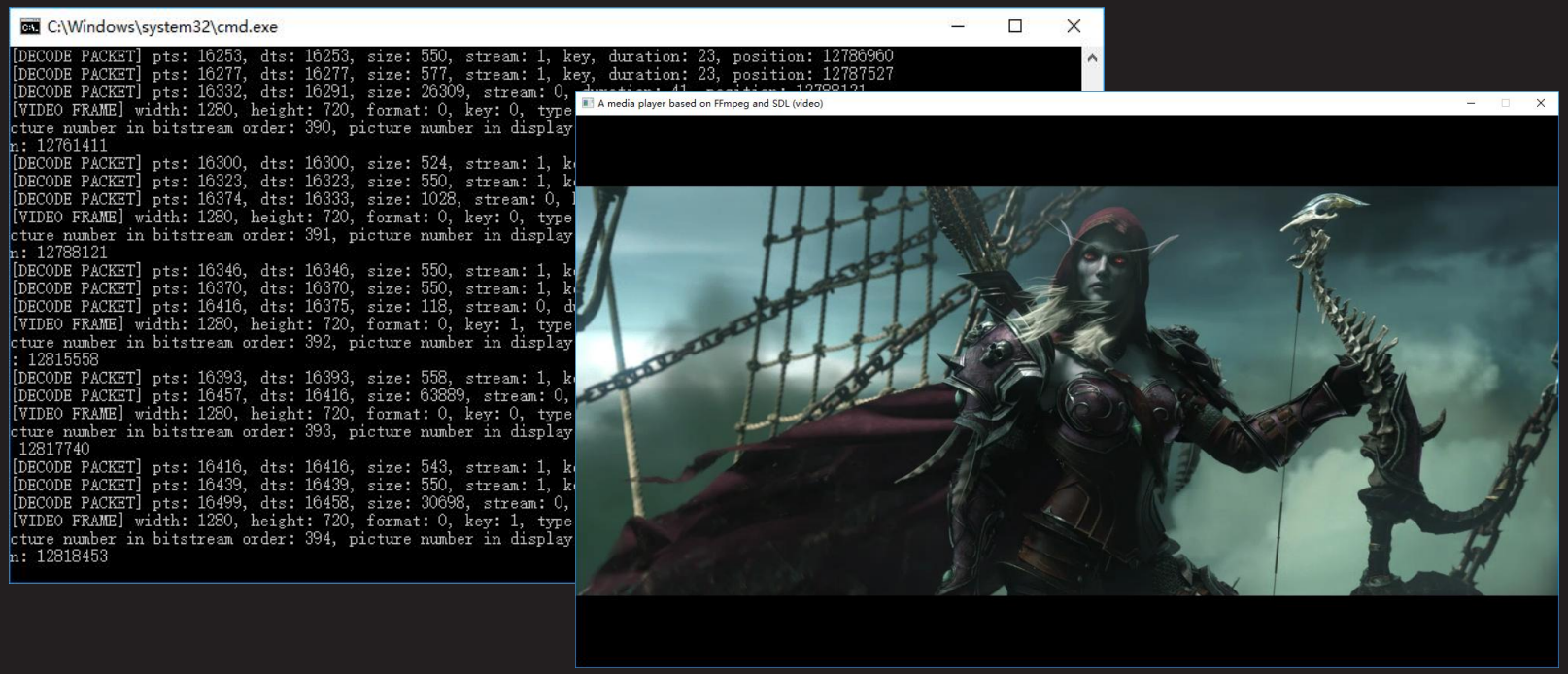
Technology Tarena 达内科技

需求分析

需求分析

- 播放视频流
  - 本地播放：VideoPlayer mayitbe.vob
  - 视频点播：VideoPlayer rtmp://192.168.1.166/vod/1.mp4
  - 接收直播：VideoPlayer rtmp://192.168.1.166/live/1

知识讲解



Technology Tarena 达内科技

概要设计

概要设计

知识讲解

进程入口
main

播放
play

播放视频
playVideo

打印源格式信息
printSrcFmt

打印解码包信息
printDecPkt

打印视频帧信息
printVidFrm

打印FFmpeg错误
ffmpegError

打印SDL错误
sdlError

事件处理
doEvent

Technology Tarena 达内科技

编码实现

avformat_network_init

- 初始化网络组件
  - `#include <libavformat/avformat.h>`
    `int avformat_network_init (void);`
  - 成功返回0，失败返回错误码(<0)
  - 在使用任何网络协议之前，必须先调用此函数，否则将导致失败

知识讲解

avformat_alloc_context

- 创建源格式上下文(AVFormatContext)对象
  - `#include <libavformat/avformat.h>`
    `AVFormatContext* avformat_alloc_context ( void);`
  - 成功返回源格式上下文对象指针，失败返回NULL
  - 所创建的源格式上下文对象，在使用完以后，必须通过avformat_close_input函数销毁
- AVFormatContext类型的源格式上下文对象，可以理解为是对流媒体源的抽象。它既可以表示一个本地输入文件，也可以表示一个远程推流应用，甚至可以表示一个流媒体采集设备，如摄像头、麦克风等

知识讲解

avformat_open_input

- 打开输入流并读取其头部信息，保存到源格式上下文中
  - #include <libavformat/avformat.h>

```
int avformat_open_input (
    AVFormatContext** ps,          // 源格式上下文
    const char*       url,         // 源统一资源定位符
    AVInputFormat*    fmt,         // 输入格式，
                                   // 取NULL自动检测
    AVDictionary**    options);    // 特殊选项，
                                   // 取NULL不设选项
```

  - 成功返回0，失败返回错误码(<0)
  - 被打开的输入流，在使用完以后，必须通过avformat_close_input函数销毁

知识讲解

avformat_find_stream_info

- 在输入流中查找流信息，保存到源格式上下文中
  -
```
#include <libavformat/avformat.h>
int avformat_find_stream_info (
  AVFormatContext* ic,        // 源格式上下文
  AVDictionary**   options); // 特殊选项，
                              // 取NULL不设选项
```
  - 成功返回0，失败返回错误码(<0)
  - 不是所有的流媒体格式都带有头部，比如MPEG，该函数可以通过对流媒体数据包的分析，自动获取有关流的信息
  - 此函数并不会改变流媒体资源的当前读取位置

知识讲解

av_dump_format

- 打印格式信息
  - `#include <libavformat/avformat.h>`

```
void av_dump_format (
    AVFormatContext* ic,          // 格式上下文
    int              index,       // 流索引
    const char*      url,         // 统一资源定位符
    int              is_output);  // 0 - 输入流,
                                  // 1 - 输出流
```

知识讲解

av_read_frame

- 从输入流中读取数据包
  -
```
#include <libavformat/avformat.h>
int av_read_frame (
    AVFormatContext* s,     // 源格式上下文
    AVPacket*        pkt); // 数据包
```
  - 成功返回0，失败或读完返回错误码(<0)
  - 不验证数据的有效性，填充于帧间的无效数据也会被读出
  - 对于视频流，一个数据包恰好包含一个视频帧
    对于音频流，一个数据包可能包含若干音频帧(PCM/ADPCM)，也可能只包含一个音频帧(MPEG)
  - 被读出的数据包，在使用完以后，必须通过av_packet_unref函数销毁

知识讲解

avformat_close_input

- 关闭处于打开状态的源格式上下文，销毁源格式上下文对象本身及其所有内容，同时将指向该对象的指针置空
  -
```
#include <libavformat/avformat.h>
void avformat_close_input (
    AVFormatContext** s); // 源格式上下文
```
  - 该函数的参数为指向某个已被打开的源格式上下文对象的指针的地址，待函数返回后，该对象的指针即变为空指针

知识讲解

avcodec_find_decoder

- 根据解码器ID查找解码器对象
  - `#include <libavcodec/avcodec.h>`

    `AVCodec* avcodec_find_decoder (`

    `enum AVCodecID id); // 解码器ID`
  - 成功返回与参数解码器ID相对应的解码器对象指针，失败返回NULL
  - `enum AVCodecID {`

    `...`

    `AV_CODEC_ID_PNG, // PNG图像`

    `...`

    `AV_CODEC_ID_MPEG2VIDEO, // DVD视频`

    `...`

    `AV_CODEC_ID_MP3, // MP3音频`

    `...`

    `};`

知识讲解

avcodec_alloc_context3

- 创建与指定编解码器相对应的编解码器上下文(AVCodecContext)对象，并将该对象初始化为缺省状态
  - `#include <libavcodec/avcodec.h>`

    `AVCodecContext* avcodec_alloc_context3 (`

    `const AVCodec* codec); // 编解码器`
  - 成功返回指向与参数编解码器相对应的编解码器上下文对象的指针，失败返回NULL
  - 所创建的编解码器对象，在使用完以后，必须通过avcodec_free_context函数销毁
- AVCodecContext类型的编解码器上下文对象，可以理解为是对编解码器运行环境的抽象。它负责记录、跟踪编解码器的工作过程，并实时反映其状态及参数的变化

avcodec_parameters_to_context

- 根据编解码器参数设置编解码器上下文
  - ```
    #include <libavcodec/avcodec.h>
    int avcodec_parameters_to_context (
      AVCodecContext*           codec, // 编解码器上下文
      const AVCodecParameters* par);  // 编解码器参数
    ```
  - 成功返回0，失败返回错误码(<0)
  - 用于设置编解码器上下文的编解码器参数，通常来自于格式上下文中的流对象
    struct AVFormatContext { ... AVStream** streams; ... };
    struct AVStream { ... AVCodecParameters* codecpar; ... };

知识讲解


    ```

avcodec_open2

- 根据编解码器的当前状态设置编解码器上下文
  -
```
#include <libavcodec/avcodec.h>
int avcodec_open2 (
    AVCodecContext* avctx,     // 编解码器上下文
    const AVCodec*  codec,     // 编解码器
    AVDictionary**  options); // 特殊选项,
                               // 取NULL不设选项
```
  - 成功返回0，失败返回错误码(<0)
  - 第一个参数所指向的编解码器上下文对象，必须是由avcodec_alloc_context3函数所创建的
  - 此函数不是线程安全的，在多线程中使用需要互斥保护

avcodec_free_context

- 销毁编解码器上下文对象及其所关联的一切，同时将指向该对象的指针置空
  - `#include <libavcodec/avcodec.h>`

    `void avcodec_free_context (`

    `  AVCodecContext** avctx); //` 编解码器上下文
  - 该函数的参数为指向某个已创建的编解码器上下文对象的指针的地址，待函数返回后，该对象的指针即变为空指针

知识讲解

avcodec_send_packet

- 向解码器发送待解码数据包
  - `#include <libavcodec/avcodec.h>`

```
int avcodec_send_packet (
    AVCodecContext* avctx,  // 解码器上下文
    const AVPacket* avpkt); // 待解码数据包
```

  - 成功返回0，失败返回错误码(<0)
  - 该函数在对数据包进行解码的过程中，会同步更新解码器上下文的状态，以反映解码工作的进程，同时影响对后续数据包的解码处理，待解码数据包在此过程中不会被修改
  - 以NULL指针作为第二个参数，向解码器发送空数据包，表示刷流操作的开始，解码器将在后续avcodec_receive_frame调用中给出缓冲区中残存的尾帧

知识讲解

av_packet_unref

- 解除编解码包对其数据缓冲区的引用，并将该编解码包重置为缺省状态
  - `#include <libavcodec/avcodec.h>`

    `void av_packet_unref (`

    `  AVPacket* pkt); // 编解码包`
  - 编解码包中的数据只要不再使用，即应通过此函数解除对其缓冲区的引用，并复位至缺省状态，准备引用新的数据
  - 解除某个编解码包对其数据缓冲区的引用，并不意味着该缓冲区立即被释放，因为它还可能被其它编解码包引用，但解除引用的确会使关于该缓冲区的引用计数减少一，当该引用计数被减至零时，说明已经没有任何编解码包引用该缓冲区，此时才会真正释放该缓冲区所占用的内存空间

知识讲解

av_frame_alloc

- 创建帧(AVFrame)对象，并将该对象初始化为缺省状态
  - `#include <libavutil/frame.h>`
    `AVFrame* av_frame_alloc (void);`
  - 成功返回帧对象指针，失败返回NULL
  - 所创建的帧对象，在使用完以后，必须通过av_frame_free函数销毁
  - 只创建帧对象本身，不包括其数据缓冲区，后者需要通过av_frame_get_buffer等函数另行分配，或者手动指定
- AVFrame类型的帧对象，可以理解为是对音视频流中基本处理单元的抽象。它既可以是某时刻的一幅静态图片(视频)，也可以是某时段的一组声音采样(音频)。任何诸如编码、解码、转码等后续处理，都是针对帧进行的

avcodec_receive_frame

- 从解码器接收已解码数据帧
  - `#include <libavcodec/avcodec.h>`

```
int avcodec_receive_frame (
    AVCodecContext* avctx,   // 解码器上下文
    AVFrame*        frame); // 已解码数据帧
```

  - 成功返回0，失败返回错误码(<0)，其中AVERROR(EAGAIN)表示暂时无帧可取，需要新的输入，AVERROR_EOF表示解码器已被刷流，再也收不到数据帧
  - 数据帧中的数据缓冲区带有引用计数，该函数在做任何操作之前，会先通过av_frame_unref函数，解除数据帧对其数据缓冲区的引用，并在操作完成后重建新的引用

知识讲解

av_frame_free

- 销毁帧对象本身，置空指向该对象的指针，同时将该帧数据缓冲区的引用计数减一，当其被减至零时，释放该缓冲区
  - `#include <libavutil/frame.h>`
    `void av_frame_free (`
    `  AVFrame** frame); // 帧`
  - 该函数的参数为指向某个已创建的帧对象的指针的地址，待函数返回后，该对象的指针即变为空指针

知识讲解

sws_getContext

- 创建缩放器上下文(SwsContext)对象
  -
```
#include <libswscale/swscale.h>
struct SwsContext* sws_getContext (
  int                srcW,      // 源图像宽度
  int                srcH,      // 源图像高度
  enum AVPixelFormat srcFormat, // 源图像格式
  int                dstW,      // 目标图像宽度
  int                dstH,      // 目标图像高度
  enum AVPixelFormat dstFormat, // 目标图像格式
  int                flags,     // 算法选型标志
  SwsFilter*         srcFilter, // 源过滤器，取NULL无源过滤器
  SwsFilter*         dstFilter, // 目标过滤器，取NULL无目标过滤器
  const double*      param);    // 额外参数，取NULL不设额外参数
```
  - 成功返回缩放器上下文对象指针，失败返回NULL
  - 所创建的缩放器上下文对象，在使用完以后，必须通过sws_freeContext函数销毁
- SwsContext类型的缩放器上下文对象，主要应用于针对视频帧的尺寸缩放和格式转换等操作

知识讲解

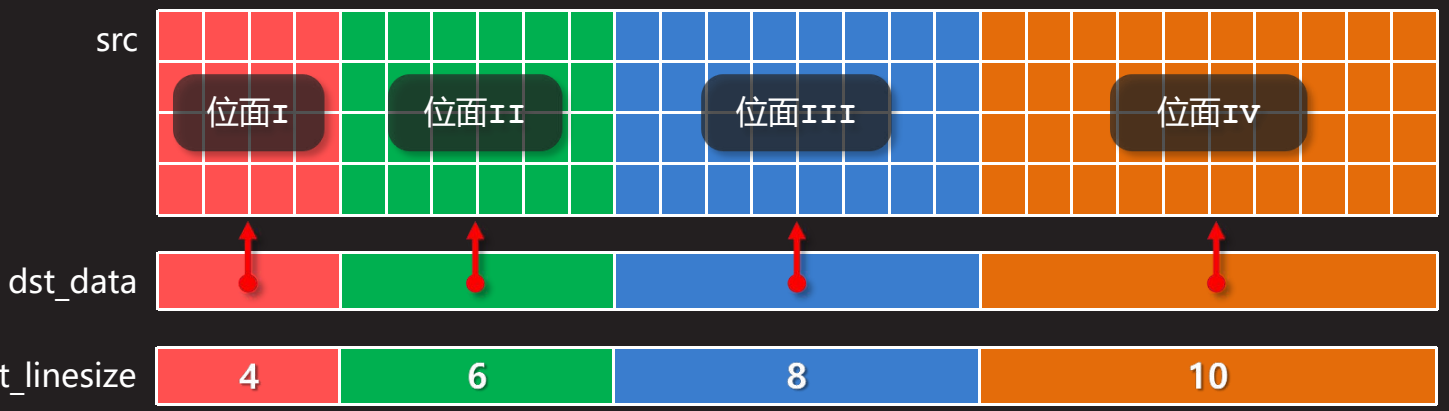
av_image_get_buffer_size

- 根据给定的参数计算存储一副静态图片所需要的字节数
  -
```
#include <libavutil/imgutils.h>
int av_image_get_buffer_size (
    enum AVPixelFormat pix_fmt, // 像素格式
    int                width,   // 宽度
    int                height,  // 高度
    int                align);  // 行对齐字节数
```
  - 成功返回所需字节数，失败返回错误码(<0)
  -
```
enum AVPixelFormat {
    ...
    AV_PIX_FMT_YUV420P, // YUV 4:2:0, 12bpp
    ...
    AV_PIX_FMT_BGR24,   // RGB 8:8:8, 24bpp, BGRBGR...
    ...
    AV_PIX_FMT_GRAY8,   // Y 8, 8bpp
    ...
};
```

知识讲解

av_image_fill_arrays

- 根据给定的参数设置图像位面指针数组和位面行长数组
  -
```
#include <libavutil/imgutils.h>
int av_image_fill_arrays (
  uint8_t*           dst_data[4],     // 位面指针数组
  int                dst_linesize[4], // 位面行长数组
  const uint8_t*     src,             // 图像缓冲区
  enum AVPixelFormat pix_fmt,         // 像素格式
  int                width,           // 图像宽度
  int                height,          // 图像高度
  int                align);          // 行对齐字节数
```
  - 成功返回图像缓冲区字节数，失败返回错误码(<0)



知识讲解

sws_scale

- 图像缩放与格式转换
  - `#include <libswscale/swscale.h>`

```
int sws_scale (
  struct SwsContext*   c,           // 缩放器上下文
  const uint8_t* const srcSlice[],  // 源图像位面指针数组
  const int            srcStride[], // 源图像位面行长数组
  int                  srcSliceY,   // 源图像起始行
  int                  srcSliceH,   // 源图像高度
  uint8_t* const       dst[],       // 目标图像位面指针数组
  const int            dstStride[]); // 目标图像位面行长数组
```

  - 成功返回目标图像高度，失败返回错误码(<0)

知识讲解

sws_freeContext

- 销毁缩放器上下文对象
  - `#include <libswscale/swscale.h>`
    `void sws_freeContext (`
    `struct SwsContext* swsContext); // 缩放器上下文`
  - 若参数swsContext取空指针，则该函数执行空操作

知识讲解

av_malloc

- 以适当的对齐方式分配一块内存
  - `#include <libavutil/mem.h>`

    `void* av_malloc (`

    `  size_t size) // 欲分配内存块的字节数`
  - 成功返回指向所分配内存块的指针，失败返回NULL
  - 所分配的内存，在使用完以后，必须通过av_free函数释放
  - 例如：

```
// 分配解码包
if (!(decPkt = (AVPacket*)av_malloc (
  sizeof (AVPacket)))) {
    ffmpegError ("av_malloc");
    goto failure;
}
```

av_free

- 释放之前通过av_malloc或者av_realloc分配的内存块
  - `#include <libavutil/mem.h>`
    `void av_free (`
    `  void* ptr); // 内存块指针`
  - 允许传入空指针，执行空操作

知识讲解

av_make_error_string

- 获取特定FFmpeg错误码的描述字符串
  - `#include <libavutil/error.h>`

```
char* av_make_error_string (
    char*  errbuf,        // 错误描述字符串缓冲区
    size_t errbuf_size,   // 错误描述字符串缓冲区字节数
    int    errnum);       // FFmpeg错误码
```

  - 返回第一个参数errbuf，其中已包含与第三个参数errnum相对应的错误描述字符串(以\0字符结尾)
  - FFmpeg错误描述字符串不会超过AV_ERROR_MAX_STRING_SIZE个字符

知识讲解

SDL_Init

- 初始化SDL库的特定子系统
  - `#include <SDL.h>`
    `int SDL_Init (`
    `  Uint32 flags); // 子系统标志`
  - 成功返回0，失败返回-1
  - flags参数可取如下值及其合理位或组合：
    - ✓ `SDL_INIT_AUDIO` // 音频子系统
    - ✓ `SDL_INIT_VIDEO` // 视频子系统
    - ✓ `SDL_INIT_JOYSTICK` // 游戏杆子系统
    - ✓ `SDL_INIT_HAPTIC` // 触觉子系统
    - ✓ …
    - ✓ `SDL_INIT_EVERYTHING` // 所有子系统

知识讲解

SDL_Quit

- 清理SDL库所有被初始化过的子系统
  - `#include <SDL.h>`
    `void SDL_Quit (void);`
  - 无论何种原因，只要进程终止，终止前总应该调用此函数

知识讲解

SDL_CreateWindow

Technology Tarena 达内科技

- 根据指定的位置、大小和标志创建一个窗口
  -
```
#include <SDL_video.h>
SDL_Window* SDL_CreateWindow (
  const char* title,  // 窗口标题，UTF-8编码
  int         x,      // 窗口左上角的屏幕坐标，以像素为单位，
  int         y,      // SDL_WINDOWPOS_CENTERED  - 居中
                      // SDL_WINDOWPOS_UNDEFINED - 默认
  int         w,      // 窗口的宽度和高度，
  int         h,      // 以像素为单位
  Uint32      flags); // 标志位
```
  - 成功返回指向所创建窗口对象的指针，失败返回NULL
  - flags参数可取如下值及其合理位或组合：
    - ✓ SDL_WINDOW_FULLSCREEN　　SDL_WINDOW_OPENGL
    - ✓ SDL_WINDOW_HIDDEN　　SDL_WINDOW_BORDERLESS
    - ✓ SDL_WINDOW_RESIZABLE　　SDL_WINDOW_MAXIMIZED
    - ✓ SDL_WINDOW_MINIMIZED　　SDL_WINDOW_INPUT_GRABBED
    - ✓ SDL_WINDOW_ALLOW_HIGHDPI

知识讲解

SDL_DestroyWindow

- 销毁窗口
  - `#include <SDL_video.h>`

```
void SDL_DestroyWindow (
    SDL_Window* window); // 窗口指针
```

知识讲解

SDL_CreateRenderer

- 为指定窗口创建2D渲染器上下文(SDL_Renderer)对象
  -
```
#include <SDL_render.h>
SDL_Renderer* SDL_CreateRenderer (
    SDL_Window* window, // 显示渲染效果的窗口
    int         index,  // 渲染器驱动索引,
                        // -1表示第一个满足要求
                        // 的渲染器驱动
    Uint32      flags); // 渲染器标志，0表示默认标志
```
  - 成功返回指向所创建渲染器上下文对象的指针，失败返回NULL
  - flags参数可取如下值及其合理位或组合：
    - ✓ SDL_RENDERER_SOFTWARE - 基于软件实现的渲染器
    - ✓ SDL_RENDERER_ACCELERATED - 使用硬件加速的渲染器
    - ✓ SDL_RENDERER_PRESENTVSYNC - 根据刷新频率同步显示
    - ✓ SDL_RENDERER_TARGETTEXTURE - 支持纹理渲染的渲染器

知识讲解

SDL_RenderCopy

- 将纹理复制到渲染目标
  -
```
#include <SDL_render.h>
int SDL_RenderCopy (
    SDL_Renderer*    renderer, // 渲染器上下文
    SDL_Texture*     texture,  // 纹理
    const SDL_Rect* srcrect,   // 源矩形,
                               // NULL表示全部
    const SDL_Rect* dstrect); // 目标矩形,
                               // NULL表示全部
```
  - 成功返回0，失败返回-1

SDL_RenderPresent

- 用渲染器更新屏幕显示
  - `#include <SDL_render.h>`

```
void SDL_RenderPresent (
    SDL_Renderer* renderer); // 渲染器上下文
```

知识讲解

SDL_DestroyRenderer

- 销毁渲染器上下文对象
  - `#include <SDL_render.h>`
    `void SDL_DestroyRenderer (`
    `SDL_Renderer* renderer); //` 渲染器上下文
  - 在渲染器上下文被销毁的同时，关联于其上的纹理也一并被销毁

知识讲解

SDL_CreateTexture

Technology Tarena 达内科技

- 为指定渲染器上下文创建纹理对象
  - `#include <SDL_render.h>`

```
SDL_Texture* SDL_CreateTexture (
    SDL_Renderer* renderer, // 渲染器上下文
    Uint32        format,   // 像素格式
    int           access,   // 访问模式
    int           w,        // 纹理宽度(像素)
    int           h);       // 纹理高度(像素)
```

  - 成功返回指向所创建纹理对象的指针，失败返回NULL
  - format参数可取如下值：
    - ✓ `SDL_PIXELFORMAT_BGR565` - 16位增强色
    - ✓ `SDL_PIXELFORMAT_BGR24` - 24位真彩色
    - ✓ `SDL_PIXELFORMAT_IYUV` - 亮度(Y)色度(UV)三位面
    - ✓ …
  - access参数可取如下值：
    - ✓ `SDL_TEXTUREACCESS_STATIC` - 少改变，不可锁定
    - ✓ `SDL_TEXTUREACCESS_STREAMING` - 频繁改变，可锁定
    - ✓ `SDL_TEXTUREACCESS_TARGET` - 可被作为渲染目标

知识讲解

SDL_UpdateYUVTexture

- 更新YUV纹理
  - #include <SDL_render.h>

```
int SDL_UpdateYUVTexture (
    SDL_Texture*     texture, // 待更新纹理
    const SDL_Rect* rect,     // 待更新矩形
    const Uint8*     Yplane,  // Y位面指针
    int              Ypitch,  // Y位面行长
    const Uint8*     Uplane,  // U位面指针
    int              Upitch,  // U位面行长
    const Uint8*     Vplane,  // V位面指针
    int              Vpitch); // V位面行长
```

  - 成功返回0，失败返回-1

知识讲解

SDL_DestroyTexture

- 销毁纹理对象
  - `#include <SDL_render.h>`

```
void SDL_DestroyTexture (
  SDL_Texture* texture); // 纹理
```

知识讲解

SDL_WaitEventTimeout

- 等待事件直至超时
  -
```
#include <SDL_events.h>
int SDL_WaitEventTimeout (
    SDL_Event* event,     // 等到的事件
    int        timeout);  // 等待超时，以毫秒计，
                          // 0表示不设超时
```
  - 成功返回1，失败返回0

知识讲解

SDL_Delay

- 延时
  - `#include <SDL_timer.h>`
    `void SDL_Delay (`
      `Uint32 ms); // 延时时长，以毫秒计`

知识讲解

SDL_GetError

- 获取最近一次SDL错误的描述字符串
  - `#include <SDL_error.h>`
    `const char* SDL_GetError (void);`
  - 返回最近一次SDL错误的描述字符串

知识讲解

Technology Tarena 达内科技

VideoPlayer

【参见：FFmpeg/Primer/VideoPlayer】

课堂练习

- 播放视频流
  - 本地播放：VideoPlayer mayitbe.vob
  - 视频点播：VideoPlayer rtmp://192.168.1.166/vod/1.mp4
  - 接收直播：VideoPlayer rtmp://192.168.1.166/live/1

Technology Tarena 达内科技

总结和答疑