原 SDL2源代码分析1：初始化（SDL_Init()）

2014年11月01日 15:46:06　阅读数：20188

==========

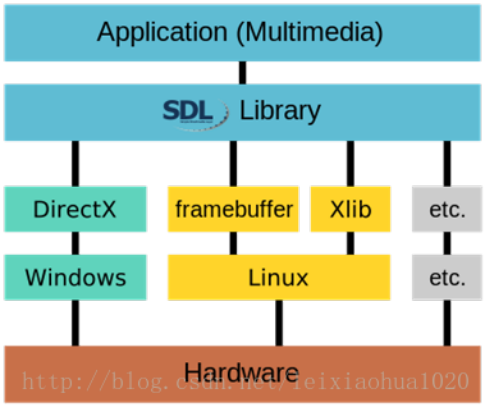
SDL源代码分析系列文章列表：

==========

打算花一段时间研究一下SDL的内部代码。前面几篇文章《 最简单的视音频播放示例1:总述 》中记录了视频、音频播放的技术，文中提及了SDL实际上封装了Direct3D，DirectSound这类的底层API。但是SDL究竟是如何封装的呢？这次打算深入其源代码一探究竟，看看它是如何封装这些API的。

SDL简介

有关SDL的简介在《 最简单的视音频播放示例7:SDL2播放RGB/YUV 》以及《 最简单的视音频播放示例9:SDL2播放PCM 》中已经叙述过了，不再重复。这两篇文章中也提到了一张SDL的原理图，如下所示：



从这个图中可以看出，SDL根据系统的不同调用不同的API完成相应的功能。至于它是如何实现的，将会在后文中详细叙述。下面不再罗嗦，直接进入正题。
使用SDL播放一个视频代码流程大体如下
初始化:

SDL_Init(): 初始化SDL。
SDL_CreateWindow(): 创建窗口（Window）。
SDL_CreateRenderer(): 基于窗口创建渲染器（Render）。
SDL_CreateTexture(): 创建纹理（Texture）。

循环渲染数据:

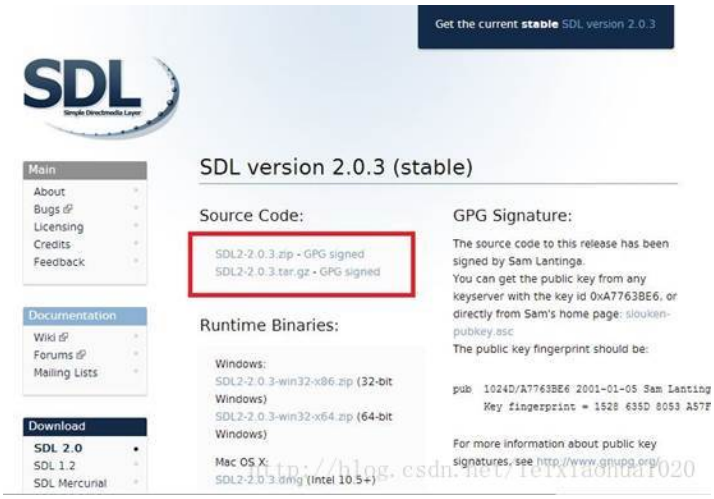
SDL_UpdateTexture(): 设置纹理的数据。
SDL_RenderCopy(): 纹理复制给渲染器。

SDL_RenderPresent(): 显示。

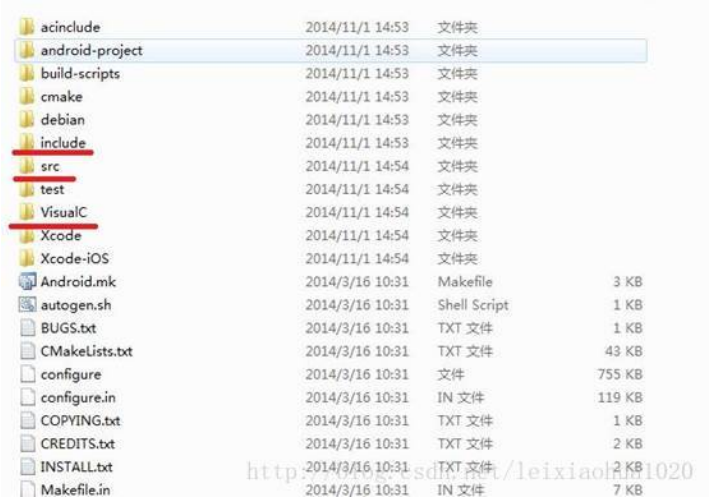
本文分析这个流程中最基本的一个函数SDL_Init()。SDL_Init()是SDL运行的初始，通过分析该函数，可以了解到SDL内部的架构。

获取源代码

SDL的源代码获取十分简单。访问SDL的官方网站（ http://www.libsdl.org/ ），单击左侧的“Download”进入下载页面，然后下载“SourceCode”栏目下的文件就可以了。



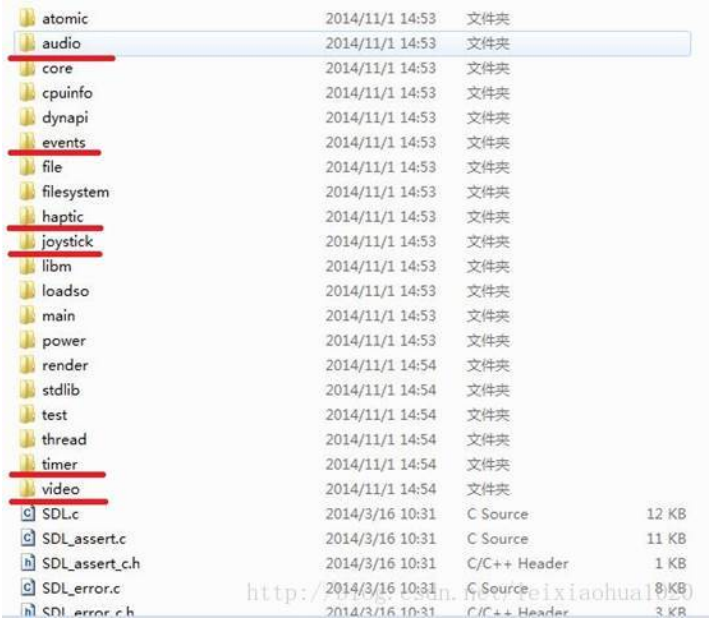
下载下来的文件只有4MB左右大小，但是解压缩之后竟然有50MB左右大小，确实不可思议。
解压缩之后，源代码目录如下图所示。



几个关键的文件夹如下所示：

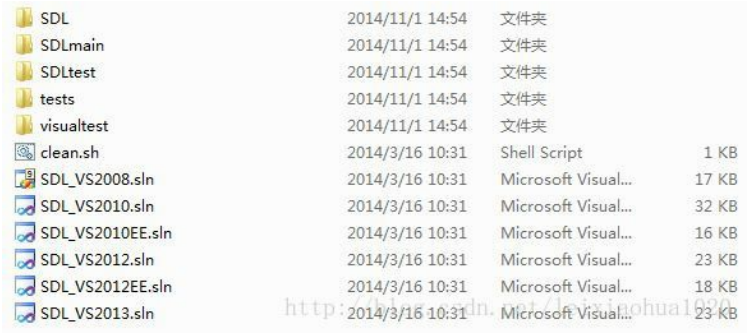
1. include：存储SDL的头文件的文件夹。

2. src：存储SDL源代码文件的文件夹。SDL根据功能模块的不同，将源代码分成了很多的文件夹。下图中标出了存储SDL几个子系统的源代码的文件夹。



3.

VisualC：存储VC解决方案的文件夹。从下图中可以看出，包含了VS2008，VS2010，VS2012，VS2013等各个版本的VC的解决方案。



实际上从文件名称我们可以看出，其它几个文件夹中，“Xcode，Xcode-iOS”包含了Xcode的项目文件，“test”包含了一些测试例子程序，“android-project”包含了Android下的项目文件。由于我们暂时不研究这些文件，就不详细分析了。

SDL_Init()

函数简介

下面这一部分进入正题，分析SDL的初始化函数SDL_Init()。该函数可以确定希望激活的子系统。SDL_Init()函数原型如下：

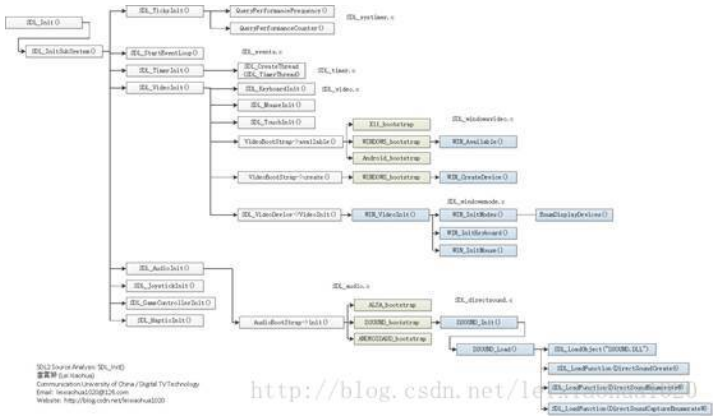
```cpp
int SDLCALL SDL_Init(Uint32 flags)
```

其中，flags可以取下列值：

SDL_INIT_TIMER：定时器
SDL_INIT_AUDIO：音频
SDL_INIT_VIDEO：视频
SDL_INIT_JOYSTICK：摇杆
SDL_INIT_HAPTIC：触摸屏
SDL_INIT_GAMECONTROLLER：游戏控制器
SDL_INIT_EVENTS：事件
SDL_INIT_NOPARACHUTE：不捕获关键信号（这个不理解）
SDL_INIT_EVERYTHING：包含上述所有选项

函数调用关系图

SDL_Init()关键函数的调用关系可以用下图表示。



上面的函数调用关系图本来是一张高清大图，但是博客不支持这么大尺寸的图片。因此把图片缩小了，看上去比较模糊。相册里面上传了一份原始的大图片：

http://my.csdn.net/leixiaohua1020/album/detail/1792993

选择上述相册里面的图片，右键选择“另存为”即可保存原始大图片。

源代码分析

SDL_Init()的实现位于SDL.c中。定义如下。

```cpp
int SDL_Init(Uint32 flags)
{
    return SDL_InitSubSystem(flags);
}
```

可以看出其代码只有一句，即调用了SDL_InitSubSystem()，下面我们看一下SDL_InitSubSystem()的定义。

```cpp
int SDL_InitSubSystem(Uint32 flags)
{
    if (!SDL_MainIsReady) {
        SDL_SetError("Application didn't initialize properly, did you include SDL_main.h in the file containing your main() function?");
        return -1;
    }

    /* Clear the error message */
    SDL_ClearError();

#if SDL_VIDEO_DRIVER_WINDOWS
    if ((flags & (SDL_INIT_HAPTIC|SDL_INIT_JOYSTICK))) {
        if (SDL_HelperWindowCreate() < 0) {
            return -1;
        }
    }
#endif

#if !SDL_TIMERS_DISABLED
    SDL_TicksInit();
#endif

    if ((flags & SDL_INIT_GAMECONTROLLER)) {
        /* game controller implies joystick */
        flags |= SDL_INIT_JOYSTICK;
    }

    if ((flags & (SDL_INIT_VIDEO|SDL_INIT_JOYSTICK))) {
        /* video or joystick implies events */
        flags |= SDL_INIT_EVENTS;
    }

    /* Initialize the event subsystem */
    if ((flags & SDL_INIT_EVENTS)) {
#if !SDL_EVENTS_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_EVENTS)) {
            if (SDL_StartEventLoop() < 0) {
```

```
                return (-1);
            }
            SDL_QuitInit();
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_EVENTS);
#else
        return SDL_SetError("SDL not built with events support");
#endif
    }


    /* Initialize the timer subsystem */
    if ((flags & SDL_INIT_TIMER)){
#if !SDL_TIMERS_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_TIMER)) {
            if (SDL_TimerInit() < 0) {
                return (-1);
            }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_TIMER);
#else
        return SDL_SetError("SDL not built with timer support");
#endif
    }


    /* Initialize the video subsystem */
    if ((flags & SDL_INIT_VIDEO)){
#if !SDL_VIDEO_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_VIDEO)) {
            if (SDL_VideoInit(NULL) < 0) {
                return (-1);
            }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_VIDEO);
#else
        return SDL_SetError("SDL not built with video support");
#endif
    }


    /* Initialize the audio subsystem */
    if ((flags & SDL_INIT_AUDIO)){
#if !SDL_AUDIO_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_AUDIO)) {
            if (SDL_AudioInit(NULL) < 0) {
                return (-1);
            }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_AUDIO);
#else
        return SDL_SetError("SDL not built with audio support");
#endif
    }


    /* Initialize the joystick subsystem */
    if ((flags & SDL_INIT_JOYSTICK)){
#if !SDL_JOYSTICK_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_JOYSTICK)) {
           if (SDL_JoystickInit() < 0) {
               return (-1);
           }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_JOYSTICK);
#else
        return SDL_SetError("SDL not built with joystick support");
#endif
    }


    if ((flags & SDL_INIT_GAMECONTROLLER)){
#if !SDL_JOYSTICK_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_GAMECONTROLLER)) {
            if (SDL_GameControllerInit() < 0) {
                return (-1);
            }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_GAMECONTROLLER);
#else
        return SDL_SetError("SDL not built with joystick support");
#endif
    }


    /* Initialize the haptic subsystem */
    if ((flags & SDL_INIT_HAPTIC)){
#if !SDL_HAPTIC_DISABLED
        if (SDL_PrivateShouldInitSubsystem(SDL_INIT_HAPTIC)) {
            if (SDL_HapticInit() < 0) {
                return (-1);
```

```
            }
        }
        SDL_PrivateSubsystemRefCountIncr(SDL_INIT_HAPTIC);
#else
        return SDL_SetError("SDL not built with haptic (force feedback) support");
#endif
    }


    return (0);
}
```

SDL_InitSubSystem()函数的定义看上去很长，实际上却并不复杂。下面简单阐述一下它的一些关键点：

1. 通过将传入的flag与子系统的宏定义（例如SDL_INIT_VIDEO，SDL_INIT_AUDIO等）相与，判断是否需要初始化该子系统。

2. 有很多的预定义的宏，用于判断SDL是否支持这些子系统。例如SDL_EVENTS_DISABLED，SDL_TIMERS_DISABLED，SDL_VIDEO_DISABLED，SDL_AUDIO_DISABLED，SDL_JOYSTICK_DISABLED，SDL_HAPTIC_DISABLED等。这些宏的定义位于SDL_config_minimal.h文件中，如下所示。

```cpp
/* Enable the dummy audio driver (src/audio/dummy/\*.c) */
#define SDL_AUDIO_DRIVER_DUMMY  1


/* Enable the stub joystick driver (src/joystick/dummy/\*.c) */
#define SDL_JOYSTICK_DISABLED   1


/* Enable the stub haptic driver (src/haptic/dummy/\*.c) */
#define SDL_HAPTIC_DISABLED 1


/* Enable the stub shared object loader (src/loadso/dummy/\*.c) */
#define SDL_LOADSO_DISABLED 1


/* Enable the stub thread support (src/thread/generic/\*.c) */
#define SDL_THREADS_DISABLED    1


/* Enable the stub timer support (src/timer/dummy/\*.c) */
#define SDL_TIMERS_DISABLED 1


/* Enable the dummy video driver (src/video/dummy/\*.c) */
#define SDL_VIDEO_DRIVER_DUMMY  1


/* Enable the dummy filesystem driver (src/filesystem/dummy/\*.c) */
#define SDL_FILESYSTEM_DUMMY  1
```

如果这些定义取值不为0，代表该子系统已经被disable了，就不编译指定子系统的源代码了。初始化的时候会调用SDL_SetError()函数输出错误信息。例如SDL_VIDEO_DISABLED如果设置为1的话，初始化视频子系统的时候会执行以下代码。

```cpp
SDL_SetError("SDL not built with video support");
```

3. 在每一个子系统真正初始化之前，都会调用一个函数SDL_PrivateShouldInitSubsystem()。该函数用于检查目标子系统是否需要初始化。

4. 在一个子系统初始化之后，都会调用一个函数SDL_PrivateSubsystemRefCountIncr()。该函数用于增加子系统的引用计数。

5. 下表列出各个子系统的初始化函数。

子系统名称	函数
AUDIO（音频）	SDL_AudioInit()
VIDEO（视频）	SDL_VideoInit()
TIMERS（定时器）	SDL_TicksInit()，SDL_TimerInit()

EVENTS（事件）	SDL_StartEventLoop()
JOYSTICK（摇杆）	SDL_GameControllerInit()
HAPTIC（触摸屏）	SDL_HapticInit()

我们先不看JOYSTICK（摇杆），HAPTIC（触摸屏）这些方面的代码，专门关注AUDIO（音频），VIDEO（视频）这两个方面的代码。

1. VIDEO（视频）

视频子系统的初始化函数是SDL_VideoInit()。它的源代码位于video\SDL_video.c文件中，如下所示。

```cpp
/*
 * Initialize the video and event subsystems -- determine native pixel format
 */
int SDL_VideoInit(const char *driver_name)
{
    SDL_VideoDevice *video;
    const char *hint;
    int index;
    int i;
    SDL_bool allow_screensaver;


    /* Check to make sure we don't overwrite '_this' */
    if (_this != NULL) {
        SDL_VideoQuit();
    }


#if !SDL_TIMERS_DISABLED
    SDL_TicksInit();
#endif


    /* Start the event loop */
    if (SDL_InitSubSystem(SDL_INIT_EVENTS) < 0 ||
        SDL_KeyboardInit() < 0 ||
        SDL_MouseInit() < 0 ||
        SDL_TouchInit() < 0) {
        return -1;
    }


    /* Select the proper video driver */
    index = 0;
    video = NULL;
    if (driver_name == NULL) {
        driver_name = SDL_getenv("SDL_VIDEODRIVER");
    }
    if (driver_name != NULL) {
        for (i = 0; bootstrap[i]; ++i) {
            if (SDL_strncasecmp(bootstrap[i]->name, driver_name, SDL_strlen(driver_name)) == 0) {
                if (bootstrap[i]->available()) {
                    video = bootstrap[i]->create(index);
                    break;
                }
            }
        }
    } else {
        for (i = 0; bootstrap[i]; ++i) {
            if (bootstrap[i]->available()) {
                video = bootstrap[i]->create(index);
                if (video != NULL) {
                    break;
                }
            }
        }
    }
    if (video == NULL) {
        if (driver_name) {
            return SDL_SetError("%s not available", driver_name);
        }
        return SDL_SetError("No available video device");
    }
    _this = video;
    _this->name = bootstrap[i]->name;
    _this->next_object_id = 1;


    /* Set some very sane GL defaults */
    _this->gl_config.driver_loaded = 0;
```

```
    _this->gl_config.dll_handle = NULL;
    SDL_GL_ResetAttributes();


    _this->current_glwin_tls = SDL_TLSCreate();
    _this->current_glctx_tls = SDL_TLSCreate();


    /* Initialize the video subsystem */
    if (_this->VideoInit(_this) < 0) {
        SDL_VideoQuit();
        return -1;
    }


    /* Make sure some displays were added */
    if (_this->num_displays == 0) {
        SDL_VideoQuit();
        return SDL_SetError("The video driver did not add any displays");
    }


    /* Add the renderer framebuffer emulation if desired */
    if (ShouldUseTextureFramebuffer()) {
        _this->CreateWindowFramebuffer = SDL_CreateWindowTexture;
        _this->UpdateWindowFramebuffer = SDL_UpdateWindowTexture;
        _this->DestroyWindowFramebuffer = SDL_DestroyWindowTexture;
    }


    /* Disable the screen saver by default. This is a change from <= 2.0.1,
       but most things using SDL are games or media players; you wouldn't
       want a screensaver to trigger if you're playing exclusively with a
       joystick, or passively watching a movie. Things that use SDL but
       function more like a normal desktop app should explicitly reenable the
       screensaver. */
    hint = SDL_GetHint(SDL_HINT_VIDEO_ALLOW_SCREENSAVER);
    if (hint) {
        allow_screensaver = SDL_atoi(hint) ? SDL_TRUE : SDL_FALSE;
    } else {
        allow_screensaver = SDL_FALSE;
    }
    if (!allow_screensaver) {
        SDL_DisableScreenSaver();
    }


    /* If we don't use a screen keyboard, turn on text input by default,
       otherwise programs that expect to get text events without enabling
       UNICODE input won't get any events.


       Actually, come to think of it, you needed to call SDL_EnableUNICODE(1)
       in SDL 1.2 before you got text input events.  Hmm...
     */
    if (!SDL_HasScreenKeyboardSupport()) {
        SDL_StartTextInput();
    }


    /* We're ready to go! */
    return 0;
}
```

下面简单阐述一下它的大致步骤：

1.

初始化一些子系统，比如EVENTS（事件）子系统。 也就是说，就算在调用SDL_Init()的时候不指定初始化EVENTS子系统，在初始化VIDEO子系统的时候，同样也会初始化EVENTS子系统。

2.

选择一个合适的SDL_VideoDevice。

在这里，涉及到两个重要的结构体：SDL_VideoDevice以及VideoBootStrap。其中SDL_VideoDevice代表了一个视频驱动程序。VideoBootStrap从字面上理解是“视频驱动程序的引导程序”，即用于创建一个SDL_VideoDevice。因此，我们先来看看VideoBootStrap这个结构体。它的定义如下（位于video\SDL_sysvideo.h）。

```cpp
typedef struct VideoBootStrap
{
    const char *name;
    const char *desc;
    int (*available) (void);
    SDL_VideoDevice *(*create) (int devindex);
} VideoBootStrap;
```

可以看出它的定义比较简单，每个字段的含义如下：

name：驱动名称

desc：描述
available()：检查是否可用的一个函数指针
create()：创建SDL_VideoDevice的函数指针

SDL中有一个VideoBootStrap类型的静态数组bootstrap。用于存储SDL支持的视频驱动程序。注意这是SDL“跨平台”特性中很重要的一部分。该静态数组定义如下（位于video\SDL_video.c）。

```cpp
/* Available video drivers */
static VideoBootStrap *bootstrap[] = {
#if SDL_VIDEO_DRIVER_COCOA
    &COCOA_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_X11
    &X11_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_MIR
    &MIR_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_DIRECTFB
    &DirectFB_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_WINDOWS
    &WINDOWS_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_WINRT
    &WINRT_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_HAIKU
    &HAIKU_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_PANDORA
    &PND_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_UIKIT
    &UIKIT_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_ANDROID
    &Android_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_PSP
    &PSP_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_RPI
    &RPI_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_WAYLAND
    &Wayland_bootstrap,
#endif
#if SDL_VIDEO_DRIVER_DUMMY
    &DUMMY_bootstrap,
#endif
    NULL
};
```

从代码中可以看出，SDL通过预编译宏取值是否非0来判断是否支持该视频驱动。我们可以看一下Windows的视频设备驱动的定义。该设备驱动同样是一个静态变量，名称为WINDOWS_bootstrap（位于video\windows\SDL_windowsvideo.c）。

```cpp
VideoBootStrap WINDOWS_bootstrap = {
    "windows", "SDL Windows video driver", WIN_Available, WIN_CreateDevice
};
```

可以看出该视频驱动名称为“windows”，描述为“SDL Windows video driver”，检查是否可用的函数为“WIN_Available()”，创建SDL_VideoDevice的函数为“WIN_CreateDevice()”。
同样， Android的视频设备驱动的名称为Android_bootstrap；PSP的视频设备驱动为PSP_bootstrap；X11的视频设备驱动为X11_bootstrap。不再一一例举。
下面看一下Windows视频驱动中那两个函数的定义。WIN_Available()定义如下。

```cpp
static int WIN_Available(void)
{
    return (1);
}
```

可见该函数没有做任何的处理。WIN_CreateDevice()定义如下。

```cpp
static SDL_VideoDevice *
```

```
WIN_CreateDevice(int devIndex)
{
    SDL_VideoDevice *device;
    SDL_VideoData *data;


    SDL_RegisterApp(NULL, 0, NULL);


    /* Initialize all variables that we clean on shutdown */
    device = (SDL_VideoDevice *) SDL_calloc(1, sizeof(SDL_VideoDevice));
    if (device) {
        data = (struct SDL_VideoData *) SDL_calloc(1, sizeof(SDL_VideoData));
    } else {
        data = NULL;
    }
    if (!data) {
        SDL_free(device);
        SDL_OutOfMemory();
        return NULL;
    }
    device->driverdata = data;


    data->userDLL = SDL_LoadObject("USER32.DLL");
    if (data->userDLL) {
        data->CloseTouchInputHandle = (BOOL (WINAPI *)( HTOUCHINPUT )) SDL_LoadFunction(data->userDLL, "CloseTouchInputHandle");
        data->GetTouchInputInfo = (BOOL (WINAPI *)( HTOUCHINPUT, UINT, PTOUCHINPUT, int )) SDL_LoadFunction(data->userDLL, "GetTouch
InputInfo");
        data->RegisterTouchWindow = (BOOL (WINAPI *)( HWND, ULONG )) SDL_LoadFunction(data->userDLL, "RegisterTouchWindow");
    }


    /* Set the function pointers */
    device->VideoInit = WIN_VideoInit;
    device->VideoQuit = WIN_VideoQuit;
    device->GetDisplayBounds = WIN_GetDisplayBounds;
    device->GetDisplayModes = WIN_GetDisplayModes;
    device->SetDisplayMode = WIN_SetDisplayMode;
    device->PumpEvents = WIN_PumpEvents;


#undef CreateWindow
    device->CreateWindow = WIN_CreateWindow;
    device->CreateWindowFrom = WIN_CreateWindowFrom;
    device->SetWindowTitle = WIN_SetWindowTitle;
    device->SetWindowIcon = WIN_SetWindowIcon;
    device->SetWindowPosition = WIN_SetWindowPosition;
    device->SetWindowSize = WIN_SetWindowSize;
    device->ShowWindow = WIN_ShowWindow;
    device->HideWindow = WIN_HideWindow;
    device->RaiseWindow = WIN_RaiseWindow;
    device->MaximizeWindow = WIN_MaximizeWindow;
    device->MinimizeWindow = WIN_MinimizeWindow;
    device->RestoreWindow = WIN_RestoreWindow;
    device->SetWindowBordered = WIN_SetWindowBordered;
    device->SetWindowFullscreen = WIN_SetWindowFullscreen;
    device->SetWindowGammaRamp = WIN_SetWindowGammaRamp;
    device->GetWindowGammaRamp = WIN_GetWindowGammaRamp;
    device->SetWindowGrab = WIN_SetWindowGrab;
    device->DestroyWindow = WIN_DestroyWindow;
    device->GetWindowWMInfo = WIN_GetWindowWMInfo;
    device->CreateWindowFramebuffer = WIN_CreateWindowFramebuffer;
    device->UpdateWindowFramebuffer = WIN_UpdateWindowFramebuffer;
    device->DestroyWindowFramebuffer = WIN_DestroyWindowFramebuffer;
    device->OnWindowEnter = WIN_OnWindowEnter;


    device->shape_driver.CreateShaper = Win32_CreateShaper;
    device->shape_driver.SetWindowShape = Win32_SetWindowShape;
    device->shape_driver.ResizeWindowShape = Win32_ResizeWindowShape;


#if SDL_VIDEO_OPENGL_WGL
    device->GL_LoadLibrary = WIN_GL_LoadLibrary;
    device->GL_GetProcAddress = WIN_GL_GetProcAddress;
    device->GL_UnloadLibrary = WIN_GL_UnloadLibrary;
    device->GL_CreateContext = WIN_GL_CreateContext;
    device->GL_MakeCurrent = WIN_GL_MakeCurrent;
    device->GL_SetSwapInterval = WIN_GL_SetSwapInterval;
    device->GL_GetSwapInterval = WIN_GL_GetSwapInterval;
    device->GL_SwapWindow = WIN_GL_SwapWindow;
    device->GL_DeleteContext = WIN_GL_DeleteContext;
#endif
    device->StartTextInput = WIN_StartTextInput;
    device->StopTextInput = WIN_StopTextInput;
    device->SetTextInputRect = WIN_SetTextInputRect;


    device->SetClipboardText = WIN_SetClipboardText;
    device->GetClipboardText = WIN_GetClipboardText;
    device->HasClipboardText = WIN_HasClipboardText;
```

```
    device->HasClipboardText = WIN_HasClipboardText;


    device->free = WIN_DeleteDevice;


    return device;
}
```

该函数首先通过SDL_calloc()的方法为创建的SDL_VideoDevice分配了一块内存，接下来为创建的SDL_VideoDevice结构体中的函数指针赋了一大堆的值。这也是SDL"跨平台"特性的一个特点：通过调用SDL_VideoDevice中的接口函数，就可以调用不同平台的具体实现功能的函数。

PS：在这里补充一个SDL中内存分配函数的知识。在SDL中分配内存使用SDL_malloc()，SDL_calloc()，这些函数实际上就是malloc()，calloc()。它们的定义位于stdlib\SDL_malloc.c文件中。如下所示：

```cpp
#define memset  SDL_memset
#define memcpy  SDL_memcpy
#define malloc  SDL_malloc
#define calloc  SDL_calloc
#define realloc SDL_realloc
#define free    SDL_free
```

下面来看一下SDL_VideoDevice这个结构体的定义（位于video\SDL_sysvideo.h）。

```cpp
struct SDL_VideoDevice
{
    /* * * */
    /* The name of this video driver */
    const char *name;


    /* * * */
    /* Initialization/Query functions */


    /*
     * Initialize the native video subsystem, filling in the list of
     * displays for this driver, returning 0 or -1 if there's an error.
     */
    int (*VideoInit) (_THIS);


    /*
     * Reverse the effects VideoInit() -- called if VideoInit() fails or
     * if the application is shutting down the video subsystem.
     */
    void (*VideoQuit) (_THIS);


    /* * * */
    /*
     * Display functions
     */


    /*
     * Get the bounds of a display
     */
    int (*GetDisplayBounds) (_THIS, SDL_VideoDisplay * display, SDL_Rect * rect);


    /*
     * Get a list of the available display modes for a display.
     */
    void (*GetDisplayModes) (_THIS, SDL_VideoDisplay * display);


    /*
     * Setting the display mode is independent of creating windows, so
     * when the display mode is changed, all existing windows should have
     * their data updated accordingly, including the display surfaces
     * associated with them.
     */
    int (*SetDisplayMode) (_THIS, SDL_VideoDisplay * display, SDL_DisplayMode * mode);


    /* * * */
    /*
     * Window functions
     */
    int (*CreateWindow) (_THIS, SDL_Window * window);
    int (*CreateWindowFrom) (_THIS, SDL_Window * window, const void *data);
    void (*SetWindowTitle) (_THIS, SDL_Window * window);
```

```
    void (*SetWindowIcon) (_THIS, SDL_Window * window, SDL_Surface * icon);
    void (*SetWindowPosition) (_THIS, SDL_Window * window);
    void (*SetWindowSize) (_THIS, SDL_Window * window);
    void (*SetWindowMinimumSize) (_THIS, SDL_Window * window);
    void (*SetWindowMaximumSize) (_THIS, SDL_Window * window);
    void (*ShowWindow) (_THIS, SDL_Window * window);
    void (*HideWindow) (_THIS, SDL_Window * window);
    void (*RaiseWindow) (_THIS, SDL_Window * window);
    void (*MaximizeWindow) (_THIS, SDL_Window * window);
    void (*MinimizeWindow) (_THIS, SDL_Window * window);
    void (*RestoreWindow) (_THIS, SDL_Window * window);
    void (*SetWindowBordered) (_THIS, SDL_Window * window, SDL_bool bordered);
    void (*SetWindowFullscreen) (_THIS, SDL_Window * window, SDL_VideoDisplay * display, SDL_bool fullscreen);
    int (*SetWindowGammaRamp) (_THIS, SDL_Window * window, const Uint16 * ramp);
    int (*GetWindowGammaRamp) (_THIS, SDL_Window * window, Uint16 * ramp);
    void (*SetWindowGrab) (_THIS, SDL_Window * window, SDL_bool grabbed);
    void (*DestroyWindow) (_THIS, SDL_Window * window);
    int (*CreateWindowFramebuffer) (_THIS, SDL_Window * window, Uint32 * format, void ** pixels, int *pitch);
    int (*UpdateWindowFramebuffer) (_THIS, SDL_Window * window, const SDL_Rect * rects, int numrects);
    void (*DestroyWindowFramebuffer) (_THIS, SDL_Window * window);
    void (*OnWindowEnter) (_THIS, SDL_Window * window);


    /* * * */
    /*
     * Shaped-window functions
     */
    SDL_ShapeDriver shape_driver;


    /* Get some platform dependent window information */
      SDL_bool(*GetWindowWMInfo) (_THIS, SDL_Window * window,
                                  struct SDL_SysWMinfo * info);


    /* * * */
    /*
     * OpenGL support
     */
    int (*GL_LoadLibrary) (_THIS, const char *path);
    void *(*GL_GetProcAddress) (_THIS, const char *proc);
    void (*GL_UnloadLibrary) (_THIS);
      SDL_GLContext(*GL_CreateContext) (_THIS, SDL_Window * window);
    int (*GL_MakeCurrent) (_THIS, SDL_Window * window, SDL_GLContext context);
    void (*GL_GetDrawableSize) (_THIS, SDL_Window * window, int *w, int *h);
    int (*GL_SetSwapInterval) (_THIS, int interval);
    int (*GL_GetSwapInterval) (_THIS);
    void (*GL_SwapWindow) (_THIS, SDL_Window * window);
    void (*GL_DeleteContext) (_THIS, SDL_GLContext context);


    /* * * */
    /*
     * Event manager functions
     */
    void (*PumpEvents) (_THIS);


    /* Suspend the screensaver */
    void (*SuspendScreenSaver) (_THIS);


    /* Text input */
    void (*StartTextInput) (_THIS);
    void (*StopTextInput) (_THIS);
    void (*SetTextInputRect) (_THIS, SDL_Rect *rect);


    /* Screen keyboard */
    SDL_bool (*HasScreenKeyboardSupport) (_THIS);
    void (*ShowScreenKeyboard) (_THIS, SDL_Window *window);
    void (*HideScreenKeyboard) (_THIS, SDL_Window *window);
    SDL_bool (*IsScreenKeyboardShown) (_THIS, SDL_Window *window);


    /* Clipboard */
    int (*SetClipboardText) (_THIS, const char *text);
    char * (*GetClipboardText) (_THIS);
    SDL_bool (*HasClipboardText) (_THIS);


    /* MessageBox */
    int (*ShowMessageBox) (_THIS, const SDL_MessageBoxData *messageboxdata, int *buttonid);


    /* * * */
    /* Data common to all drivers */
    SDL_bool suspend_screensaver;
    int num_displays;
    SDL_VideoDisplay *displays;
    SDL_Window *windows;
```

```
    Uint8 window_magic;
    Uint32 next_object_id;
    char * clipboard_text;


    /* * * */
    /* Data used by the GL drivers */
    struct
    {
        int red_size;
        int green_size;
        int blue_size;
        int alpha_size;
        int depth_size;
        int buffer_size;
        int stencil_size;
        int double_buffer;
        int accum_red_size;
        int accum_green_size;
        int accum_blue_size;
        int accum_alpha_size;
        int stereo;
        int multisamplebuffers;
        int multisamplesamples;
        int accelerated;
        int major_version;
        int minor_version;
        int flags;
        int profile_mask;
        int share_with_current_context;
        int framebuffer_srgb_capable;
        int retained_backing;
        int driver_loaded;
        char driver_path[256];
        void *dll_handle;
    } gl_config;


    /* * * */
    /* Cache current GL context; don't call the OS when it hasn't changed. */
    /* We have the global pointers here so Cocoa continues to work the way
       it always has, and the thread-local storage for the general case.
     */
    SDL_Window *current_glwin;
    SDL_GLContext current_glctx;
    SDL_TLSID current_glwin_tls;
    SDL_TLSID current_glctx_tls;


    /* * * */
    /* Data private to this driver */
    void *driverdata;
    struct SDL_GLDriverData *gl_data;

#if SDL_VIDEO_OPENGL_EGL
    struct SDL_EGL_VideoData *egl_data;
#endif

#if SDL_VIDEO_OPENGL_ES || SDL_VIDEO_OPENGL_ES2
    struct SDL_PrivateGLESData *gles_data;
#endif


    /* * * */
    /* The function used to dispose of this structure */
    void (*free) (_THIS);
};
```

这个结构体包含了一大堆的函数指针。这些指针在前文所说的VideoBootStrap的create()方法调用的时候会被赋值。SDL通过调用这些函数指针，完成视频显示的各项内容。

3.

调用选中的SDL_VideoDevice的VideoInit()方法。

选择了合适的SDL_VideoDevice之后，调用该SDL_VideoDevice的VideoInit()就可以真正的初始化视频驱动了。以Windows系统为例。从前文的函数中可以看出，Windows系统的VideoInit()接口实际上调用了WIN_VideoInit()函数。我们来看一下WIN_VideoInit()函数的定义（位于video\windows\SDL_windowsvideo.c）。

```cpp
int WIN_VideoInit(_THIS)
{
    if (WIN_InitModes(_this) < 0) {
        return -1;
    }


    WIN_InitKeyboard(_this);
    WIN_InitMouse(_this);


    return 0;
}
```

其中有3个函数：WIN_InitModes()，WIN_InitKeyboard()，WIN_InitMouse()。后两个函数用于初始化键盘和鼠标，我们暂且不研究。看一下WIN_InitModes()的函数。

```cpp
int WIN_InitModes(_THIS)
{
    int pass;
    DWORD i, j, count;
    DISPLAY_DEVICE device;


    device.cb = sizeof(device);


    /* Get the primary display in the first pass */
    for (pass = 0; pass < 2; ++pass) {
        for (i = 0; ; ++i) {
            TCHAR DeviceName[32];


            if (!EnumDisplayDevices(NULL, i, &device, 0)) {
                break;
            }
            if (!(device.StateFlags & DISPLAY_DEVICE_ATTACHED_TO_DESKTOP)) {
                continue;
            }
            if (pass == 0) {
                if (!(device.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE)) {
                    continue;
                }
            } else {
                if (device.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE) {
                    continue;
                }
            }
            SDL_memcpy(DeviceName, device.DeviceName, sizeof(DeviceName));
#ifdef DEBUG_MODES
            printf("Device: %s\n", WIN_StringToUTF8(DeviceName));
#endif
            count = 0;
            for (j = 0; ; ++j) {
                if (!EnumDisplayDevices(DeviceName, j, &device, 0)) {
                    break;
                }
                if (!(device.StateFlags & DISPLAY_DEVICE_ATTACHED_TO_DESKTOP)) {
                    continue;
                }
                if (pass == 0) {
                    if (!(device.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE)) {
                        continue;
                    }
                } else {
                    if (device.StateFlags & DISPLAY_DEVICE_PRIMARY_DEVICE) {
                        continue;
                    }
                }
                count += WIN_AddDisplay(device.DeviceName);
            }
            if (count == 0) {
                WIN_AddDisplay(DeviceName);
            }
        }
    }
    if (_this->num_displays == 0) {
        return SDL_SetError("No displays available");
    }
    return 0;
}
```

该函数的作用就是获得系统中显示设备的信息。目前还没有深入研究，待有时间再补上该函数的分析。

2. AUDIO（音频）

音频子系统的初始化函数是SDL_ AudioInit ()。它的源代码位于audio\SDL_audio.c文件中，如下所示。

```cpp
int SDL_AudioInit(const char *driver_name)
{
    int i = 0;
    int initialized = 0;
    int tried_to_init = 0;


    if (SDL_WasInit(SDL_INIT_AUDIO)) {
        SDL_AudioQuit();        /* shutdown driver if already running. */
    }


    SDL_memset(¤t_audio, '\0', sizeof(current_audio));
    SDL_memset(open_devices, '\0', sizeof(open_devices));


    /* Select the proper audio driver */
    if (driver_name == NULL) {
        driver_name = SDL_getenv("SDL_AUDIODRIVER");
    }


    for (i = 0; (!initialized) && (bootstrap[i]); ++i) {
        /* make sure we should even try this driver before doing so... */
        const AudioBootStrap *backend = bootstrap[i];
        if ((driver_name && (SDL_strncasecmp(backend->name, driver_name, SDL_strlen(driver_name)) != 0)) ||
            (!driver_name && backend->demand_only)) {
            continue;
        }


        tried_to_init = 1;
        SDL_memset(¤t_audio, 0, sizeof(current_audio));
        current_audio.name = backend->name;
        current_audio.desc = backend->desc;
        initialized = backend->init(¤t_audio.impl);
    }


    if (!initialized) {
        /* specific drivers will set the error message if they fail... */
        if (!tried_to_init) {
            if (driver_name) {
                SDL_SetError("Audio target '%s' not available", driver_name);
            } else {
                SDL_SetError("No available audio device");
            }
        }


        SDL_memset(¤t_audio, 0, sizeof(current_audio));
        return (-1);            /* No driver was available, so fail. */
    }


    finalize_audio_entry_points();


    return (0);
}
```

音频初始化和视频很类似，比视频简单一些，关键在于选择一个合适的SDL_AudioDriver。

在这里，涉及到两个重要的结构体：SDL_AudioDriver以及AudioBootStrap。其中SDL_AudioDriver代表了一个音频驱动程序。AudioBootStrap从字面上理解是“音频驱动程序的引导程序”，即用于创建一个SDL_AudioDriver。可以看出音频子系统中的结构体和视频子系统中的结构体的格式基本上是一模一样的。我们先来看看AudioBootStrap这个结构体。它的定义如下（位于audio\SDL_sysaudio.h）。

```cpp
typedef struct AudioBootStrap
{
    const char *name;
    const char *desc;
    int (*init) (SDL_AudioDriverImpl * impl);
    int demand_only;  /* 1==request explicitly, or it won't be available. */
} AudioBootStrap;
```

可以看出它的定义比较简单，每个字段的含义如下：

name：驱动名称

desc：描述

init()：创建SDL_AudioDriver的函数指针

demand_only：没有研究过。

SDL中有一个AudioBootStrap类型的静态数组bootstrap。用于存储SDL支持的音频驱动程序。该静态数组定义如下（位于audio\SDL_audio.c）。

```cpp
/* Available audio drivers */
static const AudioBootStrap *const bootstrap[] = {
#if SDL_AUDIO_DRIVER_PULSEAUDIO
    &PULSEAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_ALSA
    &ALSA_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_SNDIO
    &SNDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_BSD
    &BSD_AUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_OSS
    &DSP_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_QSA
    &QSAAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_SUNAUDIO
    &SUNAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_ARTS
    &ARTS_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_ESD
    &ESD_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_NAS
    &NAS_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_XAUDIO2
    &XAUDIO2_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_DSOUND
    &DSOUND_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_WINMM
    &WINMM_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_PAUDIO
    &PAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_HAIKU
    &HAIKUAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_COREAUDIO
    &COREAUDIO_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_DISK
    &DISKAUD_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_DUMMY
    &DUMMYAUD_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_FUSIONSOUND
    &FUSIONSOUND_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_ANDROID
    &ANDROIDAUD_bootstrap,
#endif
#if SDL_AUDIO_DRIVER_PSP
    &PSPAUD_bootstrap,
#endif
    NULL
};
```

在这里我们可以看一下DirectSound的AudioBootStrap的变量DSOUND_bootstrap（audio\directsound\SDL_directsound.c）。

```cpp
AudioBootStrap DSOUND_bootstrap = {
    "directsound", "DirectSound", DSOUND_Init, 0
};
```

可以看出该音频驱动名称为“directsound”，描述为“DirectSound”，创建SDL_AudioDriver的函数为“DSOUND_Init()”。
下面看一下DirectSound初始化函数DSOUND_Init()的定义。

```cpp
static int DSOUND_Init(SDL_AudioDriverImpl * impl)
{
    if (!DSOUND_Load()) {
        return 0;
    }


    /* Set the function pointers */
    impl->DetectDevices = DSOUND_DetectDevices;
    impl->OpenDevice = DSOUND_OpenDevice;
    impl->PlayDevice = DSOUND_PlayDevice;
    impl->WaitDevice = DSOUND_WaitDevice;
    impl->WaitDone = DSOUND_WaitDone;
    impl->GetDeviceBuf = DSOUND_GetDeviceBuf;
    impl->CloseDevice = DSOUND_CloseDevice;
    impl->Deinitialize = DSOUND_Deinitialize;


    return 1;   /* this audio target is available. */
}
```

和视频驱动的初始化一样，音频驱动初始化也是对SDL_AudioDriver的接口指针进行赋值。在这里涉及到了一个DirectSound的加载函数DSOUND_Load()，我们可以看一下它的代码。

```cpp
static int DSOUND_Load(void)
{
    int loaded = 0;


    DSOUND_Unload();


    DSoundDLL = SDL_LoadObject("DSOUND.DLL");
    if (DSoundDLL == NULL) {
        SDL_SetError("DirectSound: failed to load DSOUND.DLL");
    } else {
        /* Now make sure we have DirectX 8 or better... */
        #define DSOUNDLOAD(f) { \
            p##f = (fn##f) SDL_LoadFunction(DSoundDLL, #f); \
            if (!p##f) loaded = 0; \
        }
        loaded = 1;  /* will reset if necessary. */
        DSOUNDLOAD(DirectSoundCreate8);
        DSOUNDLOAD(DirectSoundEnumerateW);
        DSOUNDLOAD(DirectSoundCaptureEnumerateW);
        #undef DSOUNDLOAD


        if (!loaded) {
            SDL_SetError("DirectSound: System doesn't appear to have DX8.");
        }
    }


    if (!loaded) {
        DSOUND_Unload();
    }


    return loaded;
}
```

从代码中可以看出，该函数加载了“DSOUND.DLL”的DirectSoundCreate8()，DirectSoundEnumerateW()，DirectSoundCaptureEnumerateW()函数。

文章标签： SDL 初始化 源代码分析 视频 音频

个人分类： SDL

所属专栏： 开源多媒体项目源代码分析