SDL 2.0 API

在SDL2文档官网可以查询SDL的函数和数据结构的帮助信息。
SDL2文档官网（按类别） https://wiki.libsdl.org/APIByCategory
SDL2文档官网（按名称） https://wiki.libsdl.org/CategoryAPI
以下是部分函数的中文翻译

函数 SDL_Init: 初始化SDL系统

```
int SDL_Init(Uint32 flags)
```

参数: flags是以下选项的组合（按位或操作）

flags选项	子系统
SDL_INIT_VIDEO	视频子系统
SDL_INIT_EVENTS	事件处理子系统
SDL_INIT_TIMER	定时器子系统
SDL_INIT_AUDIO	音频子系统
SDL_INIT_JOYSTICK	游戏杆子系统，自动包含Event子系统
SDL_INIT_HAPTIC	触觉（力反馈）子系统
SDL_INIT_GAMECONTROLLER	游戏控制子系统，自动包含Joystick子系统
SDL_INIT_EVERYTHING	所有子系统
SDL_INIT_NOPARACHUTE	兼容性考虑，忽略

返回值	含义
0	成功
<0	失败。可调用 SDL_GetError() 获取相关信息。

详细资料参考 SDL2官网文档(SDL_Init)

使用例程：

```
#include "SDL.h"
int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) !=  0) {
        SDL_Log( "Unable to initialize SDL: %s" , SDL_GetError());
        return 1;
    }
    /* ... */
    SDL_Quit();
    return 0;
}
```

```
可以在需要的时候调用 **SDL_InitSubSystem()** 初始化所要的子系统（使用想对应的 flags 参数）
```

函数SDL_CreateWindow：创建窗口

```
SDL_Window* SDL_CreateWindow(const char* title,
        int         x,
        int         y,
        int         w,
        int         h,
        Uint32      flags)
```

返回值：若成功创建窗口，则返回指向窗口的指针；否则返回NULL。

参数	说明
title	窗口的标题（UTF-8 encoding）
x	左上角 x - 坐标，或SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_UNDEFINED
y	左上角 y - 坐标，或SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_UNDEFINED
w	窗口的宽度（像素）
h	窗口的高度（像素）
flags	0, 或者 SDL_WindowFlags 的组合（按位或操作）

详细资料参考 SDL2官网文档(SDL_CreateWindow)

编程示范：

```
#include "SDL.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
    SDL_Window *window;                    // Declare a pointer

    SDL_Init(SDL_INIT_VIDEO);              // Initialize SDL2

    // Create an application window with the following settings:
    window = SDL_CreateWindow(
        "An SDL2 window",                  // window title
        SDL_WINDOWPOS_UNDEFINED,           // initial x position
        SDL_WINDOWPOS_UNDEFINED,           // initial y position
        640,                               // width, in pixels
        480,                               // height, in pixels
        SDL_WINDOW_OPENGL                  // flags - see below
        );

    // Check that the window was successfully created
    if (window == NULL) {
        // In the case that the window could not be made...
        printf("Could not create window: %s\n", SDL_GetError());
        return 1;
    }

    // The window is open: could enter program loop here (see SDL_PollEvent())

    SDL_Delay(3000);  // Pause execution for 3000 milliseconds, for example

    // Close and destroy the window
    SDL_DestroyWindow(window);

    // Clean up
    SDL_Quit();
    return 0;
}
```

创建窗口flags可以是以下选项的组合（按位或操作）

falgs选项	含义
SDL_WINDOW_FULLSCREEN	全屏窗口
SDL_WINDOW_HIDDEN	不可见窗口
SDL_WINDOW_BORDERLESS	无边框窗口
SDL_WINDOW_RESIZABLE	大小可调节窗口
SDL_WINDOW_MINIMIZED	最小化的窗口
SDL_WINDOW_MAXIMIZED	最大化的窗口
SDL_WINDOW_INPUT_GRABBED	创建后获得接受输入

函数SDL_DestroyWindow：销毁窗口

```
void SDL_DestroyWindow(SDL_Window*  window)
```

函数SDL_CreateRenderer：创建渲染器

```
SDL_Renderer* SDL_CreateRenderer(SDL_Window* window,
    int     index,
    Uint32  flags)
```

返回值：若成功创建，则返回渲染器的指针；否则返回NULL。

flags选项	含义
window	渲染器所对应的窗口指针
index	指定渲染器的驱动, or -1 指定第一个可用的驱动
flags	0, or 渲染器选项组合

详细资料参考 SDL2官网文档(SDL_CreateRenderer)

使用实例：

```
#include "SDL.h"

int main(int argc, char *argv[])
{
    SDL_Window *win =  NULL;
    SDL_Renderer *renderer =  NULL;
    SDL_Texture *bitmapTex =  NULL;
    SDL_Surface *bitmapSurface =  NULL;
    int posX = 100, posY = 100, width = 320, height = 240;

    SDL_Init(SDL_INIT_VIDEO);
    win = SDL_CreateWindow( "Hello World", posX, posY, width, height,  0);
    renderer = SDL_CreateRenderer(win, - 1, SDL_RENDERER_ACCELERATED);

    bitmapSurface = SDL_LoadBMP( "img/hello.bmp" );
    bitmapTex = SDL_CreateTextureFromSurface(renderer, bitmapSurface);
    SDL_FreeSurface(bitmapSurface);

    while (1) {
        SDL_Event e;
        if (SDL_PollEvent(&e)) {
            if (e.type == SDL_QUIT) {
                break;
            }
```

```
        }

        SDL_RenderClear(renderer);
        SDL_RenderCopy(renderer, bitmapTex,  NULL, NULL);
        SDL_RenderPresent(renderer);
    }

    SDL_DestroyTexture(bitmapTex);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(win);

    SDL_Quit();

    return 0;
}
```

flags选项	含义
SDL_RENDERER_SOFTWARE	选择最基本软件渲染器
SDL_RENDERER_ACCELERATED	选择硬件加速的渲染器
SDL_RENDERER_PRESENTVSYNC	内容呈现与刷新率一致
SDL_RENDERER_TARGETTEXTURE	支持渲染到纹理

注意：如果不给出flags选项，即flags=0，那么将优先选择SDL_RENDERER_ACCELERATED。

相关函数

- SDL_CreateSoftwareRenderer
- SDL_GetRendererInfo
- SDL_DestroyRenderer
- SDL_GetNumRenderDrivers

函数 SDL_LoadBMP: 载入bmp格式的图像文件

```
SDL_Surface* SDL_LoadBMP(const char* file)
```

参数：file - 图像文件的文件名
返回值：如果成功载入，那么返回一个包含该图像的像面；否则返回NULL。

通过该函数创建的SDL_Surface必须通过函数SDL_FreeSurface释放：

```
void SDL_FreeSurface(SDL_Surface* surface)
```

相关函数：

- SDL_FreeSurface (https://wiki.libsdl.org/SDL_FreeSurface)
- SDL_SaveBMP (https://wiki.libsdl.org/SDL_SaveBMP)

函数 SDL_SaveBMP：将一个像面内容保存到一个bmp格式的文件中

```
int SDL_SaveBMP(SDL_Surface* surface,
                const char*  file)
```

SDL常用的数据类型

- **SDL_Surface** 是一个结构，表示一个矩形形状的像素集合，用于渲染（软件方式）

成员变量表

类型	变量名	说明
Uint32	flags	内部使用
SDL_PixelFormat*	format	像素格式（参考SDL_PixelFormat）
int	w, h	矩形的宽度和高度
int	pitch	一行像素所占用的字节数
void*	pixels	存储像素的内存地址
void*	userdata	保存用户数据的地址
int	locked	内部使用（与锁定操作有关）
void*	lock_data	内部使用（与锁定操作有关）
SDL_Rect	clip_rect	裁剪矩形（通过函数SDL_SetClipRect()指定）
SDL_BlitMap*	map	内部使用（与绘制有关）
int	refcount	被引用的次数

详细资料参考 https://wiki.libsdl.org/SDL_Surface

相关函数：

```
- SDL_ConvertSurface
- SDL_CreateRGBSurface
- SDL_CreateRGBSurfaceFrom
- SDL_FillRect
- SDL_FillRects
- SDL_FreeSurface
- SDL_GetClipRect
- SDL_GetColorKey
- SDL_GetSurfaceAlphaMod
- SDL_GetSurfaceBlendMode
- SDL_GetSurfaceColorMod
- SDL_LoadBMP_RW
- SDL_LockSurface
- SDL_LowerBlit
- SDL_MUSTLOCK
- SDL_SaveBMP_RW
- SDL_SetClipRect
- SDL_SetColorKey
- SDL_SetSurfaceAlphaMod
- SDL_SetSurfaceBlendMode
- SDL_SetSurfaceColorMod
- SDL_SetSurfacePalette
- SDL_SetSurfaceRLE
- SDL_SoftStretch
- SDL_UnlockSurface
- SDL_UpperBlit
```

函数SDL_CreateTextureFromSurface：创建纹理

```
SDL_Texture* SDL_CreateTextureFromSurface(SDL_Renderer* renderer,
                                          SDL_Surface*  surface)
```

参数名	说明
renderer	纹理所在的渲染器

参数名	说明
surface	创建纹理所用到的像面

返回值：如果成功，则返回所创建的纹理指针；否则返回NULL。失败时，可调用SDL_GetError获取有关信息。该函数不会改变surface所指向的像面。纹理使用完毕后，应该调用函数SDL_DestroyTexture释放掉。

代码示例：

```
Uint32 rmask, gmask, bmask, amask;
/* SDL interprets each pixel as a 32-bit number, so our masks must depend
   on the endianness (byte order) of the machine */
#if SDL_BYTEORDER == SDL_BIG_ENDIAN
    rmask = 0xff000000;
    gmask = 0x00ff0000;
    bmask = 0x0000ff00;
    amask = 0x000000ff;
#else
    rmask = 0x000000ff;
    gmask = 0x0000ff00;
    bmask = 0x00ff0000;
    amask = 0xff000000;
#endif

SDL_Surface *surface = SDL_CreateRGBSurface( 0, 640, 480, 32, rmask, gmask, bmask, amask);

if (surface == NULL) {
    fprintf(stderr, "CreateRGBSurface failed: %s\n" , SDL_GetError());
    exit(1);
}

SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);

if (texture == NULL) {
    fprintf(stderr, "CreateTextureFromSurface failed: %s\n" , SDL_GetError());
    exit(1);
}

SDL_FreeSurface(surface);
surface = NULL;
```

相关函数：

```
- SDL_CreateTexture
- SDL_DestroyTexture
- SDL_QueryTexture
```

函数SDL_SetRenderDrawColor：设置渲染器的画笔颜色

```
int SDL_SetRenderDrawColor (SDL_Renderer* renderer,
        Uint8         r,
        Uint8         g,
        Uint8         b,
        Uint8         a)
```

参数名	说明
renderer	渲染器
r	红色通道值
g	绿色通道值
b	蓝色通道值

参数名	说明
a	alpha通道的值

调用函数**SDL_SetRenderDrawBlendMode**设置alpha通道的使用规则。

使用示例：

```
SDL_Rect rectangle;
rectangle.x = 0;
rectangle.y = 0;
rectangle.w = 50;
rectangle.h = 50;
SDL_SetRenderDrawColor(renderer,  255, 0, 0, 255);
SDL_RenderFillRect(renderer, &rectangle);
```

相关函数：

```
- SDL_GetRenderDrawColor
- SDL_RenderClear
- SDL_RenderDrawLine
- SDL_RenderDrawLines
- SDL_RenderDrawPoint
- SDL_RenderDrawPoints
- SDL_RenderDrawRect
- SDL_RenderDrawRects
- SDL_RenderFillRect
- SDL_RenderFillRects
```

函数 SDL_RenderCopy

将纹理中的部分内容复制到渲染目标中

```
int SDL_RenderCopy(SDL_Renderer*   renderer,
        SDL_Texture*    texture,
        const SDL_Rect* srcrect,
        const SDL_Rect* dstrect)
```

参数名	说明
renderer	渲染器
texture	纹理
srcrect	纹理中待复制的矩形区域，或者NULL表示整个纹理
dstrect	目标区域矩形，或则NULL表示整个渲染目标（纹理将被拉伸）

返回值：0 - 表示成功; <0 表示失败。可调用SDL_GetError()获取相关信息。

```
#include "SDL.h"
#define SHAPE_SIZE 16

int main(int argc, char *argv[])
{
    SDL_Window* Main_Window;
    SDL_Renderer* Main_Renderer;
    SDL_Surface* Loading_Surf;
    SDL_Texture* Background_Tx;
    SDL_Texture* BlueShapes;

    /* Rectangles for drawing which will specify source (inside the texture)
       and target (on the screen) for rendering our textures. */
```

```
    SDL_Rect SrcR;
    SDL_Rect DestR;

    SrcR.x = 0;
    SrcR.y = 0;
    SrcR.w = SHAPE_SIZE;
    SrcR.h = SHAPE_SIZE;

    DestR.x = 640 / 2 - SHAPE_SIZE / 2;
    DestR.y = 580 / 2 - SHAPE_SIZE / 2;
    DestR.w = SHAPE_SIZE;
    DestR.h = SHAPE_SIZE;


    /* Before we can render anything, we need a window and a renderer */
    Main_Window = SDL_CreateWindow( "SDL_RenderCopy Example" ,
            SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,   640, 580, 0);
    Main_Renderer = SDL_CreateRenderer(Main_Window, - 1, SDL_RENDERER_ACCELERATED);

    /* The loading of the background texture. Since SDL_LoadBMP() returns
       a surface, we convert it to a texture afterwards for fast accelerated
       blitting. */
    Loading_Surf = SDL_LoadBMP( "Background.bmp" );
    Background_Tx = SDL_CreateTextureFromSurface(Main_Renderer, Loading_Surf);
    SDL_FreeSurface(Loading_Surf);  /* we got the texture now -> free surface */

    /* Load an additional texture */
    Loading_Surf = SDL_LoadBMP( "Blueshapes.bmp" );
    BlueShapes = SDL_CreateTextureFromSurface(Main_Renderer, Loading_Surf);
    SDL_FreeSurface(Loading_Surf);

    /* now onto the fun part.
       This will render a rotating selection of the blue shapes
       in the middle of the screen */
    int i;
    int n;
    for (i = 0; i < 2; ++i) {
        for(n = 0; n < 4; ++n) {
            SrcR.x = SHAPE_SIZE * (n %  2);
            if (n > 1) {
                SrcR.y = SHAPE_SIZE;
            } else {
                SrcR.y =  0;
            }

            /* render background, whereas NULL for source and destination
               rectangles just means "use the default" */
            SDL_RenderCopy(Main_Renderer, Background_Tx,   NULL, NULL);

            /* render the current animation step of our shape */
            SDL_RenderCopy(Main_Renderer, BlueShapes, &SrcR, &DestR);
            SDL_RenderPresent(Main_Renderer);
            SDL_Delay( 500);
        }
    }


    /* The renderer works pretty much like a big canvas:
       when you RenderCopy() you are adding paint, each time adding it
       on top.
       You can change how it blends with the stuff that
       the new data goes over.
       When your 'picture' is complete, you show it
       by using SDL_RenderPresent(). */

    SDL_DestroyTexture(BlueShapes);
    SDL_DestroyTexture(Background_Tx);
    SDL_DestroyRenderer(Main_Renderer);
    SDL_DestroyWindow(Main_Window);
    SDL_Quit();
```

```
    return 0;
}
```

附加说明：

- 复制过程中，纹理将和目标区域的内容混合在一起，混合的模式可以通过调用函数SDL_SetTextureBlendMode()进行设置。
- 纹理的颜色值在混合之前将根据颜色调制模式进行调制。调制模式可以通过调用函数SDL_SetTextureColorMod()进行设置。
- 纹理的alpha值在混合之前将根据alpha调制模式进行调制。调制模式可以通过调用函数SDL_SetTextureAlphaMod()进行设置。

相关函数

```
- SDL_SetTextureAlphaMod
- SDL_SetTextureBlendMode
- SDL_SetTextureColorMod
```

函数SDL_RenderClear：清除渲染器的内容（即清屏）。

```
int SDL_RenderClear(SDL_Renderer* renderer)
```

清屏的颜色为当前画笔的颜色（通过调用函数SDL_SetRenderDrawColor设置）

一些绘制函数

```
// 在像素(x,y)处画一个点
int SDL_RenderDrawPoint(SDL_Renderer* renderer,
        int           x,
        int           y);
// 画count个点，点的坐标连续保存在指针points所指向的内存中
int SDL_RenderDrawPoints(SDL_Renderer*    renderer,
        const SDL_Point* points,
        int              count);
// 从(x1,y1)到(x2,y2)画一条直线段
int SDL_RenderDrawLine(SDL_Renderer* renderer,
        int           x1,
        int           y1,
        int           x2,
        int           y2);
// 画一条折线段，包含count-1段
// 第i段的端点为points[i]和points[i+1], i=0,...,count-1
int SDL_RenderDrawLines(SDL_Renderer*    renderer,
        const SDL_Point* points,
        int              count);
// 画一个矩形框（不填充内部）
int SDL_RenderDrawRect(SDL_Renderer*   renderer,
        const SDL_Rect* rect);
// 画count个矩形框（不填充内部）
int SDL_RenderDrawRects(SDL_Renderer*   renderer,
        const SDL_Rect* rects,
        int             count);
// 画一个填充的矩形
int SDL_RenderFillRect(SDL_Renderer*   renderer,
        const SDL_Rect* rect);
// 画count个填充的矩形
int SDL_RenderFillRects(SDL_Renderer*   renderer,
        const SDL_Rect* rects,
        int             count);
// 将渲染器中的内容呈现在渲染目标上（一般是窗口）
void SDL_RenderPresent(SDL_Renderer* renderer)
```

代码示例：

```
#include "SDL.h"

int main(int argc, char* argv[])
{
    SDL_bool done = SDL_FALSE;
    SDL_Window* window =  NULL;
    SDL_Renderer* renderer =  NULL;

    SDL_Init(SDL_INIT_VIDEO);
    SDL_CreateWindowAndRenderer( 640, 480, 0, &window, &renderer)

    SDL_SetRenderDrawColor(renderer,  0, 0, 0, SDL_ALPHA_OPAQUE);
    SDL_RenderClear(renderer);
    SDL_SetRenderDrawColor(renderer,  255, 255, 255, SDL_ALPHA_OPAQUE);
    SDL_RenderDrawLine(renderer,  320, 200, 300, 240);
    SDL_RenderDrawLine(renderer,  300, 240, 340, 240);
    SDL_RenderDrawLine(renderer,  340, 240, 320, 200);
    SDL_RenderPresent(renderer);

    while (!done) {
        SDL_Event event;
        while (SDL_PollEvent(&event)) {
            if (event.type == SDL_QUIT) {
                done = SDL_TRUE;
            }
        }
    }

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

函数 SDL_SetRenderDrawBlendMode

```
int SDL_SetRenderDrawBlendMode (SDL_Renderer* renderer,
                                SDL_BlendMode blendMode)
```

参数blendMode的选项

blendMode选项	选项说明
SDL_BLENDMODE_NONE	不做混合
	dstRGBA = srcRGBA
SDL_BLENDMODE_BLEND	alpha 混合
	dstRGB = (srcRGB *srcA)* + *(dstRGB* (1-srcA))
	dstA = srcA + (dstA * (1-srcA))
SDL_BLENDMODE_ADD	叠加混合
	dstRGB = (srcRGB * srcA) + dstRGB
	dstA = dstA
SDL_BLENDMODE_MOD	颜色调和
	dstRGB = srcRGB * dstRGB
	dstA = dstA

- **SDL_Rect** 表示一个矩形区域

```
typedef struct {
  int x; // 左上角的x-坐标
  int y; // 左上角的y-坐标
  int w; // 宽度
  int h; // 高度
} SDL_Rect;
```

类型	名称	说明
int	x	the x location of the rectangle's upper left corner
int	y	the y location of the rectangle's upper left corner
int	w	the width of the rectangle
int	h	the height of the rectangle