SDL介绍

SDL2，也称为Simple DirectMedia Layer 2，是一种流行的跨平台开发库，用于创建多媒体应用程序和游戏。它提供了一个简单的API，用于处理诸如窗口创建、事件处理、图形渲染、音频播放等各种任务。

SDL2广泛支持，可以在许多平台上运行，包括Windows、macOS、Linux、iOS和Android。它是用C编写的，但提供了多种编程语言的绑定，例如C++、C#、Python等，使得不同编程背景的开发人员都能够使用它。

SDL2的特点包括：

- 窗口管理：SDL2允许您创建窗口并处理窗口事件，例如调整大小、最小化和最大化。
- 输入处理：它提供了处理输入设备（如键盘、鼠标和游戏控制器）的抽象层。您可以轻松检测到不同类型的输入事件，如按键、鼠标移动和按钮点击。
- 图形渲染：SDL2支持使用硬件加速的API（如OpenGL和Direct3D）进行2D和3D图形渲染。它提供了创建渲染上下文、加载和渲染纹理以及处理转换功能。
- 音频播放：SDL2支持音频播放和录制。它提供了一个API，用于加载和播放各种音频格式，并能处理基本的音频效果，如混音、音量控制和播放位置控制。
- 定时和多线程：SDL2包括一个计时器API，用于精确的时间控制，并提供了线程功能，用于处理并发操作。
- 文件I/O和图像加载：SDL2提供了用于文件输入/输出操作的函数，并支持各种图像格式，使您能够在应用程序中加载和显示图像。

SDL2以其简单性和高效性而闻名，常被游戏开发人员和多媒体应用程序开发人员用于创建跨平台应用程序和游戏。它为处理底层任务提供了坚实的基础，使开发人员能够专注于项目的高层方面。要开始使用SDL2，您可以访问官方SDL网站（https://www.libsdl.org/）下载库、获取文档

SDL2（Simple DirectMedia Layer 2）是SDL（Simple DirectMedia Layer）的后续版本，它们之间存在一些区别。下面是SDL2和SDL之间的一些主要区别：

- API改进：SDL2对API进行了改进和扩展，提供了更多功能和选项。它引入了新的功能，例如渲染器、窗口管理和全屏支持等。
- 多窗口支持：SDL2引入了对多个窗口的原生支持。您可以方便地创建和管理多个窗口，并为每个窗口分别处理事件和渲染。
- 渲染器系统：SDL2引入了渲染器系统，通过使用硬件加速接口，如OpenGL和Direct3D，提供更高效和灵活的图形渲染功能。
- 强化的事件处理：SDL2提供了更丰富和灵活的事件处理功能。
- 线程和定时器改进：SDL2改进了线程和定时器的支持。

总的来说，SDL2是在SDL基础上改进和扩展而来的，它提供了更多功能和选项，使得开发者能够更方便地创建跨平台的多媒体应用程序和游戏。如果您打算开始使用SDL，推荐使用SDL2，因为它提供了更先进和全面的特性集。

SDL配置

下载SDL

网址:https://github.com/libsdl-org/SDL/releases

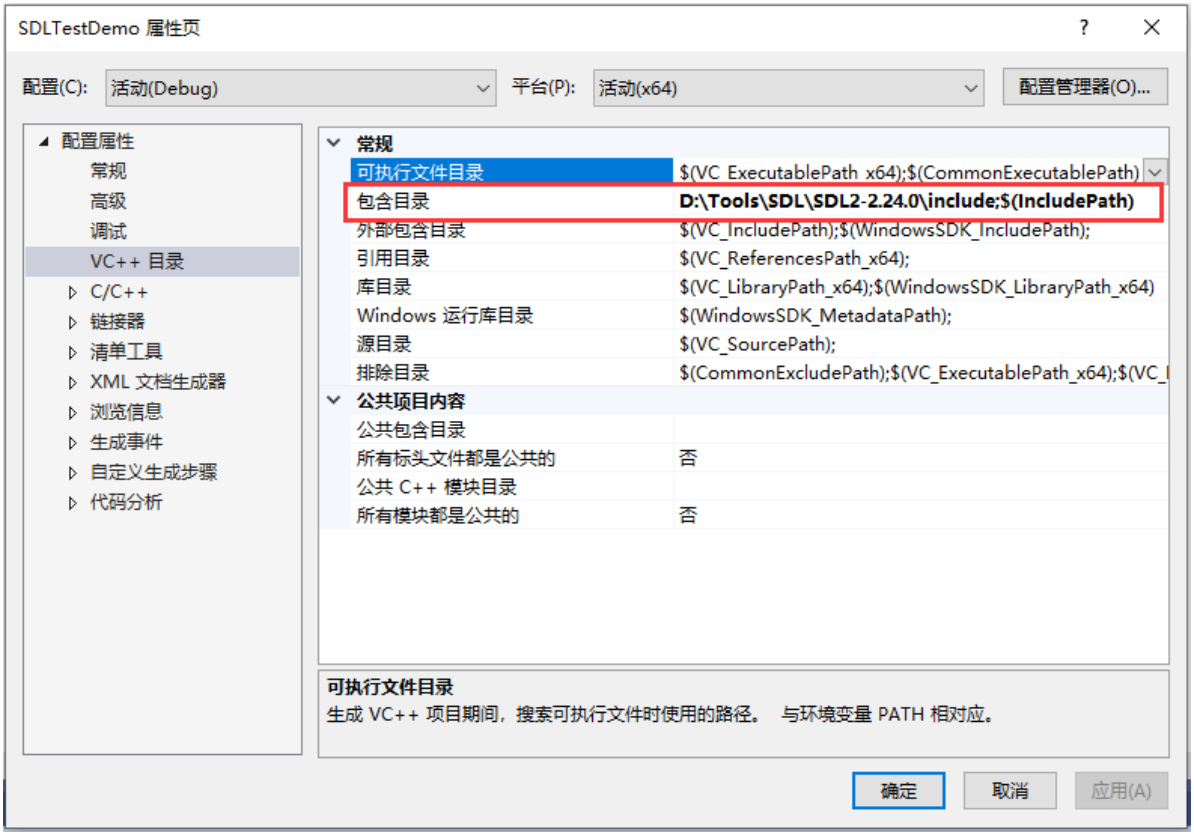
进不去的同学群里工具文件中下载即可

- Fixes and improvements to SDL_LoadObject() functionality

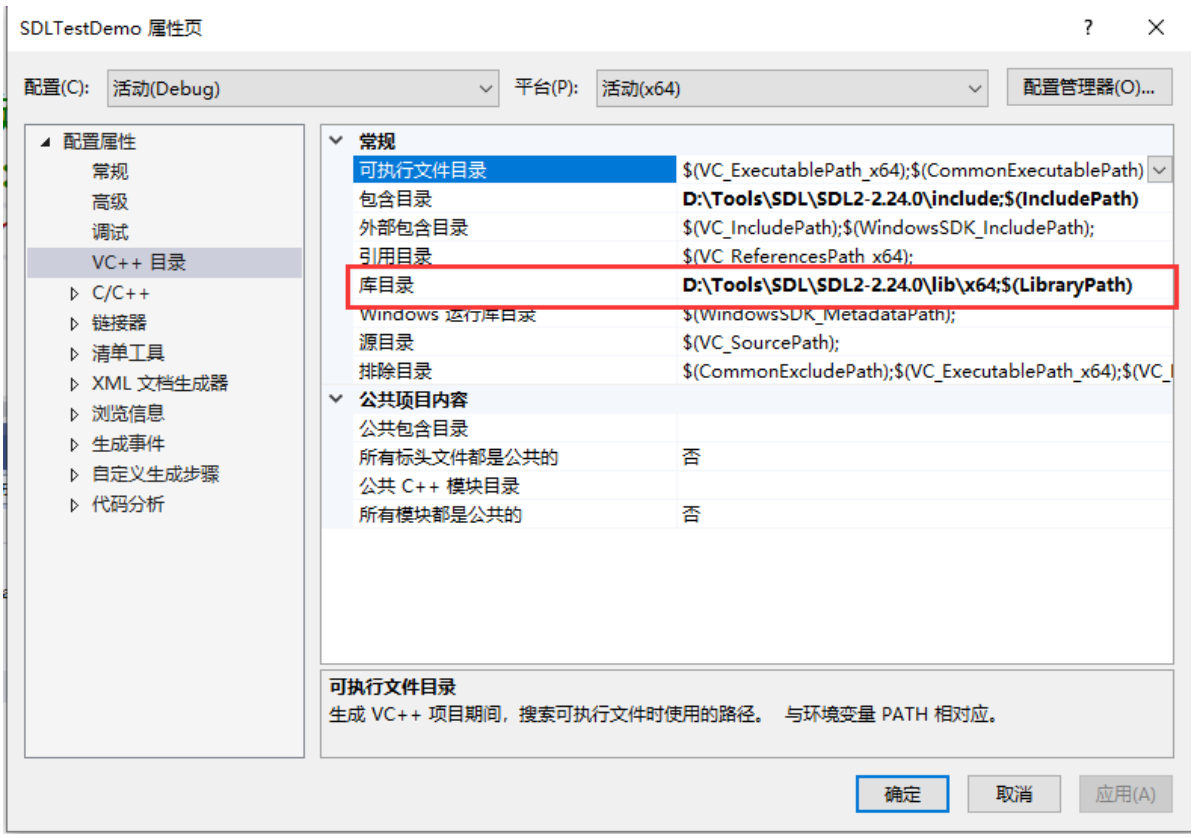
▼ Assets 12

SDL2-2.24.2-win32-x64.zip	778 KB	4 days ago
SDL2-2.24.2-win32-x86.zip	682 KB	4 days ago
SDL2-2.24.2.dmg	1.89 MB	4 days ago
SDL2-2.24.2.tar.gz	7.18 MB	4 days ago
SDL2-2.24.2.tar.gz.sig	95 Bytes	4 days ago
SDL2-2.24.2.zip	8.42 MB	4 days ago
SDL2-2.24.2.zip.sig	95 Bytes	4 days ago
SDL2-devel-2.24.2-mingw.tar.gz	14.2 MB	4 days ago
SDL2-devel-2.24.2-mingw.zip	14.3 MB	4 days ago
SDL2-devel-2.24.2-VC.zip	2.49 MB	4 days ago
Source code (zip)		4 days ago
Source code (tar.gz)		4 days ago

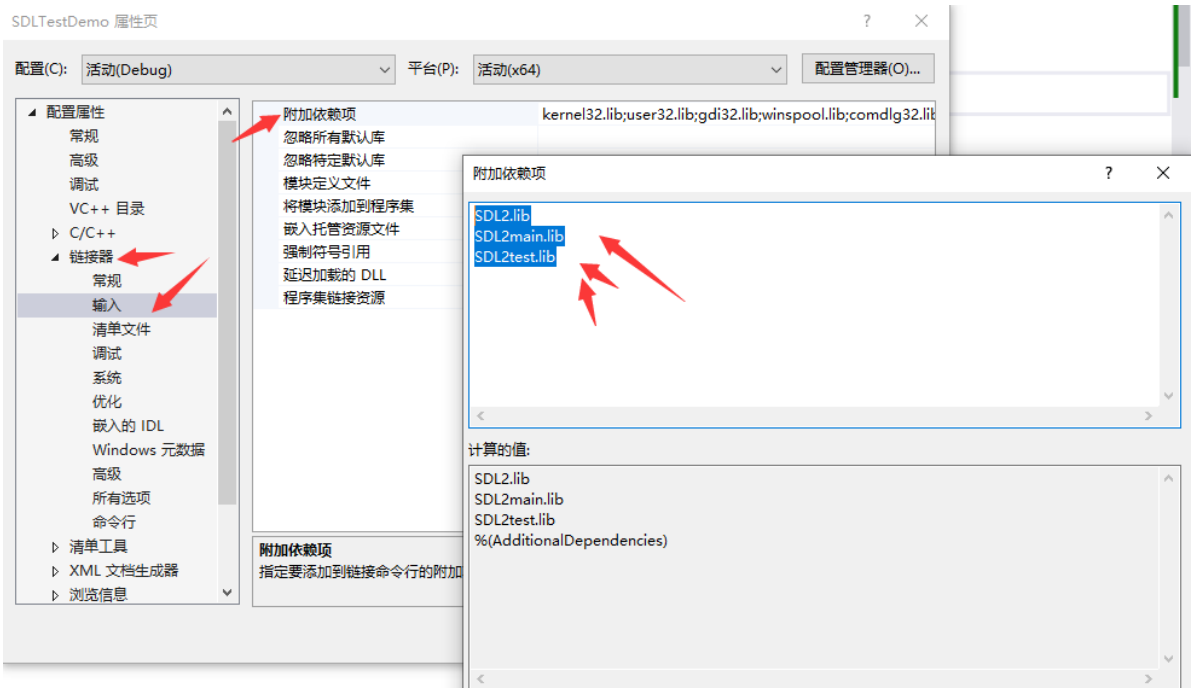
包含目录



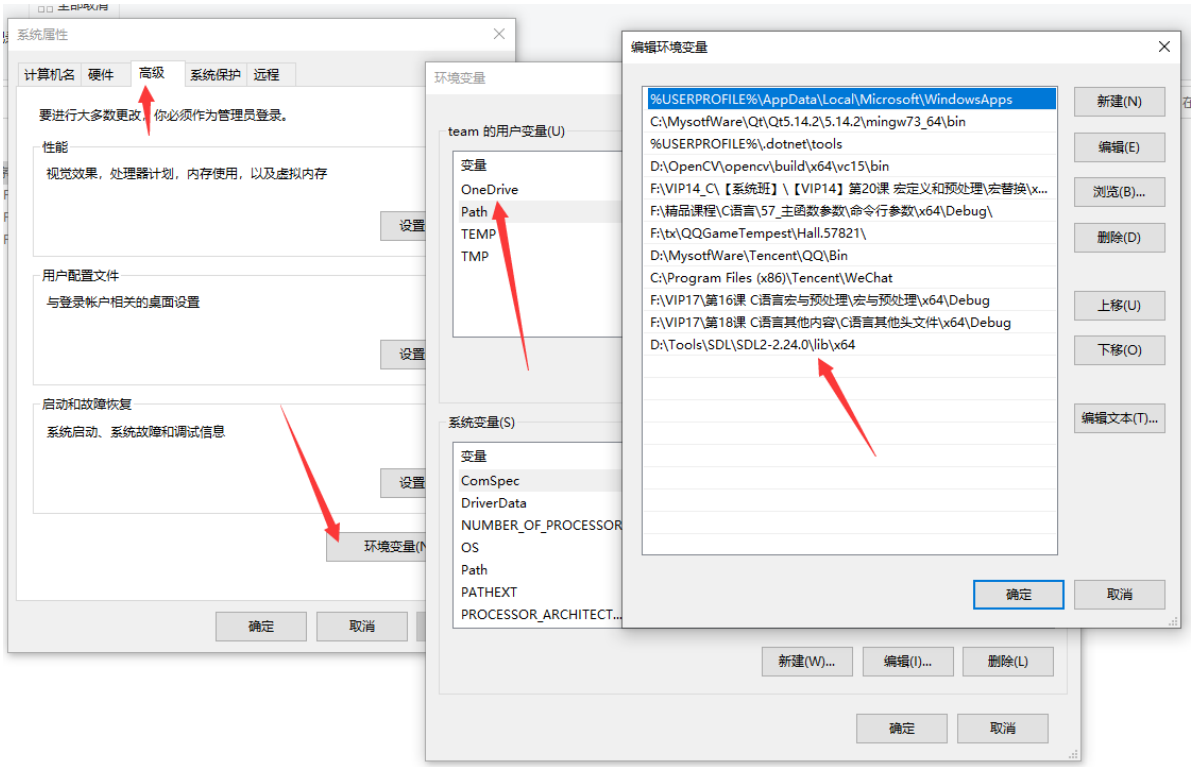
包含库目录



添加依赖项

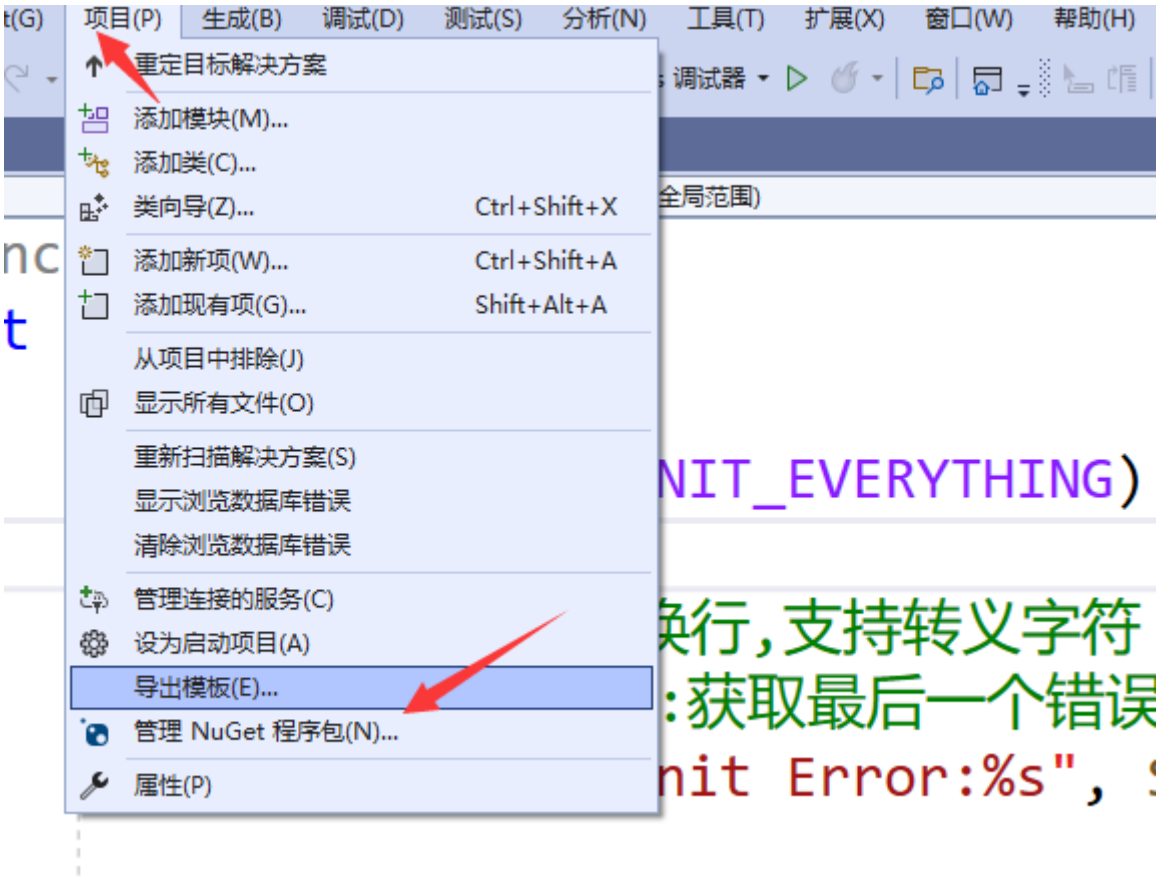


DLL环境变量



注意点： 重启vs

导出为模板



测试代码

```
#include <SDL.h>
int main()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
    return 0;
}
```

SDL窗口创建

SDL基本流程

SDL程序分为三部分:

- 初始化过程
- 渲染过程
- 释放过程

SDL初始化API

- SDL_Init : 初始化SDL
- SDL_CreateWindow(): 创建窗口
- SDL_CreateRenderer(): 创建渲染器
- SDL_CreateTexture(): 创建纹理---->图片

SDL渲染API

- SDL_UpdateTexture(): 更新纹理
- SDL_RenderCopy(): 纹理复制给渲染器
- SDL_RenderPresent(): 显示

SDL释放API

- SDL_DestroyWindow(): 释放窗口
- SDL_DestroyTexture(): 释放纹理
- SDL_DestroyRenderer(): 释放渲染器
- SDL_Quit(): 关闭SDL所有内容

窗口创建

SDL_Init函数

```
extern DECLSPEC int SDLCALL SDL_Init(Uint32 flags);
/*
SDL_INIT_EVERYTHING: 所有的资源
SDL_INIT_TIMER:      定时器
SDL_INIT_AUDIO:      音频
SDL_INIT_VIDEO:      视频
SDL_INIT_JOYSTICK:   遥杆
SDL_INIT_HAPTIC:     触摸屏
SDL_INIT_GAMECONTROLLER: 游戏控制
SDL_INIT_EVENTS:         事件
SDL_INIT_SENSOR:     传感器
SDL_INIT_NOPARACHUTE    不捕获关键信息
返回值:小于0初始化失败
*/
```

SDL_CreateWindow函数

```
extern DECLSPEC SDL_Window * SDLCALL SDL_CreateWindow(const char *title,
                                                      int x, int y, int w,
                                                      int h, Uint32 flags);
/*
    title: 标题
    x,y: 显示位置
    w,h: 宽度和高度
    flags: 显示样式
SDL_WINDOW_FULLSCREEN            :全屏窗口
SDL_WINDOW_OPENGL                :可以与openGL组合的窗口
SDL_WINDOW_SHOWN                 :窗口可见
SDL_WINDOW_HIDDEN                : 窗口不可见
SDL_WINDOW_BORDERLESS            :没有窗口装饰
SDL_WINDOW_RESIZABLE             :可改变窗口大小
SDL_WINDOW_MINIMIZED             :窗口最小化
SDL_WINDOW_MAXIMIZED             :窗口最大化
SDL_WINDOW_MOUSE_GRABBED
SDL_WINDOW_INPUT_FOCUS
SDL_WINDOW_MOUSE_FOCUS
SDL_WINDOW_FULLSCREEN_DESKTOP
SDL_WINDOW_FOREIGN
SDL_WINDOW_ALLOW_HIGHDPI
*/
```

注意点: 防止闪屏不能用死循环，事件处理会程序会崩溃,用SDL提供的延时函数 SDL_Delay(1000);

绘图流程

创建渲染器

SDL_CreateRenderer

```
extern DECLSPEC SDL_Renderer * SDLCALL SDL_CreateRenderer(SDL_Window * window,
                                                          int index, Uint32 flags);
/*
    window:  渲染器是那个窗口
    index: 渲染设备  -1 默认
    flags:

 SDL_RENDERER_SOFTWARE: 软件渲染
 SDL_RENDERER_ACCELERATED :硬件加速
 SDL_RENDERER_PRESENTVSYNC: 显示器同步刷新率
*/
```

创建纹理

SDL_CreateTextrue

```
extern DECLSPEC SDL_Texture * SDLCALL SDL_CreateTexture(SDL_Renderer * renderer,
                                                        Uint32 format,
                                                        int access,
                                                        int w,
                                                        int h);
extern DECLSPEC int SDLCALL SDL_UpdateTexture(SDL_Texture * texture,
                                              const SDL_Rect * rect,
                                              const void *pixels, int pitch);
```

复制纹理到渲染目标

SDL_RenderCopy()

```
extern DECLSPEC int SDLCALL SDL_RenderCopy(SDL_Renderer * renderer,
                                           SDL_Texture * texture,
                                           const SDL_Rect * srcrect,
                                           const SDL_Rect * dstrect);
```

显示画面

SDL_RenderPresent

```
extern DECLSPEC void SDLCALL SDL_RenderPresent(SDL_Renderer * renderer);
```

整体流程

1.初始化

2.创建窗口

3.创建渲染器

4.创建纹理

5.更新纹理

6.渲染纹理

7.显示图像

8.返回第4步，循环操作

综合代码

```
#include <SDL.h>
int main()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());
    }
    SDL_Window* window = SDL_CreateWindow("SDL", 300, 100, 800, 600,
SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    SDL_Renderer* render = SDL_CreateRenderer(window, -1,
SDL_RENDERER_SOFTWARE);
    //颜色：  四个数字   透明色
    SDL_SetRenderDrawColor(render, 0, 255, 0, 255);  //设置渲染颜色
    SDL_RenderClear(render);                         //刷新
    SDL_RenderPresent(render);                       //显示

    SDL_Delay(3000);
    return 0;
}
```

SDL基本绘图函数

常用绘图函数

SDL提供了一些基本的绘图函数，可以用来创建简单的2D图形和绘制图像：

```
SDL_SetRenderDrawColor：设置渲染器的绘制颜色。
SDL_RenderClear：清空渲染器。
SDL_RenderDrawPoint：在渲染器上绘制一个点。
SDL_RenderDrawLine：在渲染器上绘制一条直线。
SDL_RenderDrawRect：在渲染器上绘制一个矩形。
SDL_RenderFillRect：在渲染器上绘制一个填充矩形。
SDL_RenderCopy：将纹理渲染到目标矩形中。
SDL_RenderPresent：将渲染器的绘制结果输出到屏幕上。
```

画笔颜色

SDL_SetRenderDrawColor

```
extern DECLSPEC int SDLCALL SDL_SetRenderDrawColor(SDL_Renderer * renderer,
                                                   Uint8 r, Uint8 g, Uint8 b,
                                                   Uint8 a);
```

画点

SDL_RenderDrawPoint和SDL_RenderDrawPoints

```
extern DECLSPEC int SDLCALL SDL_RenderDrawPoint(SDL_Renderer * renderer,
                                                int x, int y);
extern DECLSPEC int SDLCALL SDL_RenderDrawPoints(SDL_Renderer * renderer,
                                                 const SDL_Point * points,
                                                 int count);
```

画线

SDL_RenderDrawLine和SDL_RenderDrawLines

```
extern DECLSPEC int SDLCALL SDL_RenderDrawLine(SDL_Renderer * renderer,
                                               int x1, int y1, int x2, int y2);
extern DECLSPEC int SDLCALL SDL_RenderDrawLines(SDL_Renderer * renderer,
                                                const SDL_Point * points,
                                                int count);
```

画矩形

SDL_RenderDrawRect和SDL_RenderDrawRects和SDL_RenderFillRect填充矩形

```
extern DECLSPEC int SDLCALL SDL_RenderDrawRect(SDL_Renderer * renderer,
                                               const SDL_Rect * rect);
extern DECLSPEC int SDLCALL SDL_RenderDrawRects(SDL_Renderer * renderer,
                                                const SDL_Rect * rects,
                                                int count);
extern DECLSPEC int SDLCALL SDL_RenderFillRect(SDL_Renderer * renderer,
                                               const SDL_Rect * rect);
```

自制绘制函数

```
//画圆
void SDL_RenderDrawEllipse(SDL_Renderer* render, SDL_Rect* rect)
{
    //半轴长
    int aHalf = rect->w / 2;
    int bHalf = rect->h / 2;
    int x, y;
    for (float angle = 0; angle < 360; angle += 0.1f)
    {
        x = (rect->x + aHalf) + aHalf * SDL_cosf(angle);
        y = (rect->y + aHalf) + bHalf * SDL_sinf(angle);
        SDL_RenderDrawPoint(render, x, y);
    }
}
```

综合代码

```
#include <SDL.h>

void SDL_RenderDrawEllipse(SDL_Renderer* render, SDL_Rect* rect)
{
    //半轴长
    int aHalf = rect->w / 2;
    int bHalf = rect->h / 2;
    int x, y;
    for (float angle = 0; angle < 360; angle += 0.1f)
    {
        x = (rect->x + aHalf) + aHalf * SDL_cosf(angle);
        y = (rect->y + aHalf) + bHalf * SDL_sinf(angle);
        SDL_RenderDrawPoint(render, x, y);
    }
}


int main()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());
    }
    SDL_Window* window = SDL_CreateWindow("SDL", 300, 100, 800, 600,
SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    SDL_Renderer* render = SDL_CreateRenderer(window, -1,
SDL_RENDERER_SOFTWARE);

    SDL_SetRenderDrawColor(render, 255, 0, 255, 255);
    SDL_RenderDrawPoint(render, 10, 10);
    //SDL_Point 点的坐标x和y
    SDL_Point points[4] = { 20,20,30,30,40,40,50,50 };
    SDL_RenderDrawPoints(render, points, 4);

    SDL_SetRenderDrawColor(render, 255, 0, 0, 255);
    SDL_RenderDrawLine(render, 0, 0, 800, 600);
    SDL_Point points2[4] = { 20,20,150,330,170,440,190,550 };
    SDL_RenderDrawLines(render, points2, 3);
    //SDL_Rect  x和y  w和h
    SDL_Rect  rect = { 100,100,400,400 };
    SDL_RenderDrawRect(render, &rect);
    SDL_Rect fillrect = { 400,400,100,100 };
    SDL_RenderFillRect(render, &fillrect);

    SDL_Rect rects[3] = { 100,100,50,50,200,200,50,50,300,300,50,50 };
    SDL_RenderDrawRects(render, rects,3);

    SDL_Rect  circle = { 500,500,100,100 };
    SDL_RenderDrawEllipse(render, &circle);
```

```
    SDL_RenderPresent(render);                          //显示

    SDL_Delay(3000);
    return 0;
}
```

SDL文字

SDL_ttf 是一个使用 TrueType 字体渲染文本的库，它可以与SDL一起使用，用于制作游戏、应用程序等。

常用函数

综合代码

渲染贴图

```
extern DECLSPEC int SDLCALL SDL_RenderCopy(SDL_Renderer * renderer,
                                           SDL_Texture * texture,
                                           const SDL_Rect * srcrect,
                                           const SDL_Rect * dstrect);
```

```
#include <SDL.h>
#include <stdbool.h>
SDL_Window* window = NULL;
SDL_Renderer* render = NULL;

void init_SDL()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
}
void initgraph(int w, int h)
{
    window = SDL_CreateWindow("SDL", 300, 100, w, h, SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    render = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
}

int main()
{
    init_SDL();
    //SDL_surface: 图片
    //SDL_LoadBMP: 加载
```

```
    SDL_Surface* img = SDL_LoadBMP("mm.bmp");    //自带的只支持BMP
    initgraph(img->w, img->h);
    if (img == NULL)
    {
        SDL_Log("img is error!");
        return 0;
    }
    //设置透明颜色
    //SDL_SetColorKey(img, true, SDL_MapRGB(img->format, 255,255,255));
    SDL_Texture* texture = SDL_CreateTextureFromSurface(render, img);
    SDL_Rect  dst = { 0,0,img->w / 2,img->h / 2 };
    SDL_RenderCopy(render, texture, NULL, &dst);             //缩放也可以
    SDL_RenderPresent(render);

    SDL_Delay(30000);
    return 0;
}
```

直接贴图

```
#include <SDL.h>
#include <stdbool.h>
SDL_Window* window = NULL;
SDL_Renderer* render = NULL;

void init_SDL()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)           //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
}
void initgraph(int w, int h)
{
    window = SDL_CreateWindow("SDL", 300, 100, w, h, SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    render = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
}

int main()
{
    init_SDL();
    //SDL_surface: 图片
    //SDL_LoadBMP: 加载
    SDL_Surface* img = SDL_LoadBMP("mm.bmp");    //自带的只支持BMP
    initgraph(img->w, img->h);
    if (img == NULL)
    {
        SDL_Log("img is error!");
        return 0;
    }
    //设置透明颜色
    //获取窗口surface
```

```
    SDL_Surface* windowSurface = SDL_GetWindowSurface(window);
    //贴到窗口的surface
    SDL_BlitSurface(img, NULL, windowSurface, NULL);
    SDL_UpdateWindowSurface(window);

    SDL_Delay(30000);
    return 0;
}
```

SDL_Image贴图

SDL_Image是一个用于在SDL中加载图片的库。它支持多种图像格式，包括PNG、JPG、TIF、BMP等。下面是SDL_Image库中一些常用函数的介绍：

```
IMG_Init：初始化SDL_Image库。
IMG_Quit：关闭SDL_Image库。
IMG_Load：从文件或内存中加载图像文件。
IMG_LoadTexture：从文件或内存中加载图像文件，并将其转换为渲染器所需的纹理格式。
IMG_LoadTexture_RW：从文件流中加载图像文件，并将其转换为渲染器所需的纹理格式。
IMG_isICO：检查给定文件是否为.ico图标文件。
IMG_isCUR：检查给定文件是否为.cur鼠标光标文件。
IMG_isBMP：检查给定文件是否为.bmp位图文件。
IMG_isGIF：检查给定文件是否为.gif动画文件。
IMG_isJPG：检查给定文件是否为.jpg文件。
IMG_isLBM：检查给定文件是否为.lbm格式文件。
IMG_isPCX：检查给定文件是否为.pcx格式文件。
IMG_isPNG：检查给定文件是否为.png格式文件。
IMG_isPNM：检查给定文件是否为.pnm格式文件。
IMG_isTIF：检查给定文件是否为.tif格式文件。
IMG_isXCF：检查给定文件是否为.xcf格式文件。
IMG_isXPM：检查给定文件是否为.xpm格式文件。
IMG_isXV：检查给定文件是否为.xv格式文件。
IMG_GetError：获取错误消息。
```

测试代码

```
#include <SDL.h>
#include <SDL_image.h>
int main()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
    if (IMG_Init(IMG_INIT_AVIF)<0)
    {
        SDL_Log("IMG_Init error!");
    }
    SDL_Window* window = SDL_CreateWindow("SDL_image", 100, 100, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Surface* img = IMG_Load("mm.jpg");
    SDL_Surface* surface = SDL_GetWindowSurface(window);
```

```
    SDL_BlitSurface(img, NULL, surface, NULL);
    SDL_UpdateWindowSurface(window);
    SDL_Delay(4000);
    return 0;
}
```

SDL事件

SDL处理机制

SDL事件键盘事件，鼠标事件，窗口事件，SDL先把所有事件放到队列中，对事件操作就是对队列操作

SDL操作事件队列API

- SDL_PollEvent: 从事件队列抛出去
- SDL_WaitEvent: 有事件抛出事件，没有就阻塞
- SDL_WaitEventTimeout: 时间控制阻塞状态
- SDL_PeekEvent: 从队列拿事件
- SDL_PushEvent: 插入事件到队列

SDL处理事件队列API

- SDL_WIndowEvent: 窗口事件
- SDL_KeyboardEvent: 键盘事件
- SDL_MouseMotionEvent: 鼠标移动事件
- SDL_QuitEvent: 退出事件
- SDL_UserEvent: 用户自定义事件

键盘消息

```
#include <SDL.h>
#include <stdbool.h>
SDL_Window* window = NULL;
SDL_Renderer* render = NULL;
void init_SDL()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
}
void initgraph(int w, int h)
{
    window = SDL_CreateWindow("SDL", 300, 100, w, h, SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    render = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
}
//esc关闭窗口为例
void pressEscQuit()
{
```

```
    bool close = false;
    while (close == false)
    {
        //定义事件变量
        SDL_Event event;
        //获取鼠标消息
        SDL_PollEvent(&event);
        if (&event != NULL)
        {
            if (event.type == SDL_KEYDOWN)
            {
                if (event.key.keysym.sym == SDLK_ESCAPE)
                {
                    close = true;
                }
            }
        }
    }
}
int main()
{
    init_SDL();
    initgraph(800,600);
    pressEscQuit();
    //SDL_Delay(10000);
    return 0;
}
```

鼠标消息

```
#include <SDL.h>
#include <stdbool.h>
SDL_Window* window = NULL;
SDL_Renderer* render = NULL;
void init_SDL()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)            //初始化资源
    {
        //SDL打印 自带换行,支持转义字符
        //SDL_GetError:获取最后一个错误
        SDL_Log("SDL_Init Error:%s", SDL_GetError());

    }
}
void initgraph(int w, int h)
{
    window = SDL_CreateWindow("SDL", 300, 100, w, h, SDL_WINDOW_SHOWN);
    //设置窗口背景颜色
    render = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
}
//esc关闭窗口为例
void pressEscQuit()
{
    bool close = false;
    while (close == false)
    {
```

```
        //定义事件变量
        SDL_Event event;
        //获取鼠标消息
        SDL_PollEvent(&event);
        if (&event != NULL)
        {
            if (event.type == SDL_KEYDOWN)
            {
                if (event.key.keysym.sym == SDLK_ESCAPE)
                {
                    close = true;
                }
            }
        }
    }
}
//鼠标消息
void MouseEvent()
{
    bool close = false;
    while (close == false)
    {
        SDL_Event event;
        while (SDL_PollEvent(&event) != 0)
        {
            switch (event.type)
            {
            case SDL_KEYDOWN:
                SDL_Log("press any key to close");
                close = true;
                break;
            case SDL_MOUSEMOTION:
                SDL_Log("Mouse move");
                break;
            case SDL_MOUSEBUTTONDOWN:
                if (event.button.button == SDL_BUTTON_LEFT)
                {
                    SDL_Log("left button down");
                }
                if (event.button.button == SDL_BUTTON_RIGHT)
                {
                    SDL_Log("right button down");
                }
            case SDL_QUIT:                    //窗口关闭按钮的处理
                close = true;
                break;
            }
        }
    }
}
int main()
{
    init_SDL();
    initgraph(800,600);
    //pressEscQuit();
    MouseEvent();
    //SDL_Delay(10000);
    return 0;
```

```
}
```

SDL定时器

定时器

SDL定时器是一种在指定的时间间隔内触发回调函数的机制。它可以用于实现游戏中的计时器、动画等等，也可以用于其他需要定期执行某些操作的应用程序。SDL提供了一个基于事件的定时器和一个基于回调函数的定时器。

```
SDL_AddTimer:基于事件的定时器
SDL_RemoveTimer:移除定时器
```

综合代码

```
#include <SDL2/SDL.h>

Uint32 my_callback(Uint32 interval, void* param) {
    printf("Hello, SDL Timer!\n");
    return interval;
}

int main(int argc, char* argv[]) {
    if (SDL_Init(SDL_INIT_TIMER) != 0) {
        printf("SDL_Init error: %s\n", SDL_GetError());
        return 1;
    }
    SDL_TimerID timer_id = SDL_AddTimer(1000, my_callback, nullptr);
    SDL_Event event;
    while (SDL_WaitEvent(&event))
    {
        if (event.type == SDL_QUIT)
            break;
    }

    SDL_RemoveTimer(timer_id);

    SDL_Quit();
    return 0;
}
```

SDL音频

SDL_Mixer是一款用于简化音频播放的跨平台库。它是Simple DirectMedia Layer（SDL）的一部分，旨在提供一套简单易用的API，以便在游戏和其他多媒体应用程序中播放音频，包括WAV、MP3、OGG、MOD等不同的音频格式。SDL_Mixer支持多个音频通道，用户可以在不同通道播放不同的音频并控制音量、平衡和立体声等参数。除了音频播放外，SDL_Mixer还提供了一些其他功能，包括音频混合、音频效果处理和音频流处理等。它在许多游戏和多媒体应用程序中广泛使用，特别是在需要播放多个声音效果和背景音乐的游戏中。

常用函数

SDL_Mixer提供了以下常用函数：

Mix_OpenAudio：初始化SDL_Mixer库。
Mix_CloseAudio：关闭SDL_Mixer库。
Mix_LoadWAV：从文件或内存加载WAV格式的音频文件。
Mix_FreeChunk：释放由Mix_LoadWAV函数加载的音频文件。
Mix_LoadMUS：从文件或内存中加载MIDI、MOD、OGG、MP3等格式的音频文件。
Mix_FreeMusic：释放由Mix_LoadMUS函数加载的音频文件。
Mix_PlayChannel：播放一个音效，在指定的通道上播放，支持控制音量、循环次数等参数。
Mix_PlayMusic：播放一段背景音乐，支持控制循环次数、音量等参数。
Mix_Pause：暂停特定通道或所有通道上的音频。
Mix_Resume：恢复所有通道或特定通道上的音频播放。
Mix_HaltChannel：停止特定通道上的音频播放。
Mix_HaltMusic：停止背景音乐的播放。
Mix_SetMusicPosition：设置背景音乐的播放位置。
Mix_VolumeChunk：设置特定音效通道的音量。
Mix_VolumeMusic：设置背景音乐的音量。
Mix_GetChunk：获取指定通道上正在播放的音效。
Mix_GetMusicDecoder：获取正在解码的背景音乐的解码器名称。
Mix_GetError：获取错误消息。

综合代码

SDL_ttf是一个用于在SDL中渲染字体的库。它支持TrueType和OpenType字体，可以加载字体文件并将其转换为纹理格式，以在SDL渲染器上进行渲染。

常用函数

下面是SDL_ttf库中一些常用函数的介绍：

TTF_Init：初始化SDL_ttf库。
TTF_Quit：关闭SDL_ttf库。
TTF_OpenFont：打开一个字体文件并返回一个字体对象。
TTF_CloseFont：关闭字体对象。
TTF_RenderText_Solid：使用给定字体、颜色和文本渲染一个纹理图像。
TTF_RenderText_Blended：与TTF_RenderText_Solid类似，但进行了抗锯齿处理，可以更好地显示小字体。
TTF_RenderText_Shaded：使用给定字体、颜色和文本渲染一个带阴影的纹理图像。
TTF_SizeText：获取指定字体在特定文本下的渲染尺寸。
TTF_FontHeight：获取指定字体对象的高度。
TTF_SetFontStyle：设置字体样式，包括加粗、倾斜和下划线等。
TTF_GetError：获取错误消息。

综合代码

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <string>

const std::string FONT_PATH = "arial.ttf";
const int FONT_SIZE = 28;
```

```
int main(int argc, char* argv[]) {
    SDL_Init(SDL_INIT_EVERYTHING);
    TTF_Init();
    SDL_Window* window = SDL_CreateWindow("SDL_ttf demo",
SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, 640, 480, SDL_WINDOW_SHOWN);
    SDL_Renderer* renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);
    TTF_Font* font = TTF_OpenFont(FONT_PATH.c_str(), FONT_SIZE);
    SDL_Color textColor = {255, 255, 255, 255};
    SDL_Surface* textSurface = TTF_RenderText_Blended(font, "Hello, SDL_ttf!",
textColor);
    SDL_Texture* textTexture = SDL_CreateTextureFromSurface(renderer,
textSurface);
    int textWidth = textSurface->w;
    int textHeight = textSurface->h;
    SDL_Rect dstRect = {0, 0, textWidth, textHeight};
    SDL_RenderCopy(renderer, textTexture, nullptr, &dstRect);
    SDL_RenderPresent(renderer);
    SDL_Delay(3000);
    SDL_FreeSurface(textSurface);
    SDL_DestroyTexture(textTexture);
    TTF_CloseFont(font);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    TTF_Quit();
    return 0;
}
```