# 原 SDL2源代码分析4：纹理（SDL_Texture）

=====================================================

SDL源代码分析系列文章列表：

SDL2源代码分析1：初始化（SDL_Init()）

SDL2源代码分析2：窗口（SDL_Window）

SDL2源代码分析3：渲染器（SDL_Renderer）

SDL2源代码分析4：纹理（SDL_Texture）

SDL2源代码分析5：更新纹理（SDL_UpdateTexture()）

SDL2源代码分析6：复制到渲染器（SDL_RenderCopy()）

SDL2源代码分析7：显示（SDL_RenderPresent()）

SDL2源代码分析8：视频显示总结

=====================================================

上一篇文章分析了SDL中创建渲染器的函数SDL_CreateRenderer()。这篇文章继续分析SDL的源代码。本文分析SDL的纹理（SDL_Texture）。



SDL播放视频的代码流程如下所示。
**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。

上篇文章分析了该流程中的第3个函数SDL_CreateRenderer()。本文继续分析该流程中的第4个函数SDL_CreateTexture()。

## SDL_Texture

SDL_Texture结构定义了一个SDL中的纹理。如果直接使用SDL2编译好的SDK的话，是看不到SDL_Texture的内部结构的。有关它的定义在头文件中只有一行代码，如下所示。

```cpp
/**
 *  \brief An efficient driver-specific representation of pixel data
 */
struct SDL_Texture;
typedef struct SDL_Texture SDL_Texture;
```

在源代码工程中可以看到SDL_Texture的定义，位于render\SDL_sysrender.h文件中。它的定义如下。

```cpp
1.  /* Define the SDL texture structure */
2.  struct SDL_Texture
3.  {
4.      const void *magic;
5.      Uint32 format;              /**< The pixel format of the texture */
6.      int access;                 /**< SDL_TextureAccess */
7.      int w;                      /**< The width of the texture */
8.      int h;                      /**< The height of the texture */
9.      int modMode;                /**< The texture modulation mode */
10.     SDL_BlendMode blendMode;    /**< The texture blend mode */
11.     Uint8 r, g, b, a;           /**< Texture modulation values */
12.
13.
14.     SDL_Renderer *renderer;
15.
16.
17.     /* Support for formats not supported directly by the renderer */
18.     SDL_Texture *native;
19.     SDL_SW_YUVTexture *yuv;
20.     void *pixels;
21.     int pitch;
22.     SDL_Rect locked_rect;
23.
24.
25.     void *driverdata;           /**< Driver specific texture representation */
26.
27.
28.     SDL_Texture *prev;
29.     SDL_Texture *next;
30. };
```

可以看出其中包含了一个"纹理"所具备的各种属性。下面来看看如何创建这个SDL_Texture。

## SDL_CreateTexture()

### 函数简介

使用SDL_CreateTexture()基于渲染器创建一个纹理。SDL_CreateTexture()的原型如下。

```cpp
1.  SDL_Texture * SDLCALL SDL_CreateTexture(SDL_Renderer * renderer,
2.                                          Uint32 format,
3.                                          int access, int w,
4.                                          int h);
```

参数的含义如下。

renderer：目标渲染器。

format
    ：纹理的格式。后面会详述。
access
    ：可以取以下值（定义位于SDL_TextureAccess中）

SDL_TEXTUREACCESS_STATIC
    ：变化极少
SDL_TEXTUREACCESS_STREAMING
    ：变化频繁
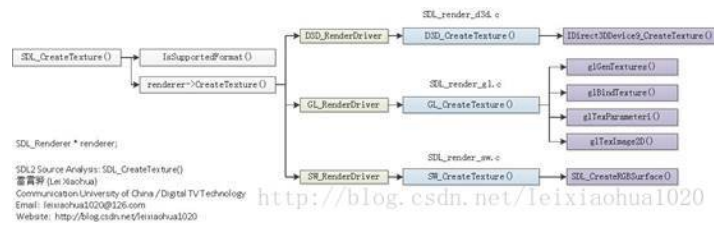SDL_TEXTUREACCESS_TARGET
    ：暂时没有理解

w
    ：纹理的宽

h
    ：纹理的高

创建成功则返回纹理的ID，失败返回0。

### 函数调用关系图

SDL_ CreateTexture ()关键函数的调用关系可以用下图表示。

上面的图片不太清晰，更清晰的图片上传到了相册里面：

http://my.csdn.net/leixiaohua1020/album/detail/1793543

把相册里面的图片保存下来就可以得到清晰的图片了。

## 源代码分析

SDL_CreateTexture()的源代码位于render\SDL_render.c中。如下所示。

```cpp
SDL_Texture * SDL_CreateTexture(SDL_Renderer * renderer, Uint32 format, int access, int w, int h)
{
    SDL_Texture *texture;


    CHECK_RENDERER_MAGIC(renderer, NULL);


    if (!format) {
        format = renderer->info.texture_formats[0];
    }
    if (SDL_ISPIXELFORMAT_INDEXED(format)) {
        SDL_SetError("Palettized textures are not supported");
        return NULL;
    }
    if (w <= 0 || h <= 0) {
        SDL_SetError("Texture dimensions can't be 0");
        return NULL;
    }
    if ((renderer->info.max_texture_width && w > renderer->info.max_texture_width) ||
        (renderer->info.max_texture_height && h > renderer->info.max_texture_height)) {
        SDL_SetError("Texture dimensions are limited to %dx%d", renderer->info.max_texture_width, renderer->info.max_texture_height);
        return NULL;
    }
    texture = (SDL_Texture *) SDL_calloc(1, sizeof(*texture));
    if (!texture) {
        SDL_OutOfMemory();
        return NULL;
    }
    texture->magic = &texture_magic;
    texture->format = format;
    texture->access = access;
    texture->w = w;
    texture->h = h;
    texture->r = 255;
    texture->g = 255;
    texture->b = 255;
    texture->a = 255;
    texture->renderer = renderer;
    texture->next = renderer->textures;
    if (renderer->textures) {
        renderer->textures->prev = texture;
    }
    renderer->textures = texture;


    if (IsSupportedFormat(renderer, format)) {
        if (renderer->CreateTexture(renderer, texture) < 0) {
            SDL_DestroyTexture(texture);
            return 0;
        }
    } else {
        texture->native = SDL_CreateTexture(renderer,
                            GetClosestSupportedFormat(renderer, format),
                            access, w, h);
        if (!texture->native) {
            SDL_DestroyTexture(texture);
            return NULL;
        }


        /* Swap textures to have texture before texture->native in the list */
        texture->native->next = texture->next;
        if (texture->native->next) {
```

```
65.              texture->native->next->prev = texture->native;
66.          }
67.          texture->prev = texture->native->prev;
68.          if (texture->prev) {
69.              texture->prev->next = texture;
70.          }
71.          texture->native->prev = texture;
72.          texture->next = texture->native;
73.          renderer->textures = texture;
74.
75.
76.          if (SDL_ISPIXELFORMAT_FOURCC(texture->format)) {
77.              texture->yuv = SDL_SW_CreateYUVTexture(format, w, h);
78.              if (!texture->yuv) {
79.                  SDL_DestroyTexture(texture);
80.                  return NULL;
81.              }
82.          } else if (access == SDL_TEXTUREACCESS_STREAMING) {
83.              /* The pitch is 4 byte aligned */
84.              texture->pitch = (((w * SDL_BYTESPERPIXEL(format)) + 3) & ~3);
85.              texture->pixels = SDL_calloc(1, texture->pitch * h);
86.              if (!texture->pixels) {
87.                  SDL_DestroyTexture(texture);
88.                  return NULL;
89.              }
90.          }
91.      }
92.      return texture;
93. }
```

从源代码中可以看出，SDL_CreateTexture()的大致流程如下。

1.
   **检查输入参数的合理性。** 例如像素格式是否支持，宽和高是否小于等于0等等。

2.
   **新建一个SDL_Texture。** 调用SDL_calloc()（实际上就是calloc()）为新建的SDL_Texture分配内存。

3.
   **调用SDL_Render的CreateTexture()方法创建纹理。** 这一步是整个函数的核心。

下面我们详细看一下几种不同的渲染器的CreateTexture()的方法。

1.
   **Direct3D**

Direct3D 渲染器中对应CreateTexture()的函数是D3D_CreateTexture()，它的源代码如下所示（位于render\direct3d\SDL_render_d3d.c）。

```cpp
[cpp]
1.   static int D3D_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
2.   {
3.       D3D_RenderData *renderdata = (D3D_RenderData *) renderer->driverdata;
4.       D3D_TextureData *data;
5.       D3DPOOL pool;
6.       DWORD usage;
7.       HRESULT result;
8.
9.
10.      data = (D3D_TextureData *) SDL_calloc(1, sizeof(*data));
11.      if (!data) {
12.          return SDL_OutOfMemory();
13.      }
14.      data->scaleMode = GetScaleQuality();
15.
16.
17.      texture->driverdata = data;
18.
19.
20.  #ifdef USE_DYNAMIC_TEXTURE
21.      if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
22.          pool = D3DPOOL_DEFAULT;
23.          usage = D3DUSAGE_DYNAMIC;
24.      } else
25.  #endif
26.      if (texture->access == SDL_TEXTUREACCESS_TARGET) {
27.          /* D3DPOOL_MANAGED does not work with D3DUSAGE_RENDERTARGET */
28.          pool = D3DPOOL_DEFAULT;
29.          usage = D3DUSAGE_RENDERTARGET;
30.      } else {
31.          pool = D3DPOOL_MANAGED;
32.          usage = 0;
33.      }
34.
35.
36.      result =
37.          IDirect3DDevice9_CreateTexture(renderdata->device, texture->w,
38.                                         texture->h, 1, usage,
39.                                         PixelFormatToD3DFMT(texture->format),
40.                                         pool, &data->texture, NULL);
41.      if (FAILED(result)) {
42.          return D3D_SetError("CreateTexture()", result);
43.      }
44.
45.
46.      if (texture->format == SDL_PIXELFORMAT_YV12 ||
47.          texture->format == SDL_PIXELFORMAT_IYUV) {
48.          data->yuv = SDL_TRUE;
49.
50.
51.          result =
52.              IDirect3DDevice9_CreateTexture(renderdata->device, texture->w / 2,
53.                                             texture->h / 2, 1, usage,
54.                                             PixelFormatToD3DFMT(texture->format),
55.                                             pool, &data->utexture, NULL);
56.          if (FAILED(result)) {
57.              return D3D_SetError("CreateTexture()", result);
58.          }
59.
60.
61.          result =
62.              IDirect3DDevice9_CreateTexture(renderdata->device, texture->w / 2,
63.                                             texture->h / 2, 1, usage,
64.                                             PixelFormatToD3DFMT(texture->format),
65.                                             pool, &data->vtexture, NULL);
66.          if (FAILED(result)) {
67.              return D3D_SetError("CreateTexture()", result);
68.          }
69.      }
70.
71.
72.      return 0;
73.  }
```

从代码中可以看出，该函数调用了Direct3D的API函数IDirect3DDevice9_CreateTexture()创建了一个纹理。

## 2.
### OpenGL

OpenGL渲染器中对应CreateTexture()的函数是GL_CreateTexture ()，它的源代码如下所示（位于render\opengl\SDL_render_gl.c）。

```cpp
[cpp]
1.   static int GL_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
2.   {
3.       GL_RenderData *renderdata = (GL_RenderData *) renderer->driverdata;
```

```c
     4.        GL_TextureData *data;
     5.        GLint internalFormat;
     6.        GLenum format, type;
     7.        int texture_w, texture_h;
     8.        GLenum scaleMode;
     9.
    10.
    11.        GL_ActivateRenderer(renderer);
    12.
    13.
    14.        if (!convert_format(renderdata, texture->format, &internalFormat,
    15.                            &format, &type)) {
    16.            return SDL_SetError("Texture format %s not supported by OpenGL",
    17.                                SDL_GetPixelFormatName(texture->format));
    18.        }
    19.
    20.
    21.        data = (GL_TextureData *) SDL_calloc(1, sizeof(*data));
    22.        if (!data) {
    23.            return SDL_OutOfMemory();
    24.        }
    25.
    26.
    27.        if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
    28.            size_t size;
    29.            data->pitch = texture->w * SDL_BYTESPERPIXEL(texture->format);
    30.            size = texture->h * data->pitch;
    31.            if (texture->format == SDL_PIXELFORMAT_YV12 ||
    32.                texture->format == SDL_PIXELFORMAT_IYUV) {
    33.                /* Need to add size for the U and V planes */
    34.                size += (2 * (texture->h * data->pitch) / 4);
    35.            }
    36.            data->pixels = SDL_calloc(1, size);
    37.            if (!data->pixels) {
    38.                SDL_free(data);
    39.                return SDL_OutOfMemory();
    40.            }
    41.        }
    42.
    43.
    44.        if (texture->access == SDL_TEXTUREACCESS_TARGET) {
    45.            data->fbo = GL_GetFBO(renderdata, texture->w, texture->h);
    46.        } else {
    47.            data->fbo = NULL;
    48.        }
    49.
    50.
    51.        GL_CheckError("", renderer);
    52.        renderdata->glGenTextures(1, &data->texture);
    53.        if (GL_CheckError("glGenTextures()", renderer) < 0) {
    54.            SDL_free(data);
    55.            return -1;
    56.        }
    57.        texture->driverdata = data;
    58.
    59.
    60.        if ((renderdata->GL_ARB_texture_rectangle_supported)
    61.            /* && texture->access != SDL_TEXTUREACCESS_TARGET */){
    62.            data->type = GL_TEXTURE_RECTANGLE_ARB;
    63.            texture_w = texture->w;
    64.            texture_h = texture->h;
    65.            data->texw = (GLfloat) texture_w;
    66.            data->texh = (GLfloat) texture_h;
    67.        } else {
    68.            data->type = GL_TEXTURE_2D;
    69.            texture_w = power_of_2(texture->w);
    70.            texture_h = power_of_2(texture->h);
    71.            data->texw = (GLfloat) (texture->w) / texture_w;
    72.            data->texh = (GLfloat) texture->h / texture_h;
    73.        }
    74.
    75.
    76.        data->format = format;
    77.        data->formattype = type;
    78.        scaleMode = GetScaleQuality();
    79.        renderdata->glEnable(data->type);
    80.        renderdata->glBindTexture(data->type, data->texture);
    81.        renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER, scaleMode);
    82.        renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER, scaleMode);
    83.        /* According to the spec, CLAMP_TO_EDGE is the default for TEXTURE_RECTANGLE
    84.           and setting it causes an INVALID_ENUM error in the latest NVidia drivers.
    85.        */
    86.        if (data->type != GL_TEXTURE_RECTANGLE_ARB) {
    87.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
    88.                                        GL_CLAMP_TO_EDGE);
    89.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
    90.                                        GL_CLAMP_TO_EDGE);
    91.        }
    92.    #ifdef __MACOSX__
    93.    #ifndef GL_TEXTURE_STORAGE_HINT_APPLE
    94.    #define GL_TEXTURE_STORAGE_HINT_APPLE        0x85BC
```

```
95.     #endif
96.     #ifndef STORAGE_CACHED_APPLE
97.     #define STORAGE_CACHED_APPLE              0x85BE
98.     #endif
99.     #ifndef STORAGE_SHARED_APPLE
100.    #define STORAGE_SHARED_APPLE             0x85BF
101.    #endif
102.        if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
103.            renderdata->glTexParameteri(data->type, GL_TEXTURE_STORAGE_HINT_APPLE,
104.                                    GL_STORAGE_SHARED_APPLE);
105.        } else {
106.            renderdata->glTexParameteri(data->type, GL_TEXTURE_STORAGE_HINT_APPLE,
107.                                    GL_STORAGE_CACHED_APPLE);
108.        }
109.        if (texture->access == SDL_TEXTUREACCESS_STREAMING
110.            && texture->format == SDL_PIXELFORMAT_ARGB8888
111.            && (texture->w % 8) == 0) {
112.            renderdata->glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_TRUE);
113.            renderdata->glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
114.            renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH,
115.                        (data->pitch / SDL_BYTESPERPIXEL(texture->format)));
116.            renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w,
117.                                texture_h, 0, format, type, data->pixels);
118.            renderdata->glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_FALSE);
119.        }
120.        else
121.    #endif
122.        {
123.            renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w,
124.                                texture_h, 0, format, type, NULL);
125.        }
126.        renderdata->glDisable(data->type);
127.        if (GL_CheckError("glTexImage2D()", renderer) < 0) {
128.            return -1;
129.        }
130.
131.
132.        if (texture->format == SDL_PIXELFORMAT_YV12 ||
133.            texture->format == SDL_PIXELFORMAT_IYUV) {
134.            data->yuv = SDL_TRUE;
135.
136.
137.            renderdata->glGenTextures(1, &data->utexture);
138.            renderdata->glGenTextures(1, &data->vtexture);
139.            renderdata->glEnable(data->type);
140.
141.
142.            renderdata->glBindTexture(data->type, data->utexture);
143.            renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER,
144.                                    scaleMode);
145.            renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER,
146.                                    scaleMode);
147.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
148.                                    GL_CLAMP_TO_EDGE);
149.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
150.                                    GL_CLAMP_TO_EDGE);
151.            renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w/2,
152.                                texture_h/2, 0, format, type, NULL);
153.
154.
155.            renderdata->glBindTexture(data->type, data->vtexture);
156.            renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER,
157.                                    scaleMode);
158.            renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER,
159.                                    scaleMode);
160.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
161.                                    GL_CLAMP_TO_EDGE);
162.            renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
163.                                    GL_CLAMP_TO_EDGE);
164.            renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w/2,
165.                                texture_h/2, 0, format, type, NULL);
166.
167.
168.            renderdata->glDisable(data->type);
169.        }
170.
171.
172.        return GL_CheckError("", renderer);
173.    }
```

从代码中可以看出，该函数调用了OpenGL的API函数glGenTextures()，glBindTexture()创建了一个纹理。并且使用glTexParameteri()设置了有关的一些参数。
在这里有一点需要注意，在OpenGL渲染器中，如果输入像素格式是YV12或者IYUV，就会使用3个纹理。

### 3.
#### Software

Software渲染器中对应CreateTexture()的函数是SW_CreateTexture ()，它的源代码如下所示（位于render\software\SDL_render_sw.c）。

```cpp
1.   static int SW_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
2.   {
3.       int bpp;
4.       Uint32 Rmask, Gmask, Bmask, Amask;
5.
6.
7.       if (!SDL_PixelFormatEnumToMasks
8.           (texture->format, &bpp, &Rmask, &Gmask, &Bmask, &Amask)) {
9.           return SDL_SetError("Unknown texture format");
10.      }
11.
12.
13.      texture->driverdata =
14.          SDL_CreateRGBSurface(0, texture->w, texture->h, bpp, Rmask, Gmask,
15.                               Bmask, Amask);
16.      SDL_SetSurfaceColorMod(texture->driverdata, texture->r, texture->g,
17.                             texture->b);
18.      SDL_SetSurfaceAlphaMod(texture->driverdata, texture->a);
19.      SDL_SetSurfaceBlendMode(texture->driverdata, texture->blendMode);
20.
21.
22.      if (texture->access == SDL_TEXTUREACCESS_STATIC) {
23.          SDL_SetSurfaceRLE(texture->driverdata, 1);
24.      }
25.
26.
27.      if (!texture->driverdata) {
28.          return -1;
29.      }
30.      return 0;
31.  }
```

该函数的源代码还没有详细分析。可以看出其中调用了SDL_CreateRGBSurface()创建了"Surface"。

文章标签： SDL  Direct3D  OpenGL  纹理  Texture

个人分类： SDL

所属专栏： 开源多媒体项目源代码分析