

学习ejoy2d——spritepack

gaccob

2014 年 3 月 12 日

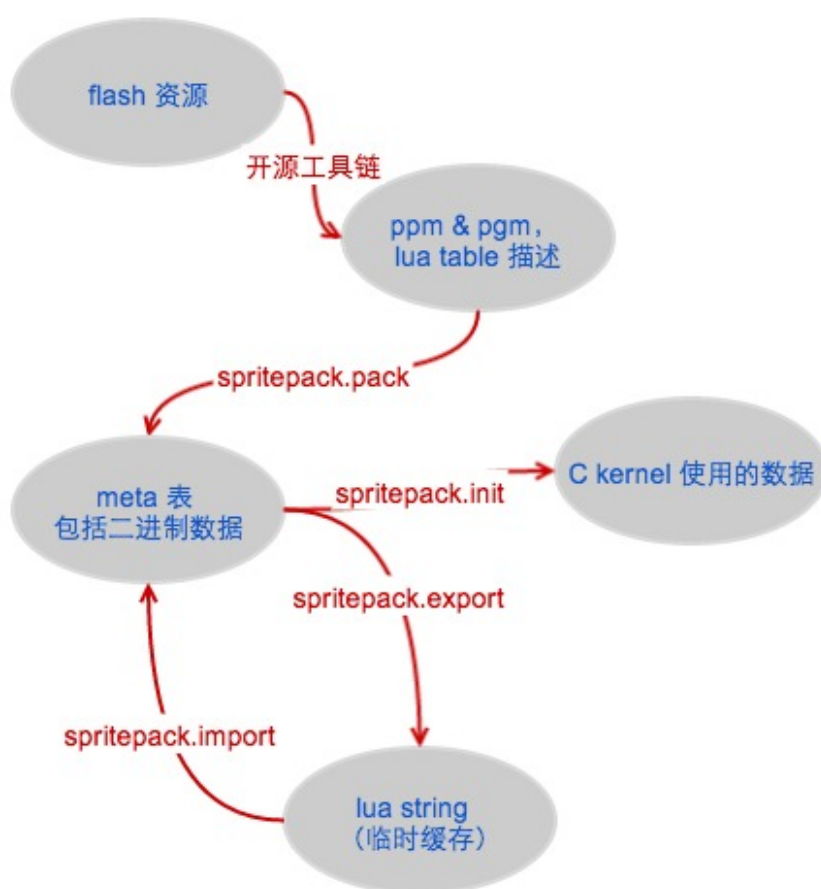
在学习sprite之前，先了解一下sprite_pack, lib/sprite_pack.h和lib/sprite_pack.c提供了底层的数据操作. ejoy2d/spritepack.lua封装了上层接口，提供数据之间的打包转换.

“这只是方便开发，在产品发行时不应该这样做”. 从版本的安全性来说，还需要做一些加密工作，防止资源被破解.

关于资源，根据github文档所述，有[开源工具链](#)支持从flash资源中导出.

sprite目前支持5种图元：

```
1 #define TYPE_EMPTY 0
2 #define TYPE_PICTURE 1
3 #define TYPE_ANIMATION 2
4 #define TYPE_POLYGON 3
5 #define TYPE_LABEL 4
6 #define TYPE_PANNEL 5
7 // anchor是一个比较特殊的类型，它没有对应的资源，仅提供了一个锚点，可以挂
  载其他sprite
8 #define TYPE_ANCHOR 6
9
10 // TYPE_PICTURE 四边形
11 struct pack_picture {
12     int n;
13     struct pack_quad rect[1];
14 };
15
16 // TYPE_ANIMATION 动画
17 struct pack_animation {
18     int frame_number;
19     int action_number;
20     int component_number;
21     struct pack_frame *frame;
22     struct pack_action *action;
```



```
23     struct pack_component component[1];
24 };
25
26 // TYPE_POLYGON 多边形
27 struct pack_poly {
28     int texid;
29     int n;
30     uint16_t *texture_coord;
31     int32_t *screen_coord;
32 };
33
34 // TYPE_LABEL 文字框
35 struct pack_label {
36     uint32_t color;
37     int width;
38     int height;
39     int align;
40     int size;
41     int edge;
42     int max_width;
43 };
44
45 // TYPE_PANNEL 面板
46 struct pack_pannel {
47     int width;
48     int height;
49     int scissor;
50 };
51
52 // component可以是任意sprite, 但是要自己保证不能成环
53 struct pack_component {
54     int id;
55     const char *name;
56 };
57
58 // C kernel 使用的sprite图元包
59 // data数组的下标是sprite的id, 也就是component引用的id.
60 struct sprite_pack {
61     int n;
62     uint8_t * type;
63     void ** data;
64     int tex[1];
65 };
```

spritepack的C代码中最重要的就是limport()函数, 它负责导入pack生成的meta数据到C使用的sprite图元包.

limport()封装成了lua接口, 在spritepack.init()中调用, 我觉得容易跟spritepack.import()的作用弄混...

```
1 // lua的输入参数格式:
2 // number: texture id | table: texture id table
3 // number: max id, pack对象的最大id
4 // number: max userdata size, 分配器的buffer size
5 // string: data | lightuserdata: data
6 //          | number: data size
7 static int
8 limport(lua_State *L) {
9     ...
10
11     // 一个简单的内存分配器, 保证了内存都from Lua (userdata)
12     struct import_alloc alloc;
13     alloc.L = L;
14     alloc.buffer = (char *)lua_newuserdata(L, size);
15     alloc.cap = size;
16
17     // 分配sprite_pack, 并初始化
18     struct sprite_pack *pack = (struct sprite_pack *)ialloc(&
19         alloc, sizeof(*pack) + (tex - 1) * sizeof(int));
20     pack->n = max_id + 1;
21
22     // 分配type, 4字节对齐(type是uint8)
23     int align_n = (pack->n + 3) & ~3;
24     pack->type = (uint8_t *)ialloc(&alloc, align_n * sizeof(
25         uint8_t));
26     memset(pack->type, 0, align_n * sizeof(uint8_t));
27
28     // 分配data指针
29     pack->data = (void **)ialloc(&alloc, pack->n * sizeof(
30         void*));
31     memset(pack->data, 0, pack->n * sizeof(void*));
32
33     // 导入texture id
34     if (lua_istable(L, 1)) {
35         int i;
36         for (i=0; i<tex; i++) {
37             lua_rawgeti(L, 1, i+1);
38             pack->tex[i] = (int)luaL_checkinteger(L, -1);
39             lua_pop(L, 1);
40         }
41     } else {
42         pack->tex[0] = (int)lua_tointeger(L, 1);
43     }
44
45     // 构造一个输入数据流(指向pack后的二进制数据), 方便后续操作
46     struct import_stream is;
47     is.alloc = &alloc;
48     is.pack = pack;
```

```

46     is.current_id = -1;
47     if (lua_isstring(L,4)) {
48         is.stream = lua_tolstring(L, 4, &is.size);
49     } else {
50         is.stream = (const char *)lua_touserdata(L, 4);
51         if (is.stream == NULL) {
52             return luaL_error(L, "Need const char *");
53         }
54         is.size = luaL_checkinteger(L, 5);
55     }
56
57     // 依次从数据流中导入数据
58     while (is.size != 0) {
59         import_sprite(&is);
60     }
61
62     return 1;
63 }

```

需要注意的是, sprite_pack数据(包括了它的所有图元), 内存都是从分配器(源自Lua)中分配, “C对象生命周期全部由Lua VM管理”。

spritepack.lua的接口pack()和init(), 将lua table描述打包成包含二进制数据的meta表, 或者在运行时导入meta为C kernel使用的sprite图元包。

```

1
2 — 输入的数据是lua table, 参照example/sample.lua
3 — 返回值是meta表
4 function spritepack.pack(data)
5     local ret = { texture = 0, maxid = 0, size = 0 , data =
6         {}, export = {} }
7     local ani_maxid = 0
8
9     for _,v in ipairs(data) do
10         if v.type ~= "particle" then
11
12             — 唯一id
13             local id = assert(tonumber(v.id))
14             if id > ret.maxid then
15                 ret.maxid = id
16             end
17
18             — export name 不是必须的, sample里只有两个animation有
19             local exportname = v.export
20             if exportname then
21                 assert(ret.export[exportname] == nil, "
                    Duplicate export name"..exportname)
22                 ret.export[exportname] = id

```

```

22         end
23
24         table.insert(ret.data, pack.word(id))
25
26         — 根据type分别pack
27         if v.type == "picture" then
28             local sz, texid = pack_picture(v, ret.data)
29             ret.size = ret.size + sz
30             if texid > ret.texture then
31                 ret.texture = texid
32             end
33         elseif v.type == "animation" then
34             local sz, maxid = pack_animation(v, ret.data)
35             ret.size = ret.size + sz
36             if maxid > ani_maxid then
37                 ani_maxid = maxid
38             end
39         elseif v.type == "polygon" then
40             local sz, texid = pack_polygon(v, ret.data)
41             ret.size = ret.size + sz
42             if texid > ret.texture then
43                 ret.texture = texid
44             end
45         elseif v.type == "label" then
46             local sz = pack_label(v, ret.data)
47             ret.size = ret.size + sz
48         elseif v.type == "pannel" then
49             local sz = pack_pannel(v, ret.data)
50             ret.size = ret.size + sz
51         else
52             error ("Unknown type " .. tostring(v.type))
53         end
54     end
55 end
56
57 if ani_maxid > ret.maxid then
58     error ("Invalid id in animation ".. ani_maxid)
59 end
60
61 ret.texture = ret.texture + 1
62
63 — 这里把table做了一次拼接，实际上就是打包成一块二进制数据
64 ret.data = table.concat(ret.data)
65
66 ret.size = ret.size + pack.pack_size(ret.maxid, ret.
67     texture)
68 return ret
end

```

```
69
70 function spritepack.init( name, texture, meta )
71     assert(pack_pool[name] == nil , string.format("sprite
       package [%s] is exist", name))
72     if type(texture) == "number" then
73         assert(meta.texture == 1)
74     else
75         assert(meta.texture == #texture)
76     end
77     -- 调用C接口来实现载入资源
78     pack_pool[name] = {
79         cobj = pack.import(texture, meta.maxid, meta.size, meta.
80             data, meta.data_sz),
81         export = meta.export,
82     }
83     meta.data = nil
84
85     return pack_pool[name]
86 end
```

spritepack.lua的接口export()和import(), 将pack后的meta数据做打包成字符串, 或者从字符串解包.

github上文档原文: "可以用 spritepack.export 在开发期预处理 spritepack.pack 生成的结果, spritepack.export 返回一个字符串, 可将这个字符串持久化到文件中".

```
1 function spritepack.export(meta)
2     local result = { true }
3     table.insert(result, pack.word(meta.maxid))
4     table.insert(result, pack.word(meta.texture))
5     table.insert(result, pack.int32(meta.size))
6     table.insert(result, pack.int32(#meta.data))
7     local s = 0
8     for k,v in pairs(meta.export) do
9         table.insert(result, pack.word(v))
10        table.insert(result, pack.string(k))
11        s = s + 1
12    end
13    result[1] = pack.word(s)
14    table.insert(result, meta.data)
15    return table.concat(result)
16 end
17
18 function spritepack.import(data)
19     local meta = { export = {} }
20     local export_n, off = pack.import_value(data, 1, 'w')
21     meta.maxid , off = pack.import_value(data, off, 'w')
```

```
22     meta.texture , off = pack.import_value(data, off, 'w')
23     meta.size , off = pack.import_value(data, off, 'i')
24     meta.data_sz , off = pack.import_value(data, off, 'i')
25     for i=1, export_n do
26         local id, name
27         id, off = pack.import_value(data, off, 'w')
28         name, off = pack.import_value(data, off, 's')
29         meta.export[name] = id
30     end
31     meta.data = pack.import_value(data, off, 'p')
32
33     return meta
34 end
```