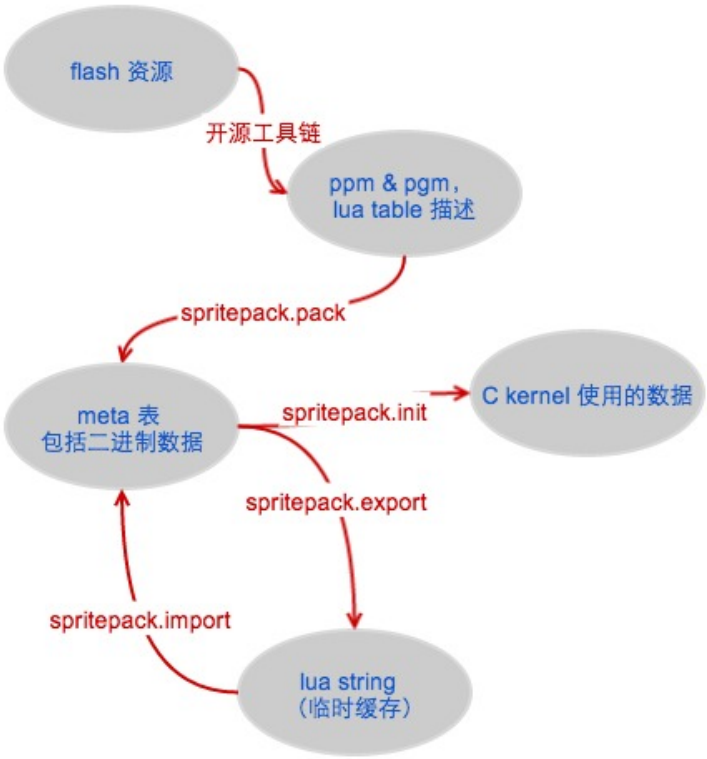
学习ejoy2d——spritepack

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/spritpack.lua封装了上层接口，提供数据之间的打解包转换.



"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:

```
#define TYPE_EMPTY 0
#define TYPE_PICTURE 1
#define TYPE_ANIMATION 2
#define TYPE_POLYGON 3
#define TYPE_LABEL 4
#define TYPE_PANNEL 5
// 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
#define TYPE_ANCHOR 6

// TYPE_PICTURE 四边形
struct pack_picture {
    int n;
    struct pack_quad rect[1];
};

// TYPE_ANIMATION 动画
struct pack_animation {
    int frame_number;
    int action_number;
    int component_number;
    struct pack_frame *frame;
    struct pack_action *action;
    struct pack_component component[1];
};

// TYPE_POLYGON 多边形
struct pack_poly {
    int texid;
    int n;
    uint16_t *texture_coord;
    int32_t *screen_coord;
};

// TYPE_LABEL 文字框
struct pack_label {
    uint32_t color;
    int width;
    int height;
    int align;
    int size;
    int edge;
    int max_width;
};

```

```
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 | lightuserdate: 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(&
           alloc, sizeof(*pack) + (tex - 1) * sizeof(int));
19     pack->n = max_id + 1;
20
```

```
21      // 分配type, 4字节对齐(type是uint8)
22      int align_n = (pack->n + 3) & ~3;
23      pack->type = (uint8_t *)ialloc(&alloc, align_n * sizeof(
            uint8_t));
24      memset(pack->type, 0, align_n * sizeof(uint8_t));
25
26      // 分配data指针
27      pack->data = (void **)ialloc(&alloc, pack->n * sizeof(
            void*));
28      memset(pack->data, 0, pack->n * sizeof(void*));
29
30      // 导入texure id
31      if (lua_istable(L,1)) {
32          int i;
33          for (i=0; i<tex; i++) {
34              lua_rawgeti(L,1,i+1);
35              pack->tex[i] = (int)luaL_checkinteger(L, -1);
36              lua_pop(L,1);
37          }
38      } else {
39          pack->tex[0] = (int)lua_tointeger(L,1);
40      }
41
42      // 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
43      struct import_stream is;
44      is.alloc = &alloc;
45      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data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 =
           {}, export = {} }
6      local ani_maxid = 0
7
8      for _,v in ipairs(data) do
9          if v.type ~= "particle" then
10
11             -- 唯一id
12             local id = assert(tonumber(v.id))
13             if id > ret.maxid then
14                 ret.maxid = id
15             end
16
17             -- export name 不是必须的, sample里只有两个animation有
18             local exportname = v.export
19             if exportname then
20                 assert(ret.export[exportname] == nil, "
                       Duplicate export name"..exportname)
21                 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
```

```
                    ret.texture = texid
                end
            elseif v.type == "label" then
                local sz = pack_label(v, ret.data)
                ret.size = ret.size + sz
            elseif v.type == "pannel" then
                local sz = pack_pannel(v, ret.data)
                ret.size = ret.size + sz
            else
                error ("Unknown type " .. tostring(v.type))
            end
        end
    end

    if ani_maxid > ret.maxid then
        error ("Invalid id in animation ".. ani_maxid)
    end

    ret.texture = ret.texture + 1

    -- 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
    ret.data = table.concat(ret.data)

    ret.size = ret.size + pack.pack_size(ret.maxid, ret.
        texture)
    return ret
end

function spritepack.init( name, texture, meta )
    assert(pack_pool[name] == nil , string.format("sprite
        package [%s] is exist", name))
    if type(texture) == "number" then
        assert(meta.texture == 1)
    else
        assert(meta.texture == #texture)
    end

    -- 调用C接口来实现载入资源
    pack_pool[name] = {
        cobj = pack.import(texture,meta.maxid,meta.size,meta.
            data, meta.data_sz),
        export = meta.export,
    }
    meta.data = nil

    return pack_pool[name]
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".

```lua
function spritepack.export(meta)
    local result = { true }
    table.insert(result, pack.word(meta.maxid))
    table.insert(result, pack.word(meta.texture))
    table.insert(result, pack.int32(meta.size))
    table.insert(result, pack.int32(#meta.data))
    local s = 0
    for k,v in pairs(meta.export) do
        table.insert(result, pack.word(v))
        table.insert(result, pack.string(k))
        s = s + 1
    end
    result[1] = pack.word(s)
    table.insert(result, meta.data)
    return table.concat(result)
end

function spritepack.import(data)
    local meta = { export = {} }
    local export_n, off = pack.import_value(data, 1, 'w')
    meta.maxid , off = pack.import_value(data, off, 'w')
    meta.texture , off = pack.import_value(data, off, 'w')
    meta.size , off = pack.import_value(data, off, 'i')
    meta.data_sz , off = pack.import_value(data, off, 'i')
    for i=1, export_n do
        local id, name
        id, off = pack.import_value(data, off, 'w')
        name, off = pack.import_value(data, off, 's')
        meta.export[name] = id
    end
    meta.data = pack.import_value(data, off, 'p')

    return meta
end
```