学习ejoy2d——sprite

gaccob

2014 年 3 月 12 日

1. sprite_pack

在学习sprite之前，先了解一下sprite_pack，它的主要作用是在运行时导入二进制的资源文件，导入的数据就存储在sprite_pack数据结构中.(根据文档所述，有开源工具支持从flash中导出二进制资源)．"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，资源还需要做加密，防止破解.

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，它没有资源数据，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;
};

// TYPE_PANNEL 面板
struct pack_pannel {
    int width;
    int height;
    int scissor;
};

// component可以是任意sprite, 但是要自己保证不能成环
struct pack_component {
    int id;
    const char *name;
};

// 这就是导入的lua描述文件的C数据结构
// data数组的下标是sprite的id, 也就是component引用的id.
struct sprite_pack {
    int n;
    uint8_t * type;
    void ** data;
    int tex[1];
};
```

spritepack中最重要的就是limport()函数, 它导入二进制资源到C数据结构中, 具体逻辑看源码:

```
// lua的输入参数格式:
// number: texture id | table: texture id table
// number: max id, pack对象的最大id
```

```
// number: max userdata size, 分配器的buffer size
// string: data | lightuserdate: data
// | number: data size
static int
limport(lua_State *L) {
    ...

    // 一个简单的分配器, max size来自lua的传入参数
    struct import_alloc alloc;
    alloc.L = L;
    alloc.buffer = (char *)lua_newuserdata(L, size);
    alloc.cap = size;

    // 从分配器中(userdata)分配sprite_pack, 并初始化
    struct sprite_pack *pack = (struct sprite_pack *)ialloc(&
        alloc, sizeof(*pack) + (tex - 1) * sizeof(int));
    pack->n = max_id + 1;

    // 分配type, 4字节对齐(type是uint8)
    int align_n = (pack->n + 3) & ~3;
    pack->type = (uint8_t *)ialloc(&alloc, align_n * sizeof(
        uint8_t));
    memset(pack->type, 0, align_n * sizeof(uint8_t));

    // 分配data指针
    pack->data = (void **)ialloc(&alloc, pack->n * sizeof(
        void*));
    memset(pack->data, 0, pack->n * sizeof(void*));

    // 导入texure id
    if (lua_istable(L,1)) {
        int i;
        for (i=0; i<tex; i++) {
            lua_rawgeti(L,1,i+1);
            pack->tex[i] = (int)luaL_checkinteger(L, -1);
            lua_pop(L,1);
        }
    } else {
        pack->tex[0] = (int)lua_tointeger(L,1);
    }

    // 构造一个输入数据流(指向目标资源数据), 方便后续导入
    struct import_stream is;
    is.alloc = &alloc;
    is.pack = pack;
    is.current_id = -1;
    if (lua_isstring(L,4)) {
        is.stream = lua_tolstring(L, 4, &is.size);
    } 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  }
```

在import_sprite()函数中, 依次import 2字节的sprite id, 1字节的type, 然后就根据不同的type分别import不同的sprite图元. 这些图元数据内存都是从import_stream中的分配器中分配(limport()中定义).

具体的过程可以参阅源码, 逻辑都比较简单, 唯一需要注意的是anchor类型的sprite id是一个特殊的默认值ANCHOR_ID(0xffff), 区别于其他常规sprite.

2. sprite

sprite是ejoy2d中最复杂的数据结构, "每个 sprite 都是若干图元以树状组合起来的".

```
1   struct sprite {
2       struct sprite * parent;
3       uint16_t type;
4
5       // 唯一id, 因为是数组, 所以不建议散的太开
6       uint16_t id;
7
8       struct sprite_trans t;
9
10      // 5种基本图元
11      union {
12          struct pack_animation *ani;
13          struct pack_picture *pic;
14          struct pack_polygon *poly;
15          struct pack_label *label;
16          struct pack_pannel *pannel;
17          struct matrix *mat;
18      } s;
19
```

```
20      // 渲染时的附加矩阵
21      struct matrix mat;
22
23      int start_frame;
24      int total_frame;
25      int frame;
26      bool visible;
27      bool message;
28      const char *name;   // name for parent
29      union {
30          struct sprite * children[1];
31          const char * text;
32          int scissor;
33      } data;
34  };
```