第 5 章

标签和菜单

游戏中文字与菜单很常用，字符串、标签和菜单经常结合在一起使用，因此本章主要介绍 Cocos2d-x Lua API 中的标签与菜单。

5.1 游戏中的文字

游戏场景中的文字包括静态文字和动态文字。静态文字如图 5-1 所示游戏场景中①号文字“COCOS2DX”，动态文字如图 5-1 所示游戏场景中的②号文字“Hello World”。

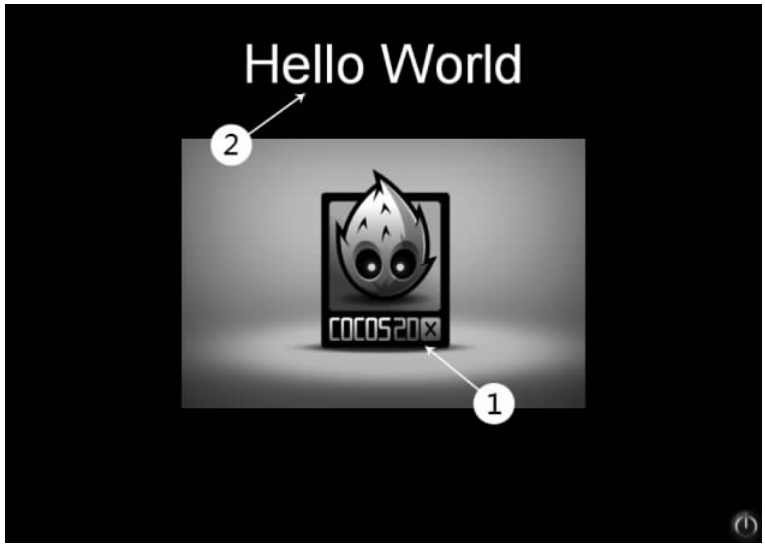

图 5-1 场景中的文字

静态文字一般是由美工使用 Photoshop 绘制在背景图片上，这种方式的优点是表现力很丰富，例如，①号文字“COCOS2DX”中的“COCOS”、“2D”和“X”设计的风格不同，而动态文字则不能，但是静态文字无法通过程序访问，无法动态修改内容。

动态文字一般是需要通过程序访问，需要动态修改内容。Cocos2d-x Lua API 可以通

过标签类实现。

5.2 使用标签

要想在游戏场景中显示动态文字,可以通过 Cocos2d-x Lua API 提供的标签类 Label 和 LabelAtlas 实现,也可通过 Cocos2d-x GUI 标签控件实现。本章介绍标签类 Label 和 LabelAtlas 实现方式。

5.2.1 Label 类

Label 类是 Cocos2d-x 3.x 后推出的新的标签类,这种标签通过使用 FreeType[①] 来使它在不同的平台上有相同的视觉效果。由于使用更快的缓存代理,它的渲染也将更加快速。Label 提供了描边和阴影等特性。Label 类可以创建系统字体标签、TTF 字体标签和位图字体标签。

提示 系统字体标签就是当前平台系统中安装的字体,我们要确保所用字体在系统中是存在的。TTF 字体[②]标签是加载 TTF 字体文件显示文字的标签。位图字体标签加载两个文件:一个纹理图集(.png)和一个字体坐标文件(.fnt)。

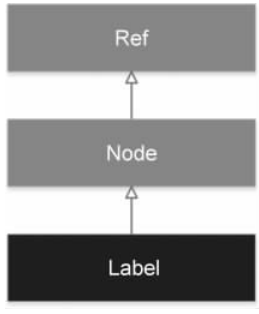

图 5-2 Label 类图

Label 类的类图如图 5-2 所示。

创建 Label 类静态 create 函数常用的有如下几个:

```
Label:createWithSystemFont(text,                          -- 是要显示的文字
      font,                                               -- 系统字体名
      fontSize,                                           -- 字体的大小
      dimensions = size(0,0),                             -- 可省略,参考 LabelTTF 定义
      vAlignment = TEXT_ALIGNMENT_LEFT,                   -- 可省略,参考 LabelTTF 定义
      vAlignment = VERTICAL_TEXT_ALIGNMENT_TOP            -- 可省略,参考 LabelTTF 定义
)

Label:createWithTTF(const std::string & text,
      fontFile,                                           -- 字体文件
      fontSize,
      dimensions = size(0,0),
      hAlignment = TEXT_ALIGNMENT_LEFT,
      vAlignment = VERTICAL_TEXT_ALIGNMENT_TOP
    )
```

① FreeType 库是一个完全免费(开源)的、高质量的且可移植的字体引擎,它提供统一的接口来访问多种字体格式文件。——引自百度百科 http://baike.baidu.com/view/4579855.htm

② TTF(TrueTypeFont)是 Apple 公司和 Microsoft 公司共同推出的字体文件格式,随着 Windows 的流行,已经变成最常用的一种字体文件表示方式。——引自百度百科 http://baike.baidu.com/view/1003637.htm

```
Label:createWithTTF(ttfConfig,                                    -- 字体配置信息
      text,
      hAlignment = TEXT_ALIGNMENT_LEFT,
      int maxLineWidth = 0                                        -- 可省略,标签的最大宽度
    )

Label:createWithBMFont(const std::string& bmfontFilePath,  -- 位图字体文件
      text,
      hAlignment = TEXT_ALIGNMENT_LEFT,
      int maxLineWidth = 0,
      imageOffset = p(0,0)                                        -- 可省略,在位图中的偏移量
    )
```

其中 createWithSystemFont 是创建系统字体标签对象,createWithTTF 是创建 TTF 字体标签对象,createWithBMFont 是创建位图字体标签对象。

5.2.2 实例:使用系统字体和 TTF 字体

下面用实例说明通过 Label 类使用系统字体和 TTF 字体,如图 5-3 所示。

图 5-3 使用系统字体和 TTF 字体

GameScene.lua 的 GameScene:createLayer()函数如下:

```
-- 创建层
function GameScene:createLayer()
    cclog("GameScene init")
    local layer = cc.Layer:create()

    -- 上下两个控件的距离
    local gap = 120

    local label1 = cc.Label:createWithSystemFont("世界你好 1", "Arial", 36)          ①
    label1:setPosition(cc.p(size.width/2, size.height - gap))
    layer:addChild(label1, 1)

    local label2 = cc.Label:createWithTTF("世界你好 2", "fonts/STLITI.ttf", 36)      ②
```

```
    label2:setPosition(cc.p(size.width/2, size.height - 2 * gap))
    layer:addChild(label2, 1)

    local ttfConfig = {}                                                    ③
    ttfConfig.fontFilePath = "fonts/Marker Felt.ttf"
    ttfConfig.fontSize = 32

    local label3 = cc.Label:createWithTTF(ttfConfig, "Hello World1")        ④
    label3:setPosition(cc.p(size.width/2, size.height - 3 * gap))
    layer:addChild(label3 , 1)

    ttfConfig.outlineSize = 4                                               ⑤
    local label4 = cc.Label:createWithTTF(ttfConfig, "Hello World2")        ⑥
    label4:setPosition(cc.p(size.width/2, size.height - 4 * gap))
    label5:enableShadow(cc.c4b(255,255,255,128), cc.size(4, -4))            ⑦
    label5:setColor(cc.c3b(255, 0, 0))                                      ⑧
    layer:addChild(label4, 1)

    return layer
end
```

上述第①行代码是通过 createWithSystemFont 函数创建系统字体标签对象。

第②行代码是通过 createWithTTF 创建 TTF 字体标签对象。

第③行代码 local ttfConfig = {}是声明一个 ttfConfig 变量，ttfConfig 的属性如下：

```
fontFilePath                                -- 字体文件路径
fontSize,                                   -- 字体大小
glyphs = GLYPHCOLLECTION_DYNAMIC,           -- 字体库类型
customGlyphs                                -- 自定义字体库
outlineSize                                 -- 字体描边
distanceFieldEnabled                        -- 开启距离字段字体开关
```

第④行代码 cc.Label:createWithTTF(ttfConfig, "Hello World1")是通过指定 ttfConfig 创建 TTF 字体标签。

第⑤行代码 ttfConfig.outlineSize = 4 设置 ttfConfig 的描边字段。

第⑥行代码 cc.Label:createWithTTF(ttfConfig, "Hello World2")是重新创建 TTF 字体标签。

第⑦行代码 label5:enableShadow(cc.c4b(255, 255,255,128), cc.size(4, -4))是设置标签的阴影效果。

第⑧行代码 label5:setColor(cc.c3b(255, 0, 0))是设置标签的颜色。

图 5-4 图片集文件

5.2.3 实例：使用位图字体

位图字体标签加载两个文件：一个图片集(.png)和一个字体坐标文件(.fnt)。图 5-4 展示了图片集文件

BMFont.png 的内容，对应还有一个字体坐标文件 BMFont.fnt。

字体坐标文件 BMFont.fnt 代码如下：

```
info face = "AmericanTypewriter" size = 64 bold = 0 italic = 0 charset = "" unicode = 0 stretchH =
100 smooth = 1 aa = 1 padding = 0,0,0,0 spacing = 2,2
common lineHeight = 73 base = 58 scaleW = 512 scaleH = 512 pages = 1 packed = 0
page id = 0 file = "BMFont.png"
chars count = 95
char id = 124 x = 2 y = 2 width = 9 height = 68 xoffset = 14 yoffset = 9 xadvance = 32 page = 0 chnl =
0 letter = "|"
char id = 41 x = 13 y = 2 width = 28 height = 63 xoffset = 1 yoffset = 11 xadvance = 29 page = 0 chnl
 = 0 letter = ")"
char id = 40 x = 43 y = 2 width = 28 height = 63 xoffset = 4 yoffset = 11 xadvance = 29 page = 0 chnl
 = 0 letter = "("
... ...
char id = 32 x = 200 y = 366 width = 0 height = 0 xoffset = 16 yoffset = 78 xadvance = 16 page = 0
chnl = 0
letter = "space"
```

使用 LabelBMFont 需要注意的是图片集文件和坐标文件需要放置在相同的资源目录下，文件命名相同。图片集合和坐标文件是可以通过位图字体工具制作而成的，位图字体工具的使用请参考本系列丛书的工具卷(《Cocos2d-x 实战：工具卷》)。

下面用实例说明通过 Label 类使用位图字体，如图 5-5 所示。

图 5-5　使用位图字体

下面看一下 GameScene.lua 的 GameScene:createLayer()函数如下：

```
-- 创建层
function GameScene:createLayer()
    cclog("GameScene init")
    local layer = cc.Layer:create()
```

```
    -- 上下两个控件的距离
    local gap = 90

    local label1 = cc.Label:createWithBMFont("fonts/bitmapFontChinese.fnt", "中国")      ①
    label1:setPosition(cc.p(size.width/2, size.height - 3 * gap))
    layer:addChild(label1, 1)

    local label2 = cc.Label:createWithBMFont("fonts/BMFont.fnt", "Hello World")          ②
    label2:setPosition(cc.p(size.width/2, size.height - 4 * gap))
    layer:addChild(label2, 1)

    return layer
end

return GameScene
```

第①和第②行代码通过 Label 类的 createWithBMFont 函数创建位图字体标签对象。

注意 标签中显示的文字字符一定包含在位图字体图片集中,例如代码第①行要显示中文,那么位图字体图片集 bitmapFontChinese.png 文件中应当包含这些中文字。

5.2.4 LabelAtlas 类

LabelAtlas 是图片集标签,其中的 Atlas 本意是“地图集”、“图片集”,这种标签显示的文字是从一个图片集中取出的,因此使用 LabelAtlas 需要额外加载图片集文件。LabelAtlas 比 LabelTTF 快很多。LabelAtlas 中的每个字符必须有固定的高度和宽度。

下面用实例说明通过 LabelAtlas 类使用图片集字体,如图 5-6 所示。

图 5-6 LabelAtlas 实现的图片集文字

LabelAtlas 实现的图片集文字主要代码如下：

```
function GameScene:createLayer()

    local layer = cc.Layer:create()

    local label = cc.LabelAtlas:create("HelloWorld",
                  "fonts/tuffy_bold_italic-charmap.png", 48, 66, string.byte(" "))      ①
    label:setPosition(cc.p(size.width/2 - label:getContentSize().width / 2,
                           size.height - label:getContentSize().height))
    layer:addChild(label, 1)

    local sprite = cc.Sprite:create("HelloWorld.png")
    sprite:setPosition(cc.p(size.width/2, size.height/2))
    layer:addChild(sprite, 0)

    return layer
end
```

上述第①行代码是创建一个 LabelAtlas 对象，create 函数的第一个参数是要显示的文字，第二个参数是图片集文件（见图 5-7），第三个参数是字符高度，第四个参数是字符宽度，第五个参数是开始字符。

使用 LabelAtlas 需要注意的是图片集文件需要放置在 res 目录下。

图 5-7　图片集文件

5.3　使用菜单

菜单中又包含了菜单项，菜单项类是 MenuItem，每个菜单项都有三个基本状态：正常、选种和禁止。回顾一下 MenuItem 类图如图 5-8 所示。

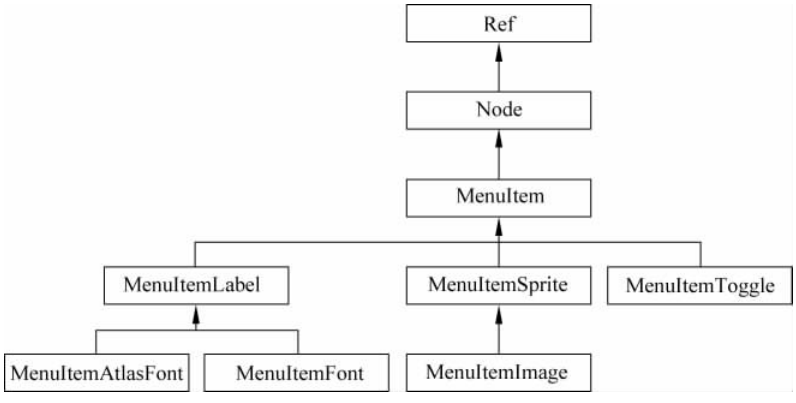

图 5-8　MenuItem 类图

菜单是按照菜单项进行分类的，从 MenuItem 类图中可见 MenuItem 的子类有 MenuItemLabel、MenuItemSprite 和 MenuItemToggle。其中 MenuItemLabel 类是文本菜单，它有 2 个子类：MenuItemAtlasFont 和 MenuItemFont。MenuItemSprite 类是精灵菜单，它的子类是 MenuItemImage，属于图片菜单。MenuItemToggle 类是开关菜单。

下面重点介绍文本菜单、精灵菜单、图片菜单和开关菜单。

5.3.1 文本菜单

文本菜单的菜单项只能显示文本，包括 MenuItemLabel、MenuItemFont 和 MenuItemAtlasFont。MenuItemLabel 是个抽象类，具体使用时只使用 MenuItemFont 和 MenuItemAtlasFont 两个类。

提示 目前的 Cocos2d-x 3.11 版本 Lua 中 MenuItemAtlasFont 还没有可移植性，因此还不能使用。可以使用 LabelAtlas 和 MenuItemLabel 结合来实现 MenuItemAtlasFont 效果。

本节通过一个实例来介绍文本菜单的使用，如图 5-9 所示，其中菜单 Start 是文本字体菜单项，菜单 Help 是 Atlas 字体菜单项。

图 5-9 文本菜单实例

GameScene.lua 的 GameScene:createLayer()函数如下：

```
function GameScene:createLayer()

    local layer = cc.Layer:create()

    local sprite = cc.Sprite:create("menu/background.png")
```

```
    sprite:setPosition(cc.p(size.width/2, size.height/2))
    layer:addChild(sprite)

    cc.MenuItemFont:setFontName("Times New Roman")                              ①
    cc.MenuItemFont:setFontSize(86)                                             ②

    local item1 = cc.MenuItemFont:create("Start")                               ③
    local function menuItem1Callback(sender)                                    ④
        cclog("Touch Start Menu Item.")
    end
    item1:registerScriptTapHandler(menuItem1Callback)                           ⑤

    local labelAtlas = cc.LabelAtlas:create("Help",
        "menu/tuffy_bold_italic-charmap.png", 48, 65, string.byte(' '))         ⑥
    local item2 = cc.MenuItemLabel:create(labelAtlas)                           ⑦
    local function menuItem2Callback(sender)
        cclog("Touch Help Menu Item.")
    end
    item2:registerScriptTapHandler(menuItem2Callback)

    local mn = cc.Menu:create(item1, item2)                                     ⑧
    mn:alignItemsVertically()                                                   ⑨
    layer:addChild(mn)                                                          ⑩

    return layer
end
```

上述第①和第②行代码是设置文本菜单的文本字体和字体大小。

第③行代码是创建 MenuItemFont 菜单项对象，它是一个一般文本菜单项，create 函数的参数是菜单项的文本内容。

第④行代码是定义 MenuItemFont 菜单项单击事件的回调函数。

第⑤行代码是注册 MenuItemFont 菜单项单击事件，其参数就是前面定义的回调函数。

第⑥行代码是定义一个 Atlas 标签。

第⑦行代码是通过 Atlas 标签创建 MenuItemLabel 菜单项对象。

第⑧行代码 cc.Menu:create(item1, item2)是创建菜单对象，把之前创建的菜单项添加到菜单中，create 函数是这些菜单项的数组。

第⑨行代码 mn:alignItemsVertically()是设置菜单项垂直对齐。

第⑩行代码 layer:addChild(mn)是把菜单对象添加到当前层中。

5.3.2　精灵菜单和图片菜单

精灵菜单的菜单项类是 MenuItemSprite，图片菜单的菜单项类是 MenuItemImage。由于 MenuItemImage 继承于 MenuItemSprite，所以图片菜单也属于精灵菜单。为什么叫精灵菜单呢？那是因为这些菜单项具有精灵的特点，可以让精灵动起来，具体使用时是把一个精灵放置到菜单中作为菜单项。

精灵菜单项类 MenuItemSprite 的一个创建函数 create 定义如下：

```
cc.MenuItemSprite:create ( normalSprite,                    -- 菜单项正常显示时的精灵
  selectedSprite,                                           -- 选择菜单项时的精灵
  disabledSprite                                            -- 菜单项禁用时的精灵
)
```

使用 MenuItemSprite 比较麻烦,在创建 MenuItemSprite 之前要先创建 3 种不同状态的精灵(即 normalSprite、selectedSprite 和 disabledSprite)。MenuItemSprite 还有一些 create 函数,在这些函数中可以省略 disabledSprite 参数。

如果精灵是由图片构成的,可以使用 MenuItemImage 实现与精灵菜单同样的效果。MenuItemImage 类的一个创建函数 create 定义如下:

```
cc.MenuItemImage:create (normalImage,                       -- 菜单项正常显示时的图片
  selectedImage,                                            -- 选择菜单项时的图片
  disabledImage                                             -- 菜单项禁用时的图片
)
```

MenuItemImage 还有一些 create 函数,在这些函数中可以省略 disabledImage 参数。

本节通过一个实例介绍精灵菜单和图片菜单的使用,如图 5-10 所示。

图 5-10 精灵菜单和图片菜单实例

GameScene.lua 的 GameScene:createLayer()函数如下:

```
function GameScene:createLayer()

    local layer = cc.Layer:create()
    local director = cc.Director:getInstance()
    local sprite = cc.Sprite:create("menu/background.png")
    sprite:setPosition(cc.p(size.width/2, size.height/2))
    layer:addChild(sprite)

    -- 开始精灵
    local startlocalNormal = cc.Sprite:create("menu/start-up.png")                          ①
    local startlocalSelected = cc.Sprite:create("menu/start-down.png")                      ②
    local startMenuItem = cc.MenuItemSprite:create(startlocalNormal, startlocalSelected)    ③
    startMenuItem:setPosition(director:convertToGL(cc.p(700, 170)))                         ④
    local function menuItemStartCallback(sender)
        cclog("Touch Start.")
    end
```

```
    startMenuItem:registerScriptTapHandler(menuItemStartCallback)

    -- 设置图片菜单
    local settingMenuItem = cc.MenuItemImage:create(
        "menu/setting - up.png",
        "menu/setting - down.png")                                          ⑤
    settingMenuItem:setPosition(director:convertToGL(cc.p(480, 400)))       ⑥
    local function menuItemSettingCallback(sender)
        cclog("Touch Setting.")
    end
    settingMenuItem:registerScriptTapHandler(menuItemSettingCallback)

    -- 帮助图片菜单
    local helpMenuItem = cc.MenuItemImage:create(
        "menu/help - up.png",
        "menu/help - down.png")                                             ⑦
    helpMenuItem:setPosition(director:convertToGL(cc.p(860, 480)))          ⑧
    local function menuItemHelpCallback(sender)
        cclog("Touch Help.")
    end
    helpMenuItem:registerScriptTapHandler(menuItemHelpCallback)

    local mn = cc.Menu:create(startMenuItem, settingMenuItem, helpMenuItem) ⑨
    mn:setPosition(cc.p(0, 0))                                              ⑩
    layer:addChild(mn)

    return layer
end
```

上述第①和第②行代码是创建两种不同状态的精灵。

第③行代码是创建精灵菜单项 MenuItemSprite 对象。

第④行代码是设置开始菜单项(startMenuItem)位置,注意这个坐标是(700, 170),由于(700, 170)的坐标是 UI 坐标,需要转换为 OpenGL 坐标。

第⑤和⑦行代码是创建图片菜单项 MenuItemImage 对象。

第⑥和⑧行代码是设置图片菜单项位置。

第⑨行代码是 Menu 对象。

第⑩行代码是菜单的位置 mn:setPosition(cc.p(0, 0))。

由于背景图片大小是 1136×640,而 Cocos Code IDE 工具模板生成的窗口大小是 960×640,所以需要重新设置大小,修改 main.lua 代码如下:

```
local function main()
    collectgarbage("collect")
    -- avoid memory leak
    collectgarbage("setpause", 100)
    collectgarbage("setstepmul", 5000)

    cc.FileUtils:getInstance():addSearchPath("src")
    cc.FileUtils:getInstance():addSearchPath("res")
    cc.Director:getInstance():getOpenGLView():setDesignResolutionSize(1136, 640, 0)    ①
```

```
    -- create scene
    local scene = require("GameScene")
    local gameScene = scene.create()

    if cc.Director:getInstance():getRunningScene() then
        cc.Director:getInstance():replaceScene(gameScene)
    else
        cc.Director:getInstance():runWithScene(gameScene)
    end

end
```

需要在第①行修改 setDesignResolutionSize(1136, 640, 0)代码。

5.3.3 开关菜单

开关菜单的菜单项类是 MenuItemToggle,它是一种可以进行 2 种状态切换的菜单项。本节通过一个实例介绍其他复杂类型的开关菜单的使用。如图 5-11 所示是一个游戏音效和背景音乐设置界面,可以通过开关菜单实现这个功能,美术设计师为每一个设置项目(音效和背景音乐)分别准备了两张图片。

图 5-11 开关菜单实例

GameScene.lua 中 MenuItemImage 菜单项的代码如下:

```
function GameScene:createLayer()

    local layer = cc.Layer:create()

    local director = cc.Director:getInstance()

    local sprite = cc.Sprite:create("menu/setting-back.png")
    sprite:setPosition(cc.p(size.width/2, size.height/2))
    layer:addChild(sprite)

    -- 音效
    local soundOnMenuItem = cc.MenuItemImage:create("menu/on.png", "menu/on.png")        ①
    local soundOffMenuItem = cc.MenuItemImage:create("menu/off.png", "menu/off.png")     ②
    local soundToggleMenuItem = cc.MenuItemToggle:create(soundOnMenuItem,
```

```
                                                soundOffMenuItem)                    ③
    soundToggleMenuItem:setPosition(director:convertToGL(cc.p(818, 220)))
    local function menuSoundToggleCallback(sender)
        cclog("Sound Toggle.")
    end
    soundToggleMenuItem:registerScriptTapHandler(menuSoundToggleCallback)

    -- 音乐
    local musicOnMenuItem = cc.MenuItemImage:create("menu/on.png", "menu/on.png")      ④
    local musicOffMenuItem = cc.MenuItemImage:create("menu/off.png", "menu/off.png")   ⑤
    local musicToggleMenuItem = cc.MenuItemToggle:create(musicOnMenuItem,
                                                         musicOffMenuItem)             ⑥
    musicToggleMenuItem:setPosition(director:convertToGL(cc.p(818, 362)))
    local function menuMusicToggleCallback(sender)
        cclog("Music Toggle.")
    end
    musicToggleMenuItem:registerScriptTapHandler(menuMusicToggleCallback)

    -- Ok 按钮
    local okMenuItem = cc.MenuItemImage:create(
        "menu/ok - down.png",
        "menu/ok - up.png")
    okMenuItem:setPosition(director:convertToGL(cc.p(600, 510)))

    local mn = cc.Menu:create(soundToggleMenuItem, musicToggleMenuItem,okMenuItem)     ⑦
    mn:setPosition(cc.p(0, 0))
    layer:addChild(mn)

    return layer
end
```

上面第①行代码是创建音效开的图片菜单项。

第②行代码是创建音效关的图片菜单项。

第③行代码是创建开关菜单项 MenuItemToggle。

第④～⑥行代码创建了背景音乐开关菜单项。

第⑦行代码是通过上面创建的开关菜单项创建 Menu 对象。

本章小结

通过对本章的学习，使读者了解了 Cocos2d-x Lua API 文字和菜单的相关知识，本章介绍了标签类 Label 和 LabelAtlas，另外，在菜单部分还介绍了文本菜单、精灵菜单、图片菜单和开关菜单等。