第 3 章 Hello Cocos2d-x Lua

在开始详细介绍 Cocos2d-x Lua 引擎的 API 之前，有必要先了解一下手机游戏引擎有哪些和了解 Cocos2d-x Lua 的前世今生。然后从一个 HelloLua 入手，介绍 Cocos2d-x Lua 的基本开发流程、Cocos2d-x Lua 生命周期以及 Cocos2d-x Lua 核心知识体系。

3.1 移动平台游戏引擎介绍

游戏引擎是指一些已编写好的游戏程序模块。游戏引擎包含渲染引擎（即“渲染器”，包括二维图像引擎和三维图像引擎）、物理引擎、碰撞检测系统、音效、脚本引擎、电脑动画、人工智能、网络引擎以及场景管理等子系统。

在目前移动平台游戏引擎中主要可以分为 2D 和 3D 引擎。2D 引擎主要有 Cocos2d-iphone、Cocos2d-x、Cocos2d-JS、Corona SDK、Construct 2、WiEngine 和 Cyclone 2D，3D 引擎主要有 Unity 3D、Unreal Development Kit、ShiVa 3D 和 Marmalade。此外还有一些针对于 HTML 5 的游戏引擎，如 Cocos2d-html5、X-Canvas 和 Sphinx 等。

这些游戏引擎各有千秋，但是目前得到市场普遍认可的 2D 引擎是 Cocos2d-iphone、Cocos2d-x 和 Cocos2d-JS，3D 引擎是 Unity 3D。

3.2 Cocos2d 游戏引擎

Cocos2d-iphone、Cocos2d-x 和 Cocos2d-JS 是目前最流行的 2D 游戏引擎。它们属于同一家族，具有相同的 API。

3.2.1 Cocos2d 游戏引擎家谱

在介绍 Cocos2d-x Lua 之前有必要先介绍一下 Cocos2d 的家谱，如图 3-1 所示是 Cocos2d 的家谱。

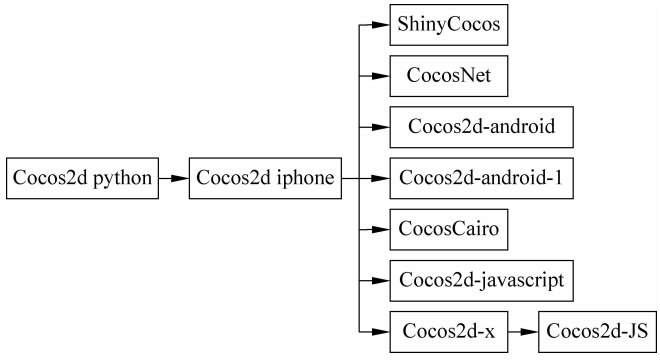


图 3-1 Cocos2d 的家谱

Cocos2d 最早是由阿根廷的 Ricardo 和他的朋友使用 Python 开发的，后移植到 iPhone 平台，使用的语言是 Objective-C。随着在 iPhone 平台取得了成功，Cocos2d 引擎变得更加多元化，将对其中各个引擎介绍如下。

- ShinyCocos：使用 Ruby 对 Cocos2d-iphone 进行封装，使用 Ruby api 开发。
- CocosNet：是在 MonoTouch 平台上使用的 Cocos2d 引擎，采用.NET 实现。
- Cocos2d-android：是为 Android 平台使用的 Cocos2d 引擎，采用 Java 实现。
- Cocos2d-android-1：是为 Android 平台使用的 Cocos2d 引擎，采用 Java 实现，由国内人员开发的。
- Cocos2d-javascript：是采用 Javascript 脚本语言实现的 Cocos2d 引擎。
- Cocos2d-x：是采用 C++实现的 Cocos2d 引擎，它是由 Cocos2d-x 团队开发的分支项目。
- Cocos2d-JS：是采用 JavaScriptAPI 的 Cocos2d 引擎，一方面它可以绑定在 Cocos2d-x 上开发基于本地技术的游戏；另一方面依托浏览器运行，开发基于 Web 的网页游戏。它也是由 Cocos2d-x 团队开发的分支项目。

此外，历史上 Cocos2d 还出现过很多分支，随着技术的发展这些逐渐消亡了，其中最有生命力的当属 Cocos2d-x 和 Cocos2d-JS 引擎了。

3.2.2 Cocos2d-x 引擎

Cocos2d-x 设计目标如图 3-2 所示。横向能够支持各种操作系统，桌面系统包括 Windows、Linux 和 Mac OS X，移动平台包括 iOS、Android、Windows Phone、Bada、BlackBerry 和 MeeGo 等。纵向方面向下能够支持 OpenGL ES1.1、OpenGL ES1.5 和 OpenGL ES2.0，以及 DirectX11 等技术，向上支持 JavaScript 和 Lua 脚本绑定。

简单说 Cocos2d-x 设计目标是为了实现跨平台，而不再为同一款游戏在不同平台发布

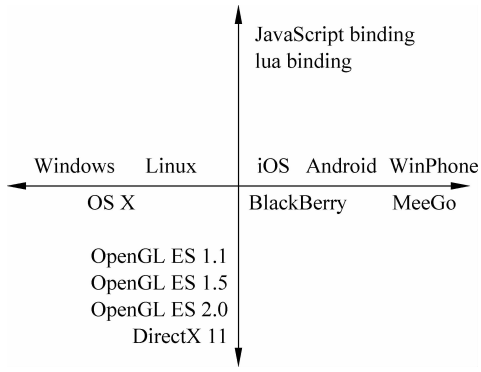


图 3-2　Cocos2d-x 设计目标

而进行编译了。而且 Cocos2d-x 为程序员考虑的更多，很多程序员可能对于 C++不熟悉，针对这种情况可以使用 JavaScript 和 Lua[①] 开发游戏。

3.2.3　JavaScript 和 Lua 绑定

Cocos2d-x 设计得非常巧妙，可以绑定 JavaScript 和 Lua 脚本语言，使得不熟悉 C++的人员也能使用 Cocos2d-x 引擎开发游戏，Cocos2d-x 不使用 C++语言的 API，而是使用 JavaScript 和 Lua 语言 API。Cocos2d-x 绑定 JavaScript 和 Lua 脚本原理如图 3-3 所示。

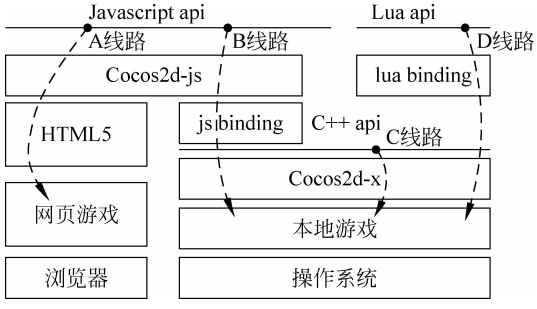


图 3-3　Cocos2d-x 绑定 JavaScript 和 Lua 脚本

从图 3-3 可知，通过 Cocos2d-x 和 Cocos2d-JS 引擎，程序员可以开发网页游戏和本地游戏。图中 A 线路和 B 线路都是给掌握 JavaScript 脚本的程序员准备的，通过 A 线路，使用 Cocos2d-JS 引擎开发基于 HTML5 的网页游戏。同样的 JavaScript 代码，也可以通过 B 线路，使用 js binding(js 绑定)技术透过 C++API 访问 Cocos2d-x 引擎，开发本地游戏。使得

① Lua 是一个小巧的脚本语言，是巴西里约热内卢天主教大学(Pontifical Catholic University of Rio de Janeiro)里的一个研究小组(Roberto Ierusalimschy、Waldemar Celes 和 Luiz Henrique de Figueiredo)于 1993 年开发。——引自于百度百科 http://baike.baidu.com/view/416116.htm?fr=wordsearch

同样的 JavaScript 代码就可以实现在不同平台下运行。

图 3-3 中的 C 线路是给熟悉 C++的程序员准备的。通过 C++API 访问 Cocos2d-x 引擎,开发本地游戏。

图 3-3 中的 D 线路是本书重点介绍的内容,是为熟悉 Lua 脚本的程序员准备的。使用 Lua binding(Lua 绑定)技术透过 C++API 访问 Cocos2d-x 引擎,开发本地游戏,为统一概念,本书中 Cocos2d-x Lua 绑定技术称为 Cocos2d-x Lua。

通过 Cocos2d-x 和 Cocos2d-JS 引擎 Cocos2d-x 团队构建了自己的技术生态圈,通过这些引擎使得游戏开发越来越简单。

3.3 搭建 Cocos2d-x Lua 开发环境

使用 Cocos2d-x Lua 开发游戏,主要的程序代码是 Lua 语言,因此,凡是能够开发 Lua 语言工具都适用于 Cocos2d-x Lua 游戏开发。本书推荐 Cocos Code IDE 工具。

3.3.1 搭建 Cocos Code IDE 开发环境

Cocos Code IDE 是 Cocos2d-x 团队开发的,用于开发 Cocos2d-JS 和 Cocos2d-x Lua 绑定的游戏工具,它是基于 Eclipse[①] 平台的开发工具,Eclipse 基于 Java 的要想运行 Cocos Code IDE 工具,需要安装 JDK 或 JRE,JDK 是 Java 开发工具包,JRE 是 Java 运行环境。

1. JDK

JDK 的安装和设置见 2.1.1 节。

2. Cocos Code IDE 的下载和安装

Cocos Code IDE 的下载地址是 http://www.cocos2d-x.org/download,在浏览器中的页面如图 3-4 所示。选择合适的文件下载,目前包括了 Mac OS X 版本和 Windows 版本,注意 Windows 有 32 位和 64 位之分,还有安装(Setup)版本和压缩(Zip)版本之分。

下载 cocos-code-ide-win64-1.0.0-rc1.zip 解压版本,解压后找到 Cocos Code IDE.exe 文件,运行可以启动 Cocos Code IDE 工具,在启动过程中需要选择 Workspace 目录,如图 3-5 所示,Workspace 目录是工程的管理目录,选择好之后单击 OK 按钮,如果该目录不存在则重新创建。

Cocos Code IDE 的具体使用方法下面章节再介绍。

---

① Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言,它只是一个框架和一组服务,用于通过插件组件构建开发环境。幸运的是,Eclipse 附带了一个标准的插件集,包括 Java 开发工具(Java Development Kit,JDK)。——引自于百度百科 http://baike.baidu.com/subview/23576/9374802.htm

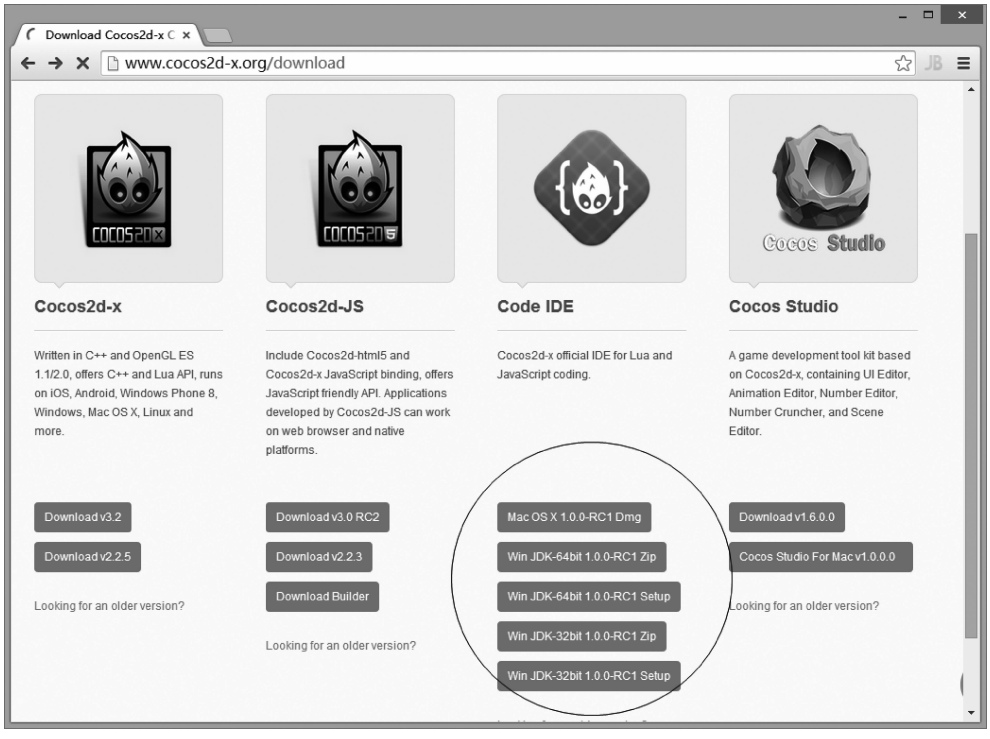


图 3-4　下载 Cocos Code IDE

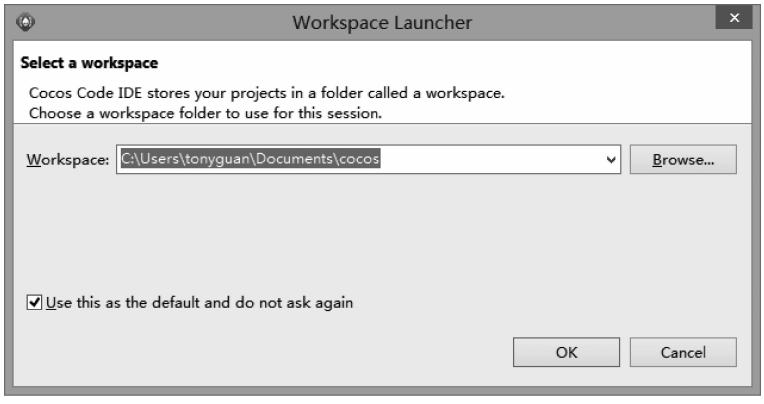


图 3-5　选择 Workspace

3.3.2　下载和使用 Cocos2d-x Lua 官方案例

首先到 Cocos2d-x 官方网站下载 Cocos2d-x 开发包。Cocos2d-x3.2 下载解压后的目录结构如图 3-6 所示。

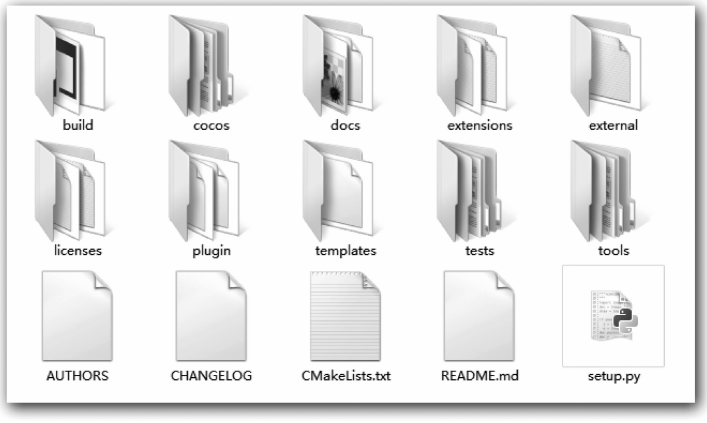


图 3-6　Cocos2d-x 3.2 开发包内容

如果要运行官方的案例可以进入到 build 目录，build 目录中的内容如图 3-7 所示，这里包含了各个平台编译和运行案例的工程等文件，其中 cocos2d_tests.xcodeproj 文件是

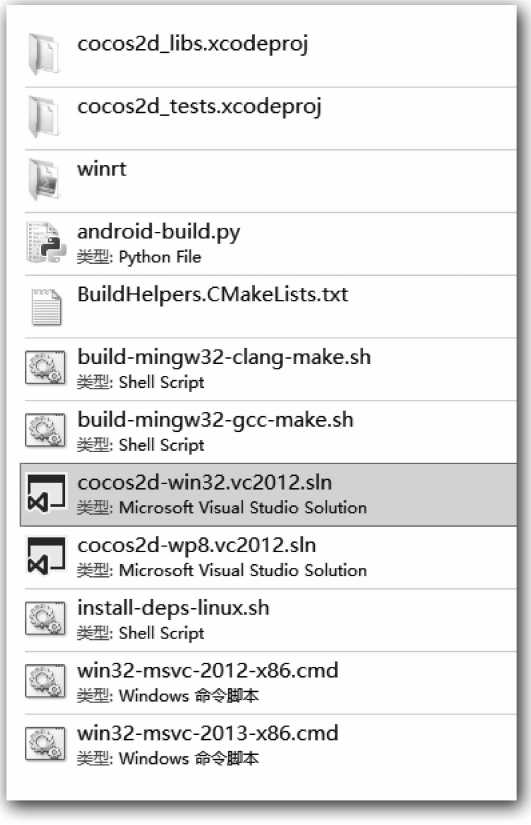


图 3-7　build 目录内容

Cocos2d-x 案例的 Xcode 工程文件，cocos2d-win32. vc2012. sln 文件是 Cocos2d-x 案例 Win32 平台下 Visual Studio 2012 解决方案文件，另外的 cocos2d-wp8. vc2012. sln 文件是 Cocos2d-x 案例 Windows Phone 8 平台下 Visual Studio 2012 解决方案文件。

如果在 Window 下学习和开发，一般运行 cocos2d-win32. vc2012. sln 解决方案就可以了。如果启动 cocos2d-win32. vc2012. sln 解决方案，进入如图 3-8 所示的 Visual Studio 2012 界面，其中的 lua-tests 工程是 Cocos2d-x 官方提供的 Cocos2d-x Lua 案例工程，需要选中 lua-tests 工程，右击选择“设置启动项目”，然后运行上方工具栏中的运行调试按钮▶，运行 lua-tests 工程。

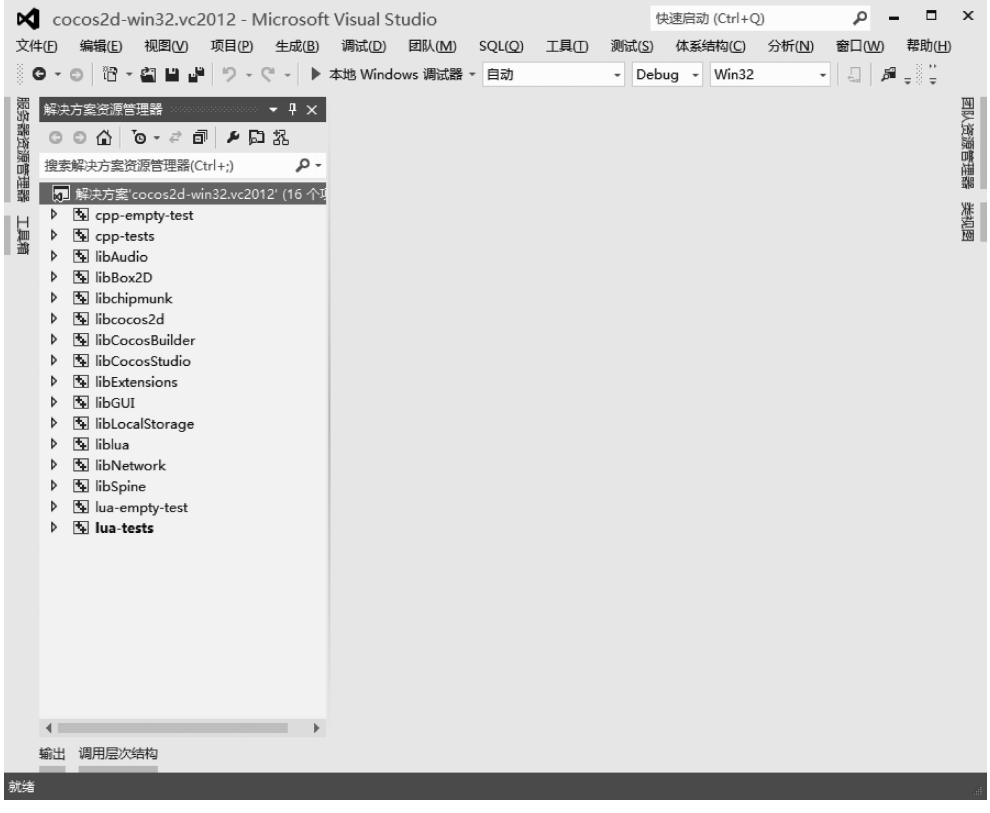
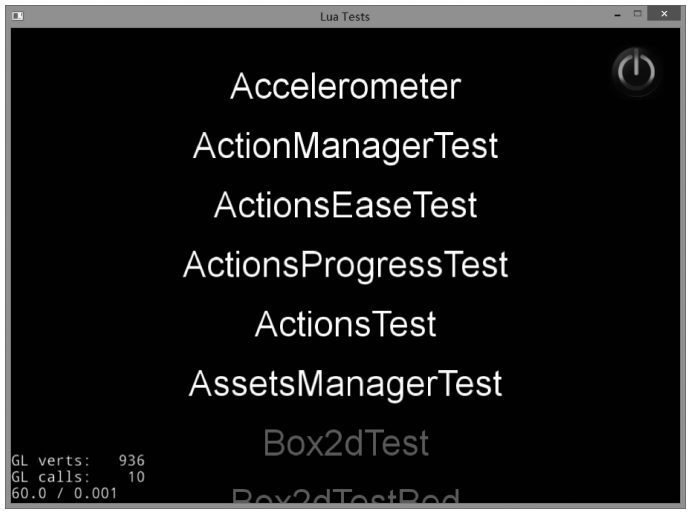


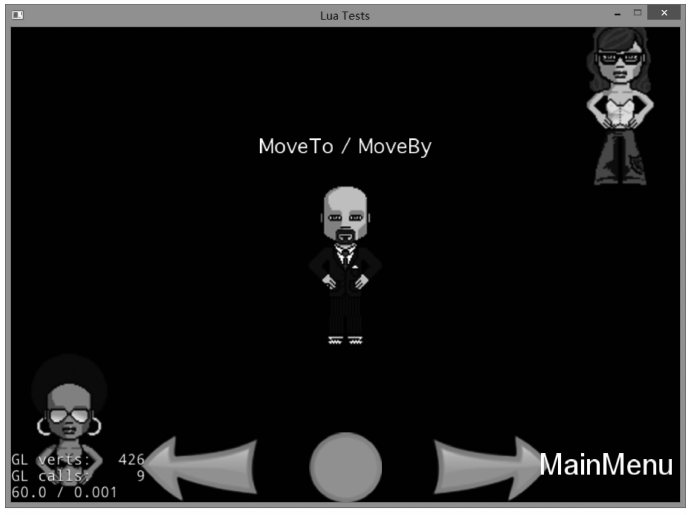
图 3-8　Cocos2d-x Lua 案例

首次运行编译 Cocos2d-x 时间会长一些，运行起来之后会看到一个 Windows 的窗口（如图 3-9(a)所示），单击其中的一个菜单项可以运行相应的示例（如图 3-9(b)所示）。

如果想查看 lua-tests 源代码，需要到＜Cocos2d-x 引擎目录＞\tests\lua-tests 目录下，使用文本编辑工具打开 Lua 代码文件查看。



(a)



(b)

图 3-9　运行案例

3.4　第一个 Cocos2d-x Lua 游戏

编写的第一个 Cocos2d-x Lua 程序命名为 HelloLua，从该工程开始学习其他的内容。

3.4.1　创建工程

创建 Cocos2d-x Lua 工程可以通过 Cocos2d-x 提供的命令工具 Cocos 实现，但这种方式不能与 Cocos Code IDE 集成开发工具很好地集成，不便于程序编写和调试。由于 Cocos

Code IDE 工具是 Cocos2d-x 开发的专门为 Cocos2d-JS 和 Cocos2d-x Lua 开发设计的，因此使用 Cocos Code IDE 工具创建 Cocos2d-x Lua 工程很方便。

首先需要在 Cocos Code IDE 工具中先配置 Lua 框架，打开 Cocos Code IDE 工具，选择菜单 Window→Preferences，弹出对话框如图 3-10 所示，选择 Cocos→Lua 在右边的 Lua Frameworks，选择＜Cocos2d-x 引擎目录＞。

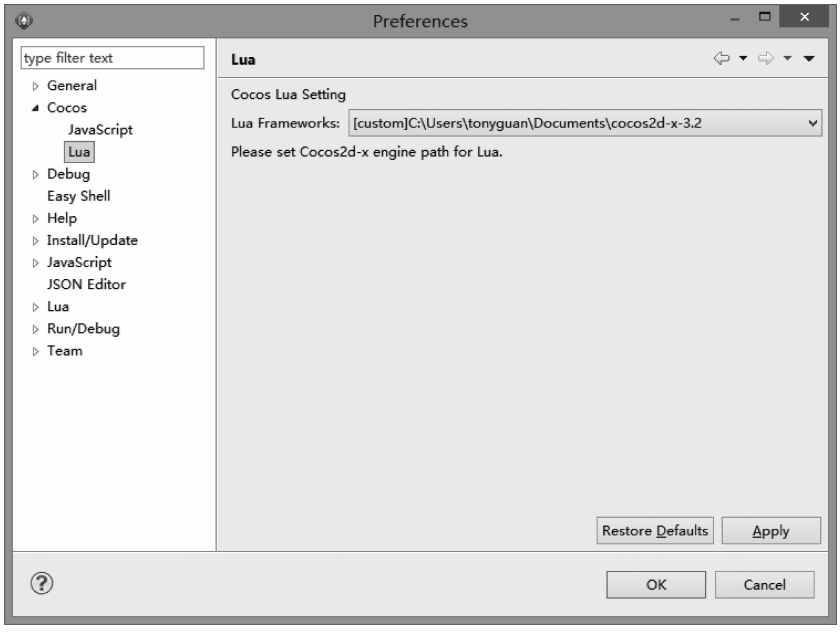


图 3-10　配置 Lua 框架

Lua 框架配置不需要每次都进行，只需要在最开始时配置，但创建工程的时候，Cocos Code IDE 工具会从这个 Lua 框架目录中创建工程文件。

接下来就可以创建 Lua 工程了，选择菜单 File→New→Project，如图 3-11 所示，弹出项目类型选择对话框。

选中 CocosLuaProject，然后单击 Next 按钮，弹出如图 3-12 所示的对话框。在 Project Name 项目中输入工程名称，Create Project in Workspace 是在 Workspace 目录中创建工程，需要选中该项目，Create From Existing Resource 项目选中可以在已经存在的工程创建，现在不需要选中该项目。

单击 Next 按钮进入如图 3-13 所示的配置运行环境对话框，在该对话框中可以配置项目运行时信息。Orientation 是配置模拟器的朝向，其中 landscape 是横屏显示，portriat 是竖屏显。Desktop Runtime Settings 中的 Title 是设置模拟器的标题，Desktop Windows initialize Size 是设置模拟器的大小。Add Native Codes 是设置添加本地代码到工程。最后单击 Finish 按钮完成创建操作，创建好工程之后如图 3-14 所示。

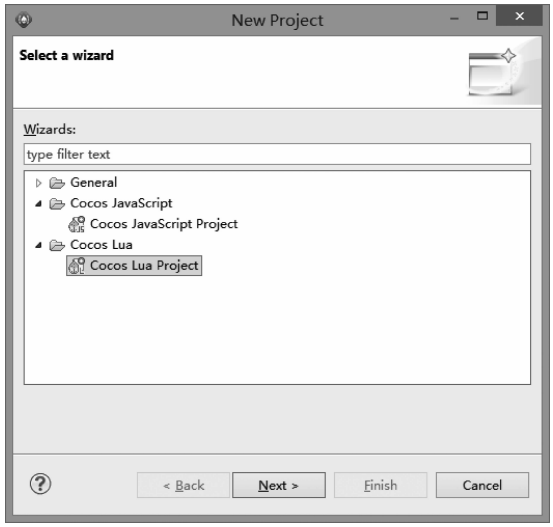


图 3-11 项目类型选择对话框

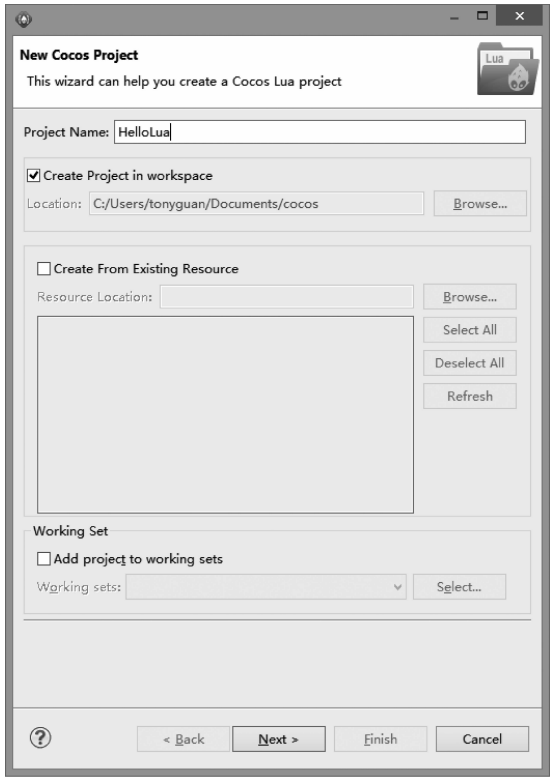


图 3-12 新建项目对话框

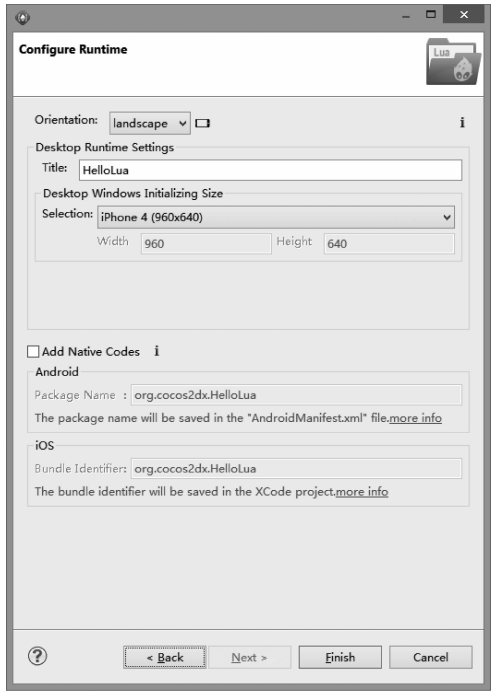


图 3-13　配置运行环境对话框

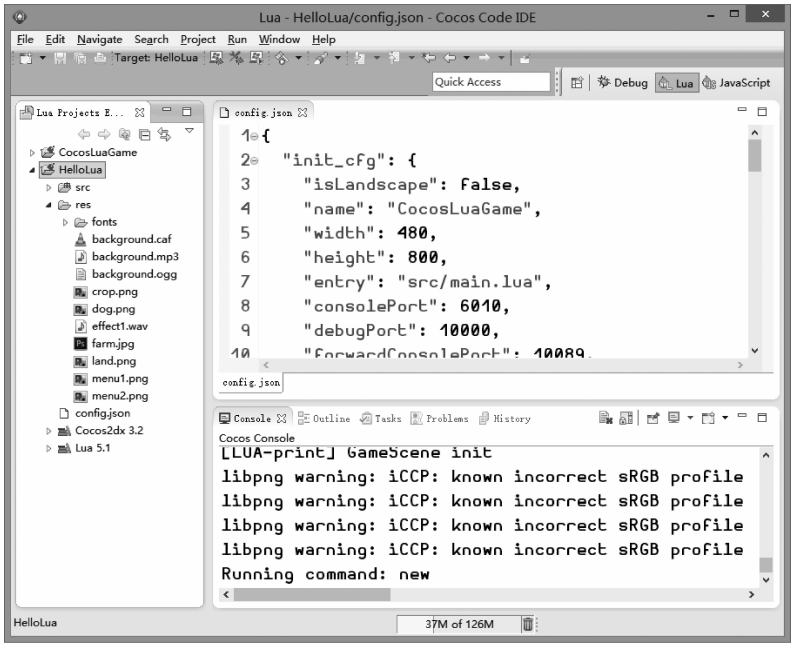


图 3-14　创建工程成功界面

3.4.2　Cocos Code IDE 中运行

创建好工程后可以测试一下，在左边的工程导航面板中选中 HelloLua 工程，右击菜单选择 Run As→Cocosluainding 运行刚刚创建的工程，运行结果如图 3-15 所示。

图 3-15　运行工程界面

编写的程序代码在 src 目录下，在本例中，Lua 文件负责处理如图 3-16 所示的场景界面逻辑。如果想调试程序，可以设置断点，如图 3-17 所示。单击行号之前的位置。

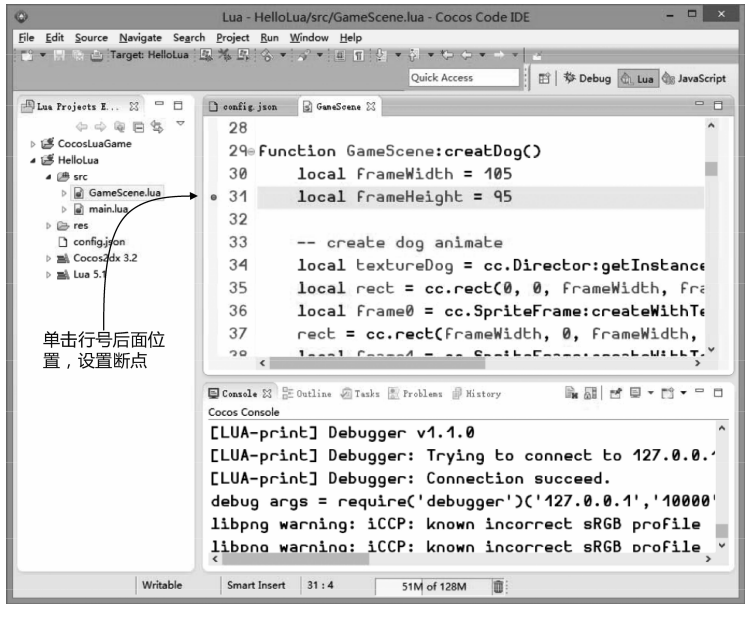


图 3-16　设置断点

右击选择 Debug As→CocosLuabinding 菜单。如图 3-17 所示，程序运行到第 31 行挂起，并进入调试视图，在调试视图中可以查看程序运行的堆栈、变量、断点、计算表达式和单步执行程序等操作。

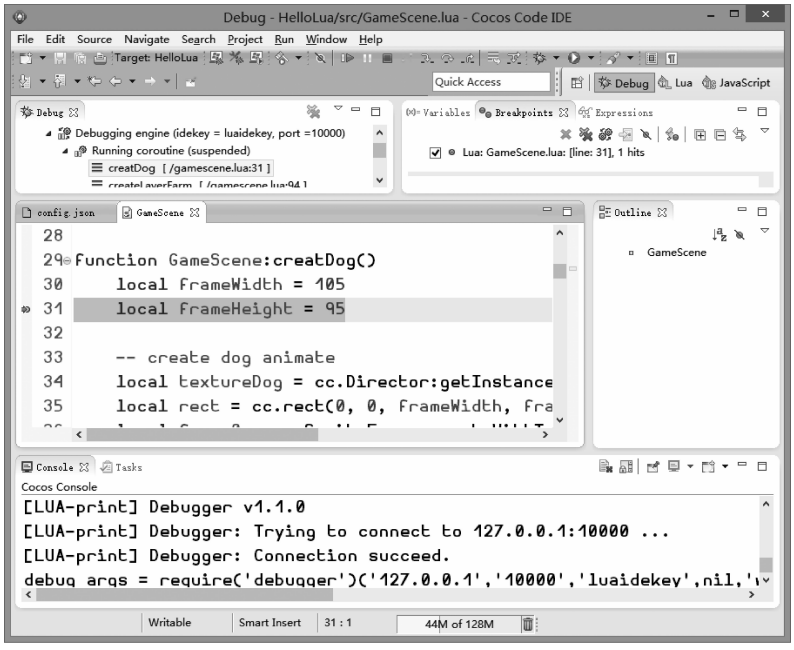


图 3-17　运行到断点挂起

3.4.3　工程文件结构

创建的 HelloLua 工程已经能够运行起来了，下面介绍一下 HelloLua 工程中的文件结构，使用 Cocos Code IDE 打开 HelloLua 工程，左侧的导航面板如图 3-18 所示。

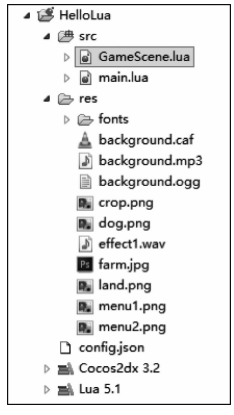


图 3-18　HelloLua 工程中的文件结构

在如图 3-18 所示的导航面板中，res 文件夹存放资源文件，src 文件夹是主要的程序代码 main.lua 和 GameScene.lua，其中 main.lua 是程序入口文件，Cocos2d-x 会在底层绑定该文件，并且启动和运行它。在 GameScene.lua 中实现了游戏场景。

3.4.4　代码解释

HelloLua 工程中主要有两文件，下面详细解释它们内部的代码。

1. main.lua 文件

main.lua 是程序入口文件，代码如下：

```
require "Cocos2d"                                                    ①

-- cclog
local cclog = function(...)                                          ②
    print(string.format(...))                                        ③
end

-- for CCLuaEngine traceback
function __G__TRACKBACK__(msg)                                       ④
    cclog("----------------------------------------")
    cclog("LUA ERROR: " .. tostring(msg) .. "\n")
    cclog(debug.traceback())
    cclog("----------------------------------------")
    return msg
end

local function main()                                                ⑤
    collectgarbage("collect")                                        ⑥
    -- avoid memory leak
    collectgarbage("setpause", 100)                                  ⑦
    collectgarbage("setstepmul", 5000)                               ⑧

    cc.FileUtils:getInstance():addSearchPath("src")
    cc.FileUtils:getInstance():addSearchPath("res")
    cc.Director:getInstance():getOpenGLView():setDesignResolutionSize(480, 320, 0)

    -- create scene
    local scene = require("GameScene")                               ⑨
    local gameScene = scene.create()                                 ⑩
    gameScene:playBgMusic()

    if cc.Director:getInstance():getRunningScene() then              ⑪
        cc.Director:getInstance():replaceScene(gameScene)
    else
        cc.Director:getInstance():runWithScene(gameScene)
    end
```

```
end


local status, msg = xpcall(main, __G__TRACKBACK__)                    ⑫
if not status then
    error(msg)
end
```

上述第①行代码 require 是在加载 Cocos2d 模块，并且可以避免重复加载。第②行代码是声明 cclog 函数，该函数的作用是输出日志信息。第③行代码 print(string.format(...))是输出函数。

第④行代码是声明__G__TRACKBACK__(msg)函数，在程序出错的时候由第⑫行的 xpcall 调用，并输出堆栈信息。日志输出的堆栈信息如下：

```
[LUA-print] stack traceback:
    [string ".\src/main.lua"]:13: in function '__index'
    [string ".\GameScene.lua"]:52: in function <[string ".\GameScene.lua"]:49>
[LUA-print] ----------------------------------------
```

第⑤行代码是 main()函数，它是由第⑪行的 xpcall 函数调用。第⑥行代码 collectgarbage("collect")中的 collectgarbage 是垃圾收集器的通用接口函数，用于操作垃圾收集器，其他的定义如下：

```
collectgrabage(opt[,arg])
```

其中，opt 参数是操作方法标志，标志包括如下。

- collect：执行一次全垃圾收集周期，见第⑥行代码。
- stop：停止垃圾收集器。
- restart：重启垃圾收集器。
- count：返回当前 Lua 中使用的内存数量，单位 KB。
- step：单步执行一个垃圾收集，步长中的 size 属性是由参数 arg 指定，如果完成一次收集周期，将返回 true。
- setpause：设置 arg/100 的值作为垃圾收集暂停时长，见第⑦行代码。
- setstepmul：设置 arg/100 的值，作为步长的增幅，即新步长＝旧步长＊(arg/100)，见第⑧行代码。

上述第⑨行代码 local scene＝require("GameScene")加载 GameScene 模块，返回值是 table 类型全局变量。第⑩行代码 local gameScene＝scene.create()通过静态 create()函数创建 GameScene 场景。

第⑪行代码 cc.Director:getInstance():getRunningScene()判断是否有一个场景正在运行，如果有场景运行则通过 cc.Director:getInstance():replaceScene(gameScene)语句使用 gameScene 场景替换当前场景，否则通过 cc.Director:getInstance():runWithScene

(gameScene)语句运行 gameScene 场景，无论是 replaceScene 还是 runWithScene 函数游戏都会是进入到 gameScene 场景。

第⑫行代码 local status, msg＝xpcall(main, __G__TRACKBACK__)中的 xpcall 函数由 Lua 提供，用于调用其他函数，并且可以捕获到错误，xpcall 函数的定义如下：

```
xpcall (f, err)
```

其中 f 参数是要调用的函数，err 是捕获到错误的时候调用的函数。返回值 status 是错误状态，msg 是错误消息。

事实上第⑫行代码 local status, msg＝xpcall(main, __G__TRACKBACK__)才是程序的入口。由它调用 main()函数，如图 3-19 所示的调用堆栈中，能够看出它们的调用顺序。

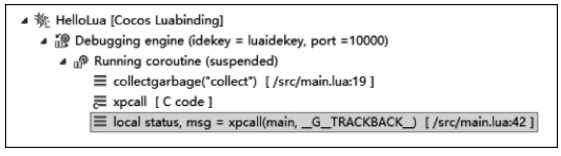


图 3-19　调用堆栈

2. GameScene.lua 文件

GameScene.lua 负责创建游戏主场景，如图 3-15 所示的场景就是在 GameScene.lua 中实现的，GameScene.lua 主要代码如下：

```
require "Cocos2d"
require "Cocos2dConstants"
-- 声明 GameScene 类
local GameScene = class("GameScene",function()                    ①
    return cc.Scene:create()
end)
-- 静态创建函数
function GameScene.create()                                       ②
    local scene = GameScene.new()                                 ③
    scene:addChild(scene:createLayerFarm())                       ④
    scene:addChild(scene:createLayerMenu())                       ⑤
    return scene
end

-- 构造函数
function GameScene:ctor()                                         ⑥
    self.visibleSize = cc.Director:getInstance():getVisibleSize()
    self.origin = cc.Director:getInstance():getVisibleOrigin()
    self.schedulerID = nil
end
-- 播放背景音乐
```

```
function GameScene:playBgMusic()
    local bgMusicPath = cc.FileUtils:getInstance():fullPathForFilename("background.mp3")
    cc.SimpleAudioEngine:getInstance():playMusic(bgMusicPath, true)
    local effectPath = cc.FileUtils:getInstance():fullPathForFilename("effect1.wav")
    cc.SimpleAudioEngine:getInstance():preloadEffect(effectPath)
end
-- 创建 Dog 精灵
function GameScene:creatDog()                                                    ⑦
   …

    local spriteDog = cc.Sprite:createWithSpriteFrame(frame0)
…

    return spriteDog
end

-- create farm 创建农场层
function GameScene:createLayerFarm()                                             ⑧
    local layerFarm = cc.Layer:create()                                          ⑨
    …

    return layerFarm
end

-- create menu 创建菜单层
function GameScene:createLayerMenu()                                             ⑩

    local layerMenu = cc.Layer:create()
    …

    return layerMenu
end

return GameScene                                                                 ⑪
```

在 GameScene.lua 中创建 GameScene 场景，并在场景中添加了农场层和菜单层。第①行代码是声明 GameScene 场景类，class("GameScene",function(){…})函数是由 Cocos2d-x Lua 引擎提供的，可以通过 Lua 创建对象。class 函数定义如下：

```
class(classname, super)
```

其中参数 classname 是函数名，它是字符串类型，super 是调用父类构造函数。

第②行代码声明 GameScene.create()静态函数，在 main.lua 中通过 scene.create()语句调用。第③行代码 local scene＝GameScene.new()是创建 GameScene 创建对象，new()函数会调用第⑥行的 GameScene:ctor()函数，ctor()是构造函数，用来初始化 GameScene 场景对象。第④行代码是调用 GameScene 场景对象的 createLayerFarm()函数创建农场层

(见第⑧行代码)。第⑤行代码是调用 GameScene 场景对象的 createLayerMenu()函数创建菜单层(见代码第⑩行)。

第⑦行代码函数是创建 Dog 精灵,使用 cc. Sprite:createWithSpriteFrame(frame0)语句创建精灵对象,将在后面详细介绍。

在创建农场层函数 createLayerFarm()中的⑨行代码 local layerFarm = cc. Layer:create()是创建层对象,将在后面详细介绍。

第⑪行代码返回 GameScene 变量,它是 table 类型,在 main()函数中调用 local scene= require("GameScene")语句时候返回。

3.5　重构 HelloLua

在 3.4 节介绍了通过 Cocos Code IDE 工具创建一个 Cocos2d-x Lua 工程,但是该工程还是有点复杂,对于初学者不容易读懂其中的代码。在本节中重新介绍一个 HelloLua 工程,在后面的章节中将以此为模板编写程序代码。

该实例运行结果如图 3-20 所示,场景中有一个 HelloWorld 标签和一个图片。

图 3-20　HelloLua 实例

修改 main.lua 主要代码如下:

```
require "Cocos2d"

-- cclog
cclog = function(...)                                                    ①
    print(string.format(...))
```

```
end

-- for CCLuaEngine traceback
function __G__TRACKBACK__(msg)
    cclog("----------------------------------------")
    cclog("LUA ERROR: " .. tostring(msg) .. "\n")
    cclog(debug.traceback())
    cclog("----------------------------------------")
    return msg
end

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(960, 640, 0)

    --create scene
    local scene = require("GameScene")
    local gameScene = scene.create()
        --gameScene:playBgMusic()                                          ②

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

end

local status, msg = xpcall(main, __G__TRACKBACK__)
if not status then
    error(msg)
end
```

上述第①行代码 cclog 函数原来是 local 修饰，这里删除了 local，说明该函数是全局函数，可以在其他模块中使用 cclog 函数。第②行代码中暂时注释 gameScene:playBgMusic() 代码。

修改 GameScene.lua 主要代码如下：

```
require "Cocos2d"
require "Cocos2dConstants"
```

```
size = cc.Director:getInstance():getWinSize()

local GameScene = class("GameScene",function()
    return cc.Scene:create()
end)

function GameScene.create()
    local scene = GameScene.new()
    scene:addChild(scene:createLayer())                                    ①
    return scene
end

function GameScene:ctor()

end

-- create layer
function GameScene:createLayer()                                           ②
    cclog("GameScene init")
    local layer = cc.Layer:create()

    local label = cc.LabelTTF:create("Hello World", "Arial", 46)           ③
    label:setPosition(cc.p(size.width/2,
                           size.height - label:getContentSize().height))   ④

    layer:addChild(label)                                                  ⑤

    local bg = cc.Sprite:create("HelloWorld.png")                          ⑥
    bg:setPosition(cc.p(size.width/2, size.height/2))                      ⑦
    layer:addChild(bg)                                                     ⑧

    return layer
end

return GameScene
```

我们只保留了一个创建层函数 createLayer()，在第①行代码创建层，第②行代码是声明创建成函数，第③行代码创建 LabelTTF 标签对象，第④行代码是设置标签对象的位置，第⑤行代码将标签对象添加到当前层中，第⑥行代码是创建精灵对象，第⑦行代码是设置精灵的位置，第⑧行代码是将精灵对象添加到当前层中。

3.6　Cocos2d-x Lua 核心概念

Cocos2d-x Lua 中有很多概念，这些概念很多来源于动画、动漫和电影等行业，例如导演、场景和层等概念，当然也有些有传统的游戏的概念。Cocos2d-x Lua 中核心概念如下：

- 导演。

- 场景。
- 层。
- 节点。
- 精灵。
- 菜单。
- 动作。
- 效果。
- 粒子运动。
- 地图。
- 物理引擎。

本节介绍导演、场景、层、精灵、菜单以及对应的类，由于节点概念很重要，将会在下面一节详细介绍。而其他的概念在后面的章节中介绍。

3.6.1 导演

导演类 Director 用于管理场景对象，采用单例设计模式，在整个工程中只有一个实例对象。由于是单例模式，能够保存一致的配置信息，便于管理场景对象。获得导演类 Director 实例语句如下：

```
local director = cc.Director:getInstance()
```

其中 cc 是 Cocos2d-x Lua 中类的命名空间，Director 是导演类，getInstance()函数获得调用实例。

导演对象职责如下：

- 访问和改变场景。
- 访问配置信息。
- 暂停、继续和停止游戏。
- 转换坐标。

Director 类图如图 3-21 所示。

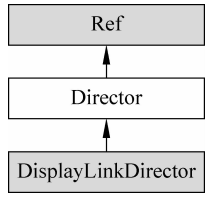


图 3-21 Director 类图运行结果

从如图 3-21 所示的类图中还可以看到它有一个子类是 DisplayLinkDirector。

3.6.2 场景

场景类 Scene 是构成游戏的界面，类似于电影中的场景。场景大致可以分为以下几类：

- 展示类场景。播放视频或简单地在图像上输出文字，用来实现游戏的开场介绍、胜利和失败提示、帮助介绍。
- 选项类场景。主菜单、设置游戏参数等。
- 游戏场景。游戏的主要内容。

场景类 Scene 的类图如图 3-22 所示，从类图可见 Scene 继承了 Node 类，Node 类是一

个重要的类，很多类都从 Node 类派生而来，其中有 Scene、Layer 等。

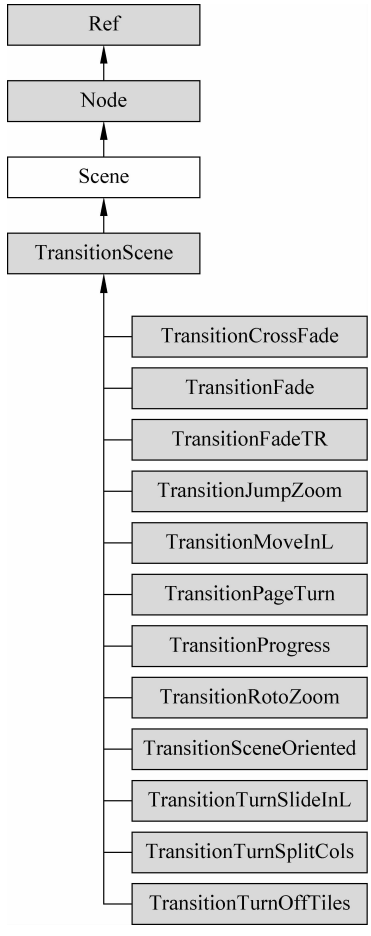


图 3-22　Scene 类图

3.6.3　层

层是写游戏的重点，大约 99%以上的时间是在层上实现游戏内容。层的管理类似于 Photoshop 中的图层，它也是一层一层叠在一起。如图 3-23 所示是一个简单的主菜单界面，它是由三个层叠加实现的。

为了让不同的层可以组合产生统一的效果，这些层基本上都是透明或者半透明的。层的叠加是有顺序的，如图 3-23(b)所示从上到下依次是菜单层→精灵层→背景层。Cocos2d-x Lua 是按照这个次序来叠加界面的。这个次序同样用于事件响应机制，即菜单层最先接收到系统事件，然后是精灵层，最后是背景层。在事件的传递过程中，如果有一个层处理了该事件，则排在后面的层将不再接收该事件了。每一层又可以包括很多各式各样的内容要素，如文本、链接、精灵、地图等内容。

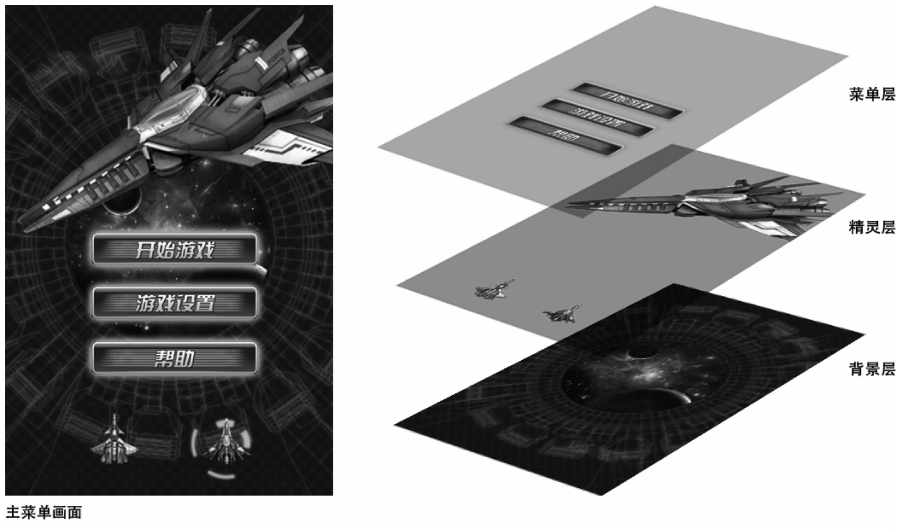


图 3-23　层叠加

层类 Layer 的类图如图 3-24 所示。

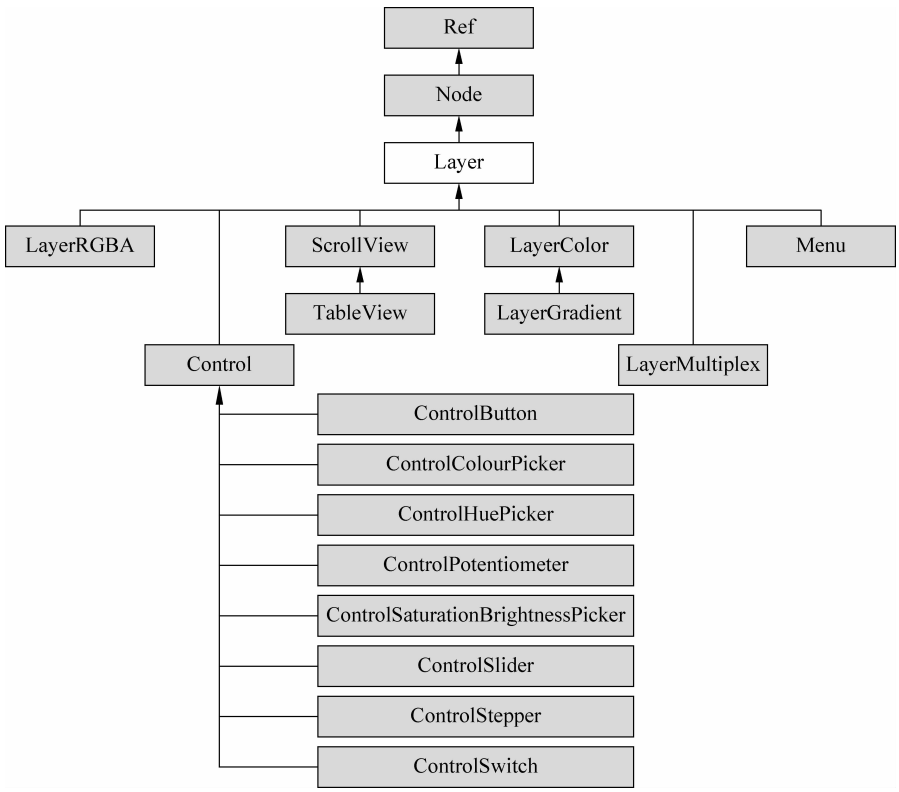


图 3-24　Layer 类图

3.6.4 精灵

精灵类 Sprite 是游戏中非常重要的概念，它包括了敌人、控制对象、静态物体和背景等。通常情况它会进行运动，运动方式包括移动、旋转、放大、缩小和动画等。

Sprite 类图如图 3-25 所示，从图中可见 Sprite 是 Node 子类，Sprite 包含很多类型，例如物理引擎精灵类 PhysicsSprite 也都属于精灵。

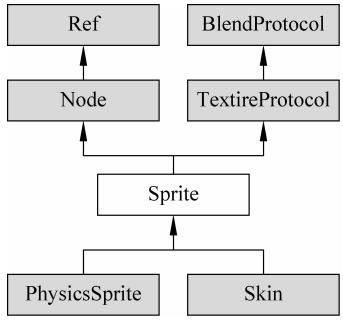


图 3-25 Sprite 类图

3.6.5 菜单

菜单在游戏中是非常重要的概念，它提供操作的集合，在 Cocos2d-x Lua 中菜单类是 Menu，Menu 类图如图 3-26 所示，从类图可见 Menu 类派生于 Layer。

在菜单中又包含了菜单项 MenuItem，MenuItem 类图如图 3-27 所示，从图 3-27 中可见菜单项 MenuItem 有很多种形式的子类，如 MenuItemLabel、MenuItemSprite 和 MenuItemToggle，它表现出不同的效果。每个菜单项都有 3 个基本状态：正常、选种和禁止。

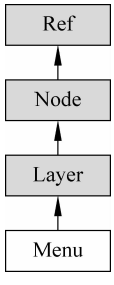


图 3-26 Menu 类图

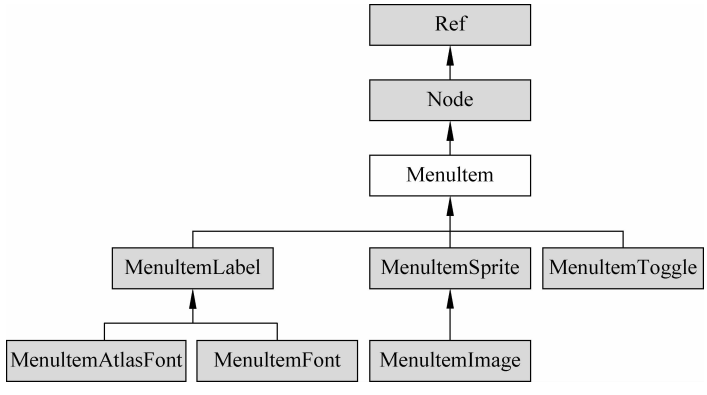


图 3-27 MenuItem 类图

3.7 Node 与 Node 层级架构

Cocos2d-x Lua 采用层级(树状)结构管理场景、层、精灵、菜单、文本、地图和粒子系统等节点(Node)对象。一个场景包含了多个层，一个层又包含多个精灵、菜单、文本、地图和粒子系统等对象。层级结构中的节点可以是场景、层、精灵、菜单、文本、地图和粒子系统等任何对象。

节点的层级结构如图 3-28 所示。

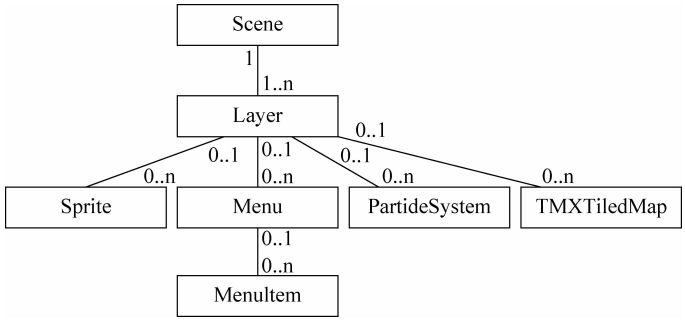


图 3-28 节点的层级结构

这些节点有一个共同的父类 Node，Node 类图如图 3-29 所示。Node 类是 Cocos2d-x Lua 最为重要的根类，它是场景、层、精灵、菜单、文本、地图和粒子系统等类的根类。

3.7.1 Node 中重要的操作

Node 作为根类，它有很多重要的函数，下面分别介绍：

- 创建节点。local childNode＝cc.Node:create()。
- 增加新的子节点。node:addChild (childNode, 0, 123)，第 2 个参数是 Z 轴绘制顺序，第 3 个参数是标签。
- 查找子节点。local node＝node:getChildByTag(123)，通过标签查找子节点。
- node:removeChildByTag(123, true)通过标签删除子节点，并停止该节点上的一切动作。
- node:removeChild(childNode, true)删除 childNode 节点，并停止该子节点上的一切动作。
- node:removeAllChildrenWithCleanup(true)删除 node 节点的所有子节点，并停止这些子节点上的一切动作。
- node:removeFromParentAndCleanup(true)从父节点删除 node 节点，并停止该节点上的一切动作。

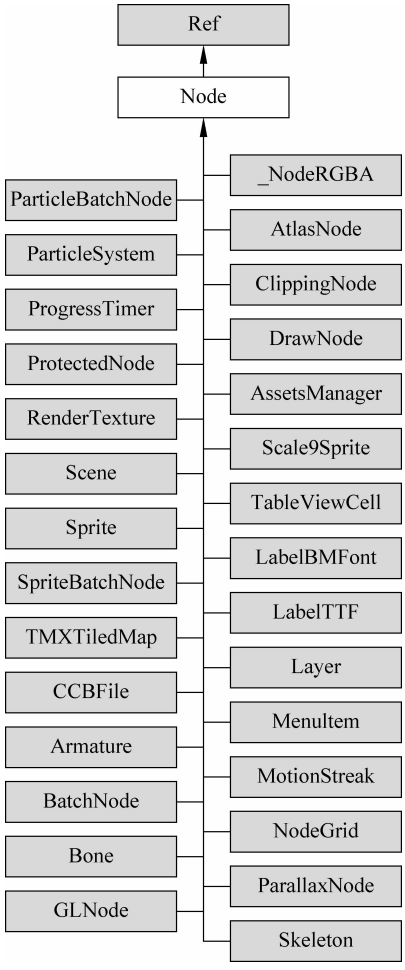


图 3-29 Node 类图

3.7.2 Node 中重要的属性

此外，Node 还有两个非常重要的属性：position 和 anchorPoint。

position(位置)属性是 Node 对象的实际位置。position 属性往往还要配合使用 anchorPoint 属性，为了将一个 Node 对象(标准矩形图形)精准地放置在屏幕某一个位置上，需要设置该矩形的锚点，anchorPoint 是相对于 position 的比例，默认是(0.5,0.5)。

如图 3-30 所示是 anchorPoint 为(0.5,0.5)情况，这是默认情况。

如图 3-31 所示是 anchorPoint 为(0.0,0.0)情况。

如图 3-32 所示是 anchorPoint 为(1.0,1.0)情况。

如图 3-33 所示是 anchorPoint 为(0.66, 0.5)情况。

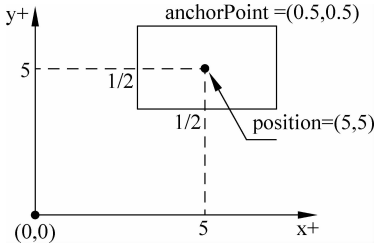


图 3-30　anchorPoint 为(0.5,0.5)

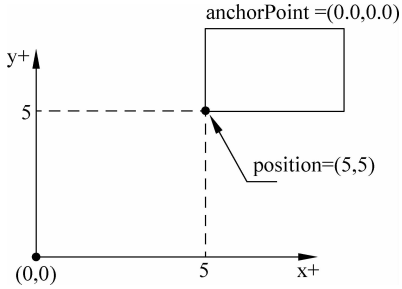


图 3-31　anchorPoint 为(0.0,0.0)

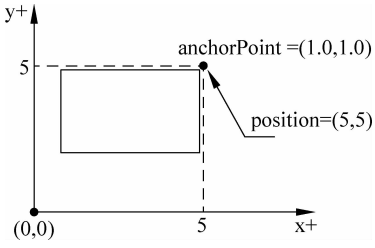


图 3-32　anchorPoint 为(1.0,1.0)

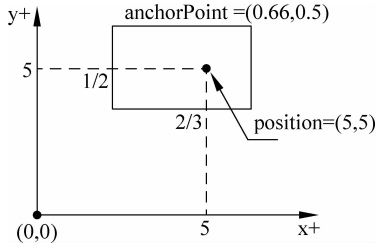


图 3-33　anchorPoint 为(0.66, 0.5)

为了进一步了解 anchorPoint 使用，修改 HelloLua 实例，修改 GameScene.lua 的 GameScene:createLayer()函数如下，其中加粗字体显示的是添加的代码。

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

    local label = cc.LabelTTF:create("Hello World", "Arial", 46)
    label:setPosition(cc.p(size.width/2,
                           size.height - label:getContentSize().height))
  label:setAnchorPoint(cc.p(1.0, 1.0))
    layer:addChild(label)

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

    return layer
end
```

运行结果如图 3-34 所示，Hello World 标签设置的 anchorPoint 为(1.0,1.0)。

图 3-34　Hello World 标签的 anchorPoint 为(1.0,1.0)

3.7.3　游戏循环与调度

每一个游戏程序都有一个循环在不断运行,它是由导演对象来管理和维护的。如果需要场景中的精灵运动起来,可以在游戏循环中使用定时器(Scheduler)对精灵等对象的运行进行调度。因为 Node 类封装了 Scheduler 类,所以也可以直接使用 Node 中定时器相关函数。

Node 中定时器相关函数主要如下:

- scheduleUpdateWithPriorityLua(nHandler, priority)。每个 Node 对象只要调用该函数,那么这个 Node 对象就会定时地每帧回调用一次 nHandler 函数。priority 是优先级,priority 值越小越先执行。
- unscheduleUpdate ()。停止 scheduleUpdateWithPriorityLua 的调度。

为了进一步了解游戏循环与调度的使用,修改 HelloLua 实例。修改后的 GameScene.lua 文件代码如下:

```
require "Cocos2d"
require "Cocos2dConstants"

size = cc.Director:getInstance():getWinSize()
local label                                                    ①
```

```
local GameScene = class("GameScene",function()
    return cc.Scene:create()
end)

function GameScene.create()
    local scene = GameScene.new()
    scene:addChild(scene:createLayer())
    return scene
end

function GameScene:ctor()

end

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

    label = cc.LabelTTF:create("Hello World", "Arial", 46)
    label:setPosition(cc.p(size.width/2,
        size.height - label:getContentSize().height))
    label:setTag(123)
    label:setAnchorPoint(cc.p(1.0, 1.0))
    layer:addChild(label)

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

    local function update(delta)                                    ②
        local x,y = label:getPosition()
        label:setPosition(cc.p(x + 2, y - 2))
    end

    -- 开始游戏调度
    layer:scheduleUpdateWithPriorityLua(update, 0)                  ③

    function onNodeEvent(tag)                                       ④
        if tag == "exit" then                                       ⑤
            -- 开始游戏调度
            layer:unscheduleUpdate()                                ⑥
        end
    end
    layer:registerScriptHandler(onNodeEvent)                        ⑦

    return layer
```

```
end

return GameScene
```

上述第①行代码定义了模块级标签对象 label。第②行代码定义的 update(delta)函数是调度函数。第③行代码 layer:scheduleUpdateWithPriorityLua(update, 0)是开启游戏调度,按照帧率进行调度,优先级 0 是默认值。

第④行代码是层处理事件回调函数,第⑤行代码是判断是否为退出层事件,如果是退出层事件则调用第⑥行代码停止调度。第⑦行代码 layer:registerScriptHandler(onNodeEvent)是注册层事件监听器。

3.8 Cocos2d-x Lua 坐标系

在图形图像和游戏应用开发中,坐标系是非常重要的,在 Android 和 iOS 等平台应用开发时使用的二维坐标系的原点在左上角。而在 Cocos2d-x Lua 坐标系中,它的原点在左下角,而且 Cocos2d-x Lua 坐标系又分为世界坐标和模型坐标。

3.8.1 UI 坐标

UI 坐标就是 Android 和 iOS 等应用开发时使用的二维坐标系。它的原点在左上角(见图 3-35)。

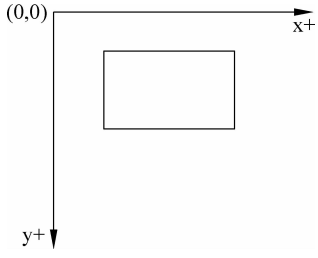


图 3-35 UI 坐标

UI 坐标原点在左上角,x 轴向右为正,y 轴向下为正。在 Android 和 iOS 等平台使用的视图、控件等都是遵守这个坐标系。然而在 Cocos2d-x Lua 默认不是采用 UI 坐标的情况下,有的时候也会用到 UI 坐标,例如在触摸事件发生的时候会获得一个触摸对象(Touch),触摸对象(Touch)提供了很多获得位置信息的函数,如下面代码所示:

```
cc.p touchLocation = touch:getLocationInView()
```

使用 getLocationInView()函数获得触摸点坐标事实上就是 UI 坐标,它的坐标原点在左上角。而不是 Cocos2d-x Lua 默认坐标,可以采用下面的语句进行转换:

```
cc.p touchLocation2 = cc.Director:getInstance():convertToGL(touchLocation)
```

通过上面的语句就可以将触摸点位置从 UI 坐标转换为 OpenGL 坐标，OpenGL 坐标就是 Cocos2d-x Lua 默认坐标。

3.8.2 OpenGL 坐标

上面提到了 OpenGL 坐标是三维坐标。由于 Cocos2d-x Lua 底层采用 OpenGL 渲染，因此默认坐标就是 OpenGL 坐标，只不过只采用两维（x 和 y 轴）。如果不考虑 z 轴，OpenGL 坐标的原点在左下角（见图 3-26）。

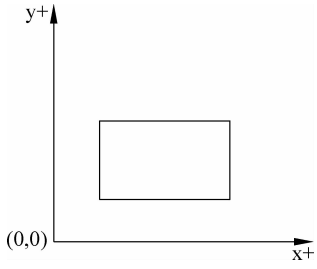


图 3-36 OpenGL 坐标

提示：三维坐标根据 z 轴的指向不同分为左手坐标和右手坐标。右手坐标是 z 轴指向屏幕外，如图 3-37(a)所示。左手坐标是 z 轴指向屏幕里，如图 3-37(b)所示。OpenGL 坐标是右手坐标，而微软平台的 Direct3D[①] 是左手坐标。

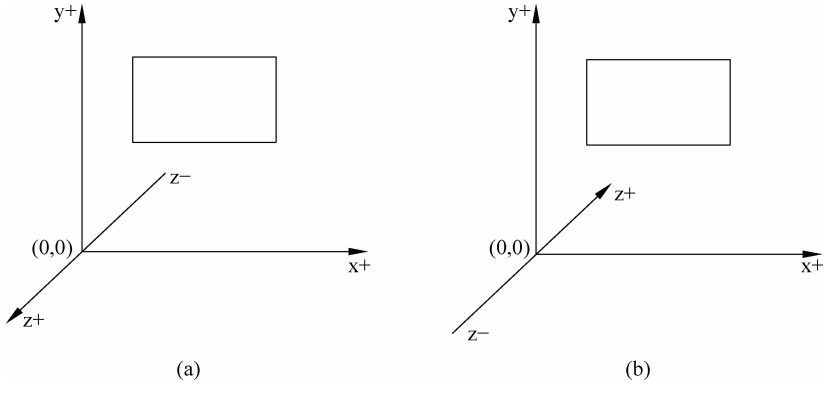


图 3-37 三维坐标

① Direct3D(D3D)是微软公司在 Microsoft Windows 操作系统上所开发的一套 3D 绘图编程接口，是 DirectX 的一部分，目前广为各家显卡所支持。与 OpenGL 同为计算机绘图软件和计算机游戏最常使用的两套绘图编程接口之一。——引自于维基百科 http://zh.wikipedia.org/wiki/Direct3D

3.8.3 世界坐标和模型坐标

由于 OpenGL 坐标分为世界坐标和模型坐标，所以 Cocos2d-x Lua 的坐标也有世界坐标和模型坐标之分。

你是否有过这样的问路经历：张三会告诉你向南走一公里，再向东走 500 米。而李四会告诉你向右走一公里，再向左走 500 米。这里两种说法或许都可以找到你要寻找的地点。张三采用的坐标是世界坐标，把地球作为参照物，表述位置使用地理的东、南、西和北。而李四采用的坐标是模型坐标，让你自己作为参照物，表述位置使用你的左边、你的前边、你的右边和你的后边。

如图 3-38 所示，从图中可以看到 A 的坐标是(5,5)，B 的坐标是(6,4)，事实上这些坐标值就是世界坐标。如果采用 A 的模型坐标来描述 B 的位置，则 B 的坐标是(1,−1)。

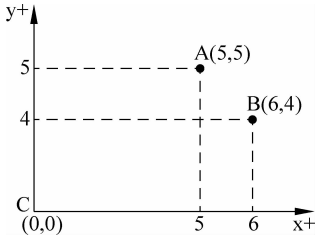


图 3-38 世界坐标和模型坐标

有时候需要将世界坐标与模型坐标互相转换。可以通过 Node 对象的如下函数实现：

- convertToNodeSpace(worldPoint)：将世界坐标转换为模型坐标。
- convertToNodeSpaceAR(worldPoint)：将世界坐标转换为模型坐标，AR 表示相对于锚点。
- convertTouchToNodeSpace(touch)：将世界坐标中触摸点转换为模型坐标。
- convertTouchToNodeSpaceAR(touch)：将世界坐标中触摸点转换为模型坐标，AR 表示相对于锚点。
- convertToWorldSpace(nodePoint)：将模型坐标转换为世界坐标。
- convertToWorldSpaceAR(nodePoint)：将模型坐标转换为世界坐标，AR 表示相对于锚点。

下面通过两个示例了解一下世界坐标与模型坐标互相转换。

1. 世界坐标转换为模型坐标

如图 3-39 所示是世界坐标转换为模型坐标的实例运行结果。

在游戏场景中有两个 Node 对象，其中 Node1 的坐标是(400, 500)，大小是 300×100 像素。Node2 的坐标是(200, 300)，大小也是 300×100 像素。这里的坐标事实上就是世界坐标，它的坐标原点是屏幕的左下角。

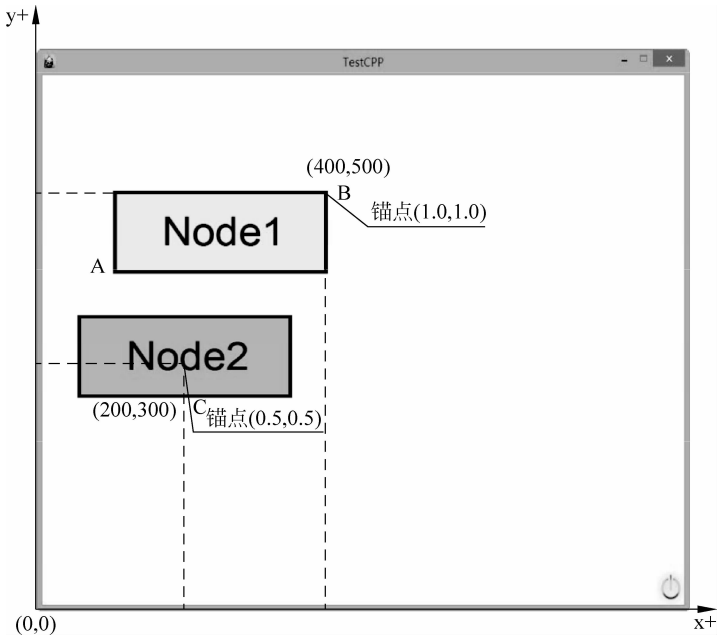


图 3-39　世界坐标转换为模型坐标

编写代码如下：

```
function GameScene:createLayer()
    cclog("GameScene init")
    local layer = cc.Layer:create()
        -- 创建背景
    local bg = cc.Sprite:create("bg.png")                                    ①
    bg:setPosition(cc.p(size.width/2, size.height/2))
    layer:addChild(bg)                                                       ②

    local closeItem = cc.MenuItemImage:create(
        "CloseNormal.png",
        "CloseSelected.png")
    closeItem:setPosition(cc.p(size.width - closeItem:getContentSize().width/2,
                                            closeItem:getContentSize().height/2))
    local menu = cc.Menu:create(closeItem)
    menu:setPosition(cc.p(0, 0))
    layer:addChild(menu)

    -- 创建 Node1
    local node1 = cc.Sprite:create("node1.png")                              ③
    node1:setPosition(cc.p(400,500))
    node1:setAnchorPoint(cc.p(1.0, 1.0))
```

```
    layer:addChild(node1, 0)                                                    ④
        -- 创建 Node2
    local node2 = cc.Sprite:create("node2.png")                                 ⑤
    node2:setPosition(cc.p(200,300))
    node2:setAnchorPoint(cc.p(0.5, 0.5))
    layer:addChild(node2, 0)                                                    ⑥

    local posX,posY = node2:getPosition()
    local point1 = node1:convertToNodeSpace(cc.p(posX,posY))                    ⑦
    local point3 = node1:convertToNodeSpaceAR(cc.p(posX,posY))                  ⑧

    cclog("Node2 NodeSpace = (%f, %f)",point1.x, point1.y)
    cclog("Node2 NodeSpaceAR = (%f, %f)",point3.x,point3.y)

    return layer
end
```

第①～②行代码是创建背景精灵对象，这个背景是一个白色 900×640 像素的图片。第③～④行代码是创建 Node1 对象，并设置了位置和锚点属性。第⑤～⑥行代码是创建 Node2 对象，并设置了位置和锚点属性。第⑦行代码将 Node2 的世界坐标转换为相对于 Node1 的模型坐标。而第⑧行代码是类似的，它是相对于锚点的位置。

运行结果如下：

```
Node2 NodeSpace = (100.000000, -100.000000)
Node2 NodeSpaceAR = (-200.000000, -200.000000)
```

结合图 3-39 解释一下，Node2 的世界坐标转换为相对于 Node1 的模型坐标，就是将 Node1 的左下角作为坐标原点（图 3-39 中的 A 点），不难计算出 A 点的世界坐标是（100，400），那么 convertToNodeSpace 函数就是 A 点坐标减去 C 点坐标，结果是（－100，100）。

而 convertToNodeSpaceAR 函数要考虑锚点，因此坐标原点是 B 点，B 点坐标减去 C 点坐标，结果是（－200，－200）。

2. 模型坐标转换为世界坐标

如图 3-40 所示是模型坐标转换为世界坐标的实例运行结果。

在游戏场景中有两个 Node 对象，其中 Node1 的坐标是（400，500），大小是 300×100 像素。Node2 是放置在 Node1 中的，它对于 Node1 的模型坐标是（0，0），大小是 150×50 像素。

编写代码如下：

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

        -- 创建背景
```

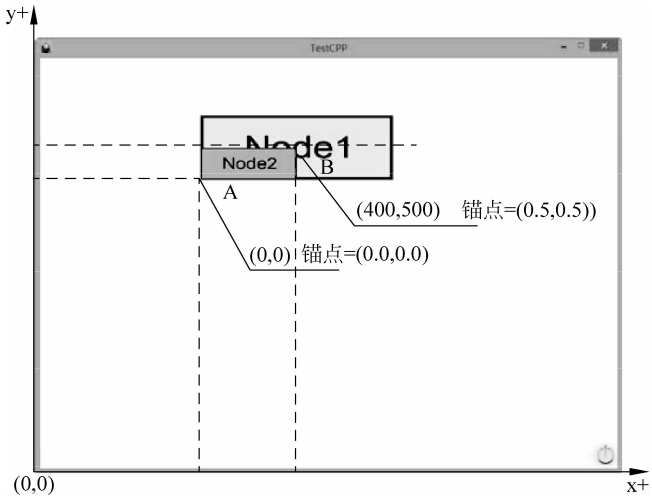


图 3-40 模型坐标转换为世界坐标

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

local closeItem = cc.MenuItemImage:create(
    "CloseNormal.png",
    "CloseSelected.png")
closeItem:setPosition(cc.p(size.width - closeItem:getContentSize().width/2,
                           closeItem:getContentSize().height/2))
local menu = cc.Menu:create(closeItem)
menu:setPosition(cc.p(0, 0))
layer:addChild(menu)

    -- 创建 Node1
local node1 = cc.Sprite:create("node1.png")
node1:setPosition(cc.p(400,500))
layer:addChild(node1, 0)

    -- 创建 Node2
local node2 = cc.Sprite:create("node2.png")
node2:setPosition(cc.p(0.0, 0.0))                                    ①
node2:setAnchorPoint(cc.p(0.0, 0.0))
node1:addChild(node2, 0)                                             ②

local posX,posY = node2:getPosition()
local point1 = node1:convertToWorldSpace(cc.p(posX,posY))            ③
local point3 = node1:convertToWorldSpaceAR(cc.p(posX,posY))          ④
```

```
    cclog("Node2 WorldSpace = (%f, %f)",point1.x, point1.y)
    cclog("Node2 WorldSpaceAR = (%f, %f)",point3.x,point3.y)

    return layer
end
```

上述主要关注第②行代码，它是将 Node2 放到 Node1 中，这是与之前代码的区别。这样第①行代码设置的坐标就变成了相对于 Node1 的模型坐标了。

第③行代码将 Node2 的模型坐标转换为世界坐标。而第④行代码是类似的，它是相对于锚点的位置。

运行结果如下：

```
Node2 WorldSpace = (250.000000,450.000000)
Node2 WorldSpaceAR = (400.000000,500.000000)
```

如图 3-26 所示的位置，可以用世界坐标描述。第①～②行代码修改如下：

```
node2->setPosition(Vec2(250, 450));
node2->setAnchorPoint(Vec2(0.0, 0.0));
this->addChild(node2, 0);
```

本章小结

通过对本章的学习，读者可以了解 Cocos2d-x Lua 开发环境的搭建，以及熟悉 Cocos2d-x Lua 核心概念，这些概念包括导演、场景、层、精灵和菜单等节点对象。此外，还重点学习了 Node 和 Node 层级架构。最后还介绍了 Cocos2d-x Lua 的坐标系。