QuickCocos2dx程序设计

一：介绍及环境搭建

二：Quick基础知识

三：UI控件使用

一：介绍及环境搭建

由于 Cocos2d-x 中使用的是 C++ 语言，而 C++ 又对开发人员要求较高，所以逐渐地，开发者们开始将 Cocos2d-x 的 C++ 接口转成了 Lua 接口，从而衍生出了 Cocos2d-lua 的版本。而 Quick（Quick-Coco2d-x）是 Cocos2d-lua 的一个豪华增强和扩展版本，它重写了支持代码、解决了内存泄露和只能使用全局函数做回调等等问题。Quick 能让开发者使用 Lua 这种简单易懂的脚本语言来编写游戏，并大大提高了开发效率。

我们这里采用的版本为 v3.3 Final，大家可在Cocos引擎中文官网中找到对应的下载地址并安装该程序。

双击安装包进行安装，mac下安装包的格式为dmg,win下安装包的格式为exe，安装完成之后，文件结构如下图1-1所示

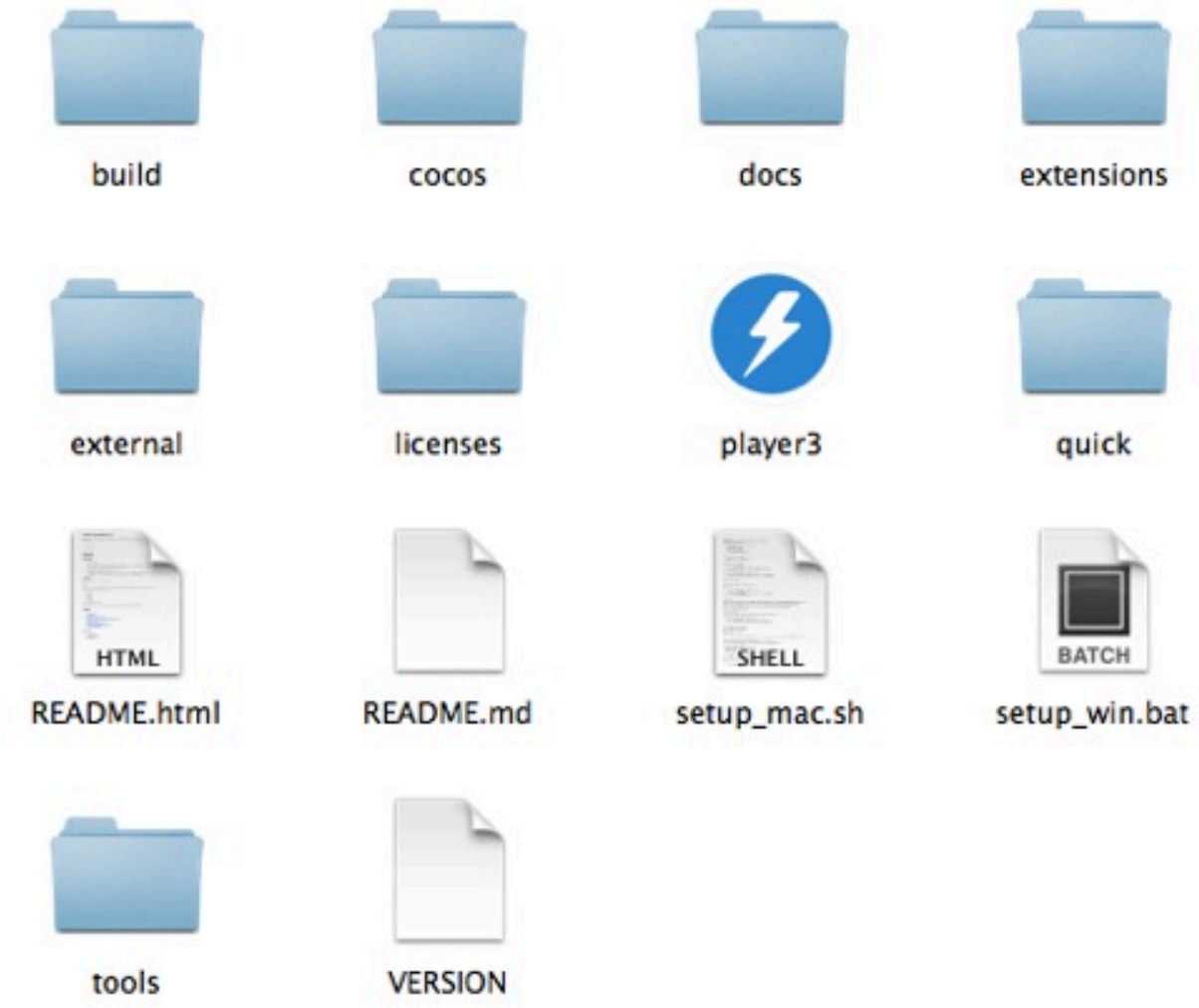

build
cocos
docs
extensions
external
licenses
player3
quick
HTML
README.html
README.md
SHELL
setup_mac.sh
BATCH
setup_win.bat
tools
VERSION

图1-1:文件结构

build：该目录是 Cocos2d-x 的项目存放目录。

cocos：该文件夹中包含了大部分引擎的库文件，其中包括：2d、3d、声音、基础库、数学库、物理库等等一系列相关的类文件。

docs：该文件夹下包含了引擎的API文档、发布文档（最新版本更改介绍，运行环境要求，编译环境要求及如何运行测试用例的相关命令）、Cocos编程规范等等文档。我们可以通过它查看引擎的代码API，以及最新版本更改介绍，quick运行环境要求，编译环境要求及如何运行测试用例的相关命令。里面的文件多是html和md格式的。

extensions：其中主要是GUI扩展库。

external中包含物理引擎第三方库，Box2D和chipmunk；数据库第三方库，sqlite3；网络第三方库，webp,websockets；以及一些其他第三方库，像编码转换库、数据格式库等等。

licenses里面包含了引擎中用到的各种许可证文件。LICENSE_SpiderMonkey，spider引擎中用到的SpiderMonkey-JS运行环境，需要此许可证，该许可证适用于MPL/GPL/LGPL几种许可证LICENSE_chipmunk，LICENSE_JS，LICENSE_lua等等。引擎在这些许可证下可以对相应的源代码进行任意拷贝和修改。

quick：这个是Quick引擎代码。其中包含了创建各个平台新工程的批处理工具，Quick框架的核心目录，2dx和一些其他依赖的c++文件，模版工程，Quick所带的例子等等Quick的核心文件。

README.html / README.md：Quick的使用指南，关于Quick的安装、使用、创建等等信息都可以出这里获取，它其实相当于docs内文件的目录。

setup_mac.sh：搭建Mac开发环境的脚本。

setup_win.bat：搭建Windows开发环境的脚本。

tools：Quick用做luabinding的工具，可用来导出自定义的C++类。

version：版本标示。

Quick 安装完成后，在它的根目录下可以找到有两个名为setup_mac.sh、setup_win.bat的批处理脚本，它们分别是搭建Mac和Windows开发环境的脚本，根据自己系统的需要运行相应的脚本，就可以自动为你完成 Quick 环境的配置。在此之后，我们就可以双击安装目录下的 player3 图标（Windows 下桌面上会生成 player 的快捷键），启动 Quick 自带的模拟器了。在该模拟器界面中，我们可以创建、打开、运行项目，同时还能查看很多 Quick 自带的示例项目。

开发工具选择以及搭建

一般情况下，我们通常都会采用Cocos Code IDE作为开发工具来快速开发游戏，这里Cocos Code IDE是一个基于 Eclipse 的跨平台 IDE，专门为 Cocos2d-x Lua & JavaScript 开发人员准备，通过 IDE 你可以方便的创建游戏工程、编写并且支持在不同平台上调试代码、实时查看代码被改变后的效果，最终直接发布成一个可上架的安装包。但是在该版本存在一部分Bug，所以我们在此选择Sublime + QuickXDev进行我们的开发。

开发工具的安装配置

Sublime Text 是一个具有漂亮的用户界面和强大的功能的代码编辑器，也是HTML和散文先进的文本编辑器，它的很多功能都依赖于其强大的插件系统。Sublime Text支持Lua语言，但它本身不具有像代码提示这样的功能，所以要想用Sublime Text快速的开发Quick-Coco2d-x程序，我们就必须安装强大的QuickXDev插件。
Sublime Text的下载地址为：http://www.sublimetext.com/，下载后直接安装即可。

QuickXDev的下载地址为：http://git.oschina.net/lonewolf/QuickXDev，将它下载解压之后重命名为QuickXDev，然后把该QuickXDev文件夹放入到Sublime Text的Packages目录下（可通过Sublime Text->Preferences->Browse Packages打开）。

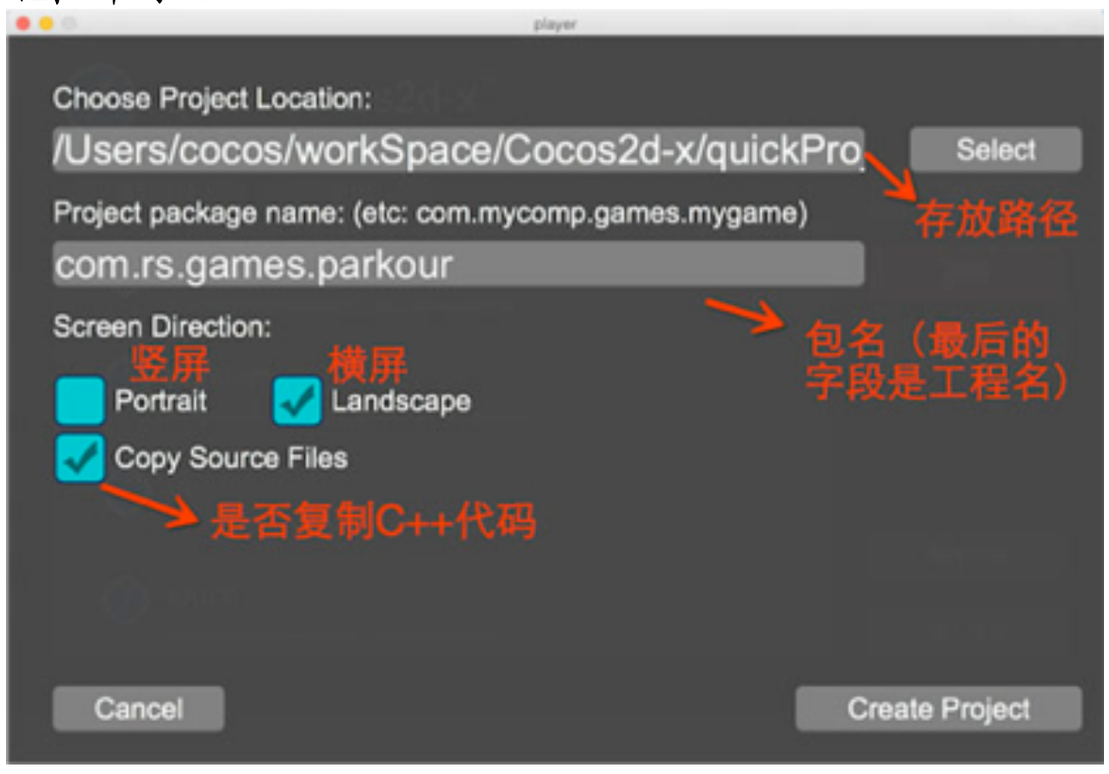
接着依次打开Preferences->Package Settings->QuickXDev->Settings－User，如下图1-2所示

```
{
    "quick_cocos2dx_root":"/Users/zyuq/Documents/quick-3.3",
    "author":"zyuq"
}
```

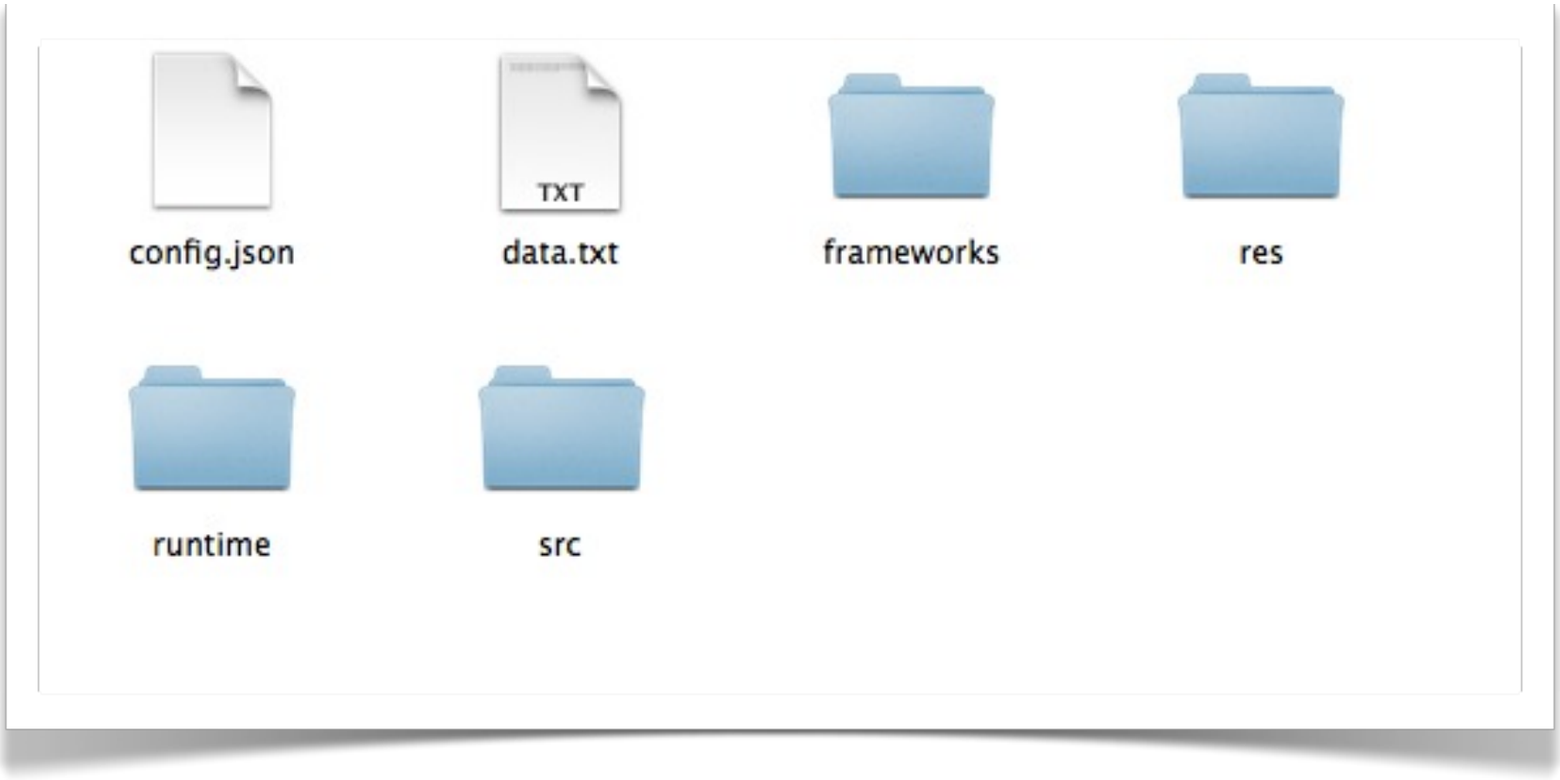
图1-2 输入内容

新建项目

首先启动 Quick 下的 player3，在这儿的示例标签下你可以看到很多Quick自带的示例，。其界面如下图所示：



项目创建成功之后，打开我们创建项目的根目录，目录结构如下图所示：



config.json:项目信息配置文件。
debug.log:项目日志，即打印控制台窗口输出的所有日志文件。
frameworks:存放Cocos2d-x引擎核心代码及各个平台运行时资源。
res:存放项目资源的文件夹，也就是说，我们游戏开发中用到的所有图片、字体、音频等资源都放在这里。
runtime: 存放预编译的运行时库。
src:项目源码所存放文件夹，即游戏中的所有的 .lua 文件都放在这里。
以上目录中 res 和 src 文件夹是比较最要的，开发中我们也只需要对这两个文件夹里的内容进行操作，就可以实现游戏的开发。

在新项目的src文件夹中，现在你是可以看到一些 .lua 文件的，这些就是我们工程的lua代码。接下来我们简单的介绍下src中各项的功能：

cocos：cocos引擎代码
framework：quick的核心部分，在Cocos2d-x基础上自己搭建的一套framework
config.lua：工程配置文件，包括分辨率适配等信息
main.lua：工程入口
app：工程的界面等文件，存放我们的游戏代码
MyApp.lua：游戏的第一个界面
scenes：存放游戏各个场景代码的文件夹
MainScene：游戏的第一个场景

在游戏开发中，需要修改和添加界面时，我们只需要在相应的文件夹中添加场景就可以了。

提示：我们将项目里面的src目录直接拖动到SublimeText2之中，就可以进行代码的编辑，节省大量时间。

config.lua介绍

下面是它现有属性的具体含义：

DEBUG：配置Quick工程的调试信息状态，0表示关闭，1表示打印少量调试信息，2表示打印标准调试信息。
DEBUG_FPS：是否显示模拟器左下角的FPS信息
DEBUG_MEM：是否要每10秒打印一次内存信息
LOAD_DEPRECATED_API：是否加载已经废弃了的API
LOAD_SHORTCODES_API：是否加载短代码
CONFIG_SCREEN_ORIENTATION：游戏中的屏幕方向，landscape横屏，portrait竖屏。

CONFIG_SCREEN_WIDTH: 屏幕宽度，但屏幕方向为“landscape”横屏时，该属性表示屏幕的高度

CONFIG_SCREEN_HEIGHT: 屏幕高度，但屏幕方向为“landscape”横屏时，该属性表示屏幕的宽度

CONFIG_SCREEN_AUTOSCALE: 屏幕适配策略，如FIXED_WIDTH、FIXED_HEIGHT和FILL_ALL

main.lua

在src目录下，除了 config.lua 文件外，还有一个 main.lua 文件，这个 main.lua 是 Quick 项目的通用入口文件，它类似于 Cocos2d-x 中的 AppDelegate.h/cpp 文件，同时也类似于一般 Windows 工程中的 main 文件。

打开 main.lua 文件，其内容如下所示：

```
function __G__TRACKBACK__(errorMessage)
    print("----------------------------------------")
    print("LUA ERROR: " .. tostring(errorMessage) .. "\n")
    print(debug.traceback("", 2))
    print("----------------------------------------")
end
package.path = package.path .. ";src/"
cc.FileUtils:getInstance():setPopupNotify(false)
require("app.MyApp").new():run()
```

每个新项目的 main.lua 文件内容都是一样的，一般情况下，我们不需要改动它。

我们只需要知道，main 文件的最后一行代码中通过载入的 app.MyApp 模块创建了一个 MyApp 实例，并且还调用执行了该 MyApp 实例的 run 方法。这行代码将启动并执行 MyApp 脚本。从此处我们也可以看出，main.lua 文件是应用程序 lua 脚本的启动文件。

require 方法表示引入一个文件，使用 require 方法加载文件的过程会检查文件的 lua 语法，同时会完成被加载文件内部变量的初始化。

二：Quick基础知识

2.1 class介绍

使用 class 函数定义一个 MyApp 类。

class 方法本身是 Quick 框架中定义的一个用于创建自定义 lua 类的函数，该函数可在 Quick 引擎的 “quick/framework/functions.lua”文件中找到。下面我们看一下具体使用方法。

```
-- 定义名为 Shape 的基础类
local Shape = class("Shape")
-- ctor() 是类的构造函数，在调用 Shape.new() 创建 Shape 对象实例时会自动执行
function Shape:ctor(shapeName)
    self.shapeName = shapeName
    printf("Shape:ctor(%s)", self.shapeName)
end
-- 为 Shape 定义个名为 draw() 的方法
function Shape:draw()
    printf("draw %s", self.shapeName)
end
-- Circle 是 Shape 的继承类
local Circle = class("Circle", Shape)
function Circle:ctor()
    -- 如果继承类覆盖了 ctor() 构造函数，那么必须手动调用父类构造函数
    -- 类名.super 可以访问指定类的父类
    Circle.super.ctor(self, "circle")
    self.radius = 100
end
```

```
function Circle:setRadius(radius)
    self.radius = radius
end

-- 覆盖父类的同名方法
function Circle:draw()
    printf("draw %s, raidus = %0.2f", self.shapeName, self.raidus)
end

--

local Rectangle = class("Rectangle", Shape)

function Rectangle:ctor()
    Rectangle.super.ctor(self, "rectangle")
end

local circle = Circle.new()                 -- 输出: Shape:ctor(circle)
circle:setRaidus(200)
circle:draw()                               -- 输出: draw circle, radius = 200.00

local rectangle = Rectangle.new()           -- 输出: Shape:ctor(rectangle)
rectangle:draw()
```

2.2 基础类的创建

精灵等元素创建

我们在ctor函数里面写如下代码

```
local sp=cc.Sprite:create("a.png")
sp:pos(200, 200)
sp:setScale(0.5)
self:addChild(sp)
display.newSprite("b.png")
:pos(300, 300)
:addTo(self)
```

创建精灵有多种方式，可以直接调用API创建，也可以使用quick封装的display.newSprite() 来进行创建。两者表现不同，本质一样。

```
local png = "hero/minerAction.png"
local plist = "hero/minerAction.plist"
display.addSpriteFrames(plist, png);
self._sp = display.newSprite("#miner_0701.png"):pos(200, 200)
self:addChild(self._sp, 0)
local frames = display.newFrames("miner_0%d.png", 701, 10);
local animate = display.newAnimation(frames, 0.08);
self._sp:playAnimationForever(animate)
```

图2-1 精灵播放动画

2.3 动作的使用

正常播放一个动作，如下代码所示：

```
local sp=cc.Sprite:create("a.png")
sp:pos(200, 200)
sp:setScale(0.5)
self:addChild(sp)
local move=cc.MoveTo:create(3,cc.p(400,400))
sp:runAction(move)
```

为了使用更加方便，quick封装了一些方法用来实现动作（transition），我们看一下常见的一些函数。

```
-- 耗时 0.5 秒将 sprite 旋转到 180 度
transition.rotateTo(sprite, {rotate = 180, time = 0.5})
```

```
    -- 等待 1.0 后开始移动对象
    -- 耗时 1.5 秒，将对象移动到屏幕中央
    -- 移动使用 backout 缓动效果
    -- 移动结束后执行函数，显示 move completed
    transition.execute(sp, cc.MoveTo:create(1.5, cc.p(display.cx,
     display.cy)),
    {
     delay = 1.0,
     easing = "backout",
     onComplete = function()
         print("move completed")
     end,
    })
    -- 移动到屏幕中心
    transition.moveTo(sprite, {x = display.cx, y = display.cy, time = 1.5})
    -- 移动到屏幕左边，不改变 y
    transition.moveTo(sprite, {x = display.left, time = 1.5})
    -- 移动到屏幕底部，不改变 x
    transition.moveTo(sprite, {y = display.bottom, time = 1.5})
    -- 不管显示对象当前的透明度是多少，最终设置为 128
    transition.fadeTo(sprite, {opacity = 128, time = 1.5})
    --创建一个动作序列对象。
    local sequence = transition.sequence({
        cc.MoveTo:create(0.5, cc.p(display.cx, display.cy)),
        cc.FadeOut:create(0.2),
        cc.DelayTime:create(0.5),
        cc.FadeIn:create(0.3),
    })
    sprite:runAction(sequence)
```

```
--在显示对象上播放一次动画，并返回 Action 动作对象。
local frames = display.newFrames("Walk%04d.png", 1, 20)
local animation = display.newAnimation(frames, 0.5 / 20) -- 0.5s play 20 frames
transition.playAnimationOnce(sprite, animation)

--还可以用 Sprite 对象的 playAnimationOnce() 方法来直接播放动画:
local frames = display.newFrames("Walk%04d.png", 1, 20)
local animation = display.newAnimation(frames, 0.5 / 20) -- 0.5s play 20 frames
sprite:playAnimationOnce(animation)
```

2.4 触摸处理

Quick里面对触摸的处理发生了一些变化，代码如下所示：

```
function MainScene.testTouch(self)
    local sp=cc.Sprite:create("BlueBlock.png")
    self:addChild(sp)
    sp:setTouchEnabled(true)
    sp:setTouchSwallowEnabled(false)
    sp:addNodeEventListener(cc.NODE_TOUCH_EVENT,
      function(event)
    -- dump(event)
    if event.name == "ended" then
        print("ended.....")
    elseif event.name =="began" then
     print("began.....")
     return true
    end
  end)
end
```

2.5 时间调度

在游戏里面需要大量使用时间调度，使用方式之一如下：

```
local scheduID;
sharedScheduler = cc.Director:getInstance():getScheduler();
scheduID = sharedScheduler:scheduleScriptFunc(function ()
        print("schedule")
        cc.Director:getInstance()
            :getScheduler():unscheduleScriptEntry(scheduID)
    end, 1, false)--时间可以是0，意思是每帧都调用
```

在游戏里面需要大量使用时间调度，使用方式之二如下：

```
local function onInterval(dt)
    dump(dt)
    print("aaaa")
end
-- 每 0.5 秒执行一次 onInterval()
local handle = scheduler.scheduleGlobal(onInterval, 0.5)
scheduler.unscheduleGlobal(handle)
```

当我们使用第二种方式的时候，必须导入需要的模块，该模块在框架初始化时不会自动载入。如下代码所示：

```
local scheduler = require(cc.PACKAGE_NAME .. ".scheduler")
```

2.6 音频处理

我们在游戏中，大量使用到了音乐和音效，我们下面看一下如何在Quick里面添加音乐和音效。

```
-- 返回音乐的音量值
audio.getMusicVolume()
-- 设置音乐的音量
audio.setMusicVolume(volume)
-- 预载入一个音乐文件
audio.preloadMusic(filename)
-- @param string filename 音乐文件名
-- @param boolean isLoop 是否循环播放，默认为 true
audio.playMusic(filename, isLoop)

-- 播放音效，并返回音效句柄
-- 如果音效尚未载入，则会载入后开始播放。
-- 该方法返回的音效句柄用于 audio.stopSound()、audio.pauseSound()
等方法。
audio.playSound(filename, isLoop)
```

audio在框架初始化的时候会进行导入，不需要我们手动导入。

2.7 display介绍及使用

display 模块封装了绝大部分与显示有关的功能，并负责根据 config.lua 中定义的分辨率设定计算屏幕的设计分辨率。如下图所示的许多定义都包含在里面。

```
框架初始化后，display 模块提供下列属性：
-   display.sizeInPixels.width,
-   display.sizeInPixels.height 屏幕的像素分辨率
-   display.widthInPixels,
-   display.heightInPixels 屏幕的像素分辨率
-   display.contentScaleFactor 内容缩放因子
-   display.size.width,
-   display.size.height 屏幕的设计分辨率
-   display.width,
-   display.height 屏幕的设计分辨率
-   display.cx,
-   display.cy 屏幕中央的 x 坐标和 y 坐标
-   display.left,
-   display.top,
-   display.right,
-   display.bottom 屏幕四边的坐标
-   display.c_left,
-   display.c_top,
-   display.c_right,
-   display.c_bottom 当父对象在屏幕中央时，屏幕四边的坐标
```

```
-- 创建一个新场景
local nextScene = display.newScene("NextScene")
-- 包装过渡效果
local transition = display.wrapSceneWithTransition(nextScene, "fade",
0.5)
-- 切换到新场景
display.replaceScene(transition)
```

```
local group = display.newNode()          -- 创建一个容器
group:addChild(sprite1)                  -- 添加显示对象到容器中
group:addChild(sprite2)                  -- 添加显示对象到容器中

-- 移动容器时，其中包含的子对象也会同时移动
transition.moveBy(group, {time = 2.0, x = 100})
```

创建并返回一个 Sprite 显示对象

display.newSprite() 有三种方式创建显示对象：

- 从图片文件创建
- 从缓存的图像帧创建
- 从 SpriteFrame 对象创建

```
-- 从图片文件创建显示对象
local sprite1 = display.newSprite("hello1.png")
-- 从缓存的图像帧创建显示对象
-- 图像帧的名字就是图片文件名，但为了和图片文件名区分，所以此处需要在文件名前添加 “#” 字符
-- 添加 “#” 的规则适用于所有需要区分图像和图像帧的地方
local sprite2 = display.newSprite("#frame0001.png")
-- 从 SpriteFrame 对象创建
local frame = display.newFrame("frame0002.png")
local sprite3 = display.newSprite(frame)
```

三：UI控件使用

3.1 UISlider滑块的使用

滑块是我们游戏中常用的元素之一，我们在此通过代码介绍滑块的使用方法。

```
local barHeight = 40
local barWidth = 400
cc.ui.UISlider.new(display.LEFT_TO_RIGHT, {
bar = "SliderBar.png",
button = "SliderButton.png"}, {scale9 = true})
    :onSliderValueChanged(function(event)
        print(string.format("value = %0.2f", event.value))
    end)
    :setSliderSize(barWidth, barHeight)
    :setSliderValue(75)
    :align(display.LEFT_BOTTOM, display.left + 40, display.top - 80)
    :addTo(self)
```

滑块的使用方式比较简单，主要用于血条，游戏进度等内容的控制显示。

3.2 UIPushButton的使用

我们在游戏里面大量使用到了按钮操作，这里我们讲解一下UIPushButton的使用。

```
local btn1=cc.ui.UIPushButton.new({normal="leftBut1.png",
    pressed="leftBut2.png"},{scale9 = true})
btn1:onButtonClicked(function(event)
      print("onButtonClicked")
    end)
btn1:onButtonPressed(function (event)
    print("onButtonPressed")
end)
btn1:setButtonSize(240, 60)
btn1:pos(200,160)
self:addChild(btn1)
--UILabel使用
    cc.ui.UILabel.new({text = "原始图", size = 24, color =
display.COLOR_BLACK})
        :align(display.CENTER, 240, 480)
        :addTo(self)
```

3.3 UIInput使用

文本框是我们游戏中常用的内容，这里我们讲解一下UIInput的使用，在之前的版本中，名称是EditBox，现在统一替换为UIInput，下面我们看一下具体的使用方法。

```
    -- 输入事件监听方法
local function onEdit(event, editbox)
    if event == "began" then
    -- 开始输入
        print("开始输入")
    elseif event == "changed" then
    -- 输入框内容发生变化
        print("输入框内容发生变化")
        local text = editbox:getText()
        print(text)
    elseif event == "ended" then
    -- 输入结束
        print("输入结束")
    elseif event == "return" then
    -- 从输入框返回
        print("从输入框返回")
    end
end

local a = cc.ui.UIInput.new({
    image = "GreenBlock.png", -- 输入控件的背景
    listener = onEdit, -- 绑定输入监听事件处理方法
    x = 200,
    y = 300,
    size = cc.size(200, 40)
})
a:addTo(self)
a:setInputFlag(0)
a:setPlaceHolder("请输入密码")
```

可用参数：

- image：输入框的图像，可以是图像名或者是 Sprite9Scale 显示对象。用 display.newScale9Sprite() 创建 Sprite9Scale 显示对象。
- imagePressed：输入状态时输入框显示的图像（可选）
- imageDisabled：禁止状态时输入框显示的图像（可选）
- listener：回调函数
- size：输入框的尺寸，用 cc.size(宽度，高度) 创建

3.4 UICheckBox使用

UICheckBox也就是复选框，经常用在设置一些音乐音效的开关，以及游戏其他需要不同状态控制的场所。下面我们看一下具体的使用方法。

```
local function updateCheckBoxButtonLabel(checkbox)
        local state=""
        if checkbox:isButtonSelected() then
            state="on"
        else
            state="off"
        end

        if not checkbox:isButtonEnabled() then
            state = state .. " (disabled)"
        end
        checkbox:setButtonLabelString(string.format("state is %s", state))
    end
    local  checkboxButton=cc.ui.UICheckBoxButton.new(
        MainScene.CHECKBOX_BUTTON_IMAGES)
    :setButtonLabel(cc.ui.UILabel.new({text = "", size = 22,  color = cc.c3b(255,
96, 255)}))
    :setButtonLabelOffset(0, -40)
    :setButtonLabelAlignment(display.CENTER)
    :onButtonStateChanged(function(event)
            updateCheckBoxButtonLabel(event.target)
        end)
    :align(display.LEFT_CENTER, display.left + 40, display.top - 80)
    :addTo(self)
```

3.4 UICheckBoxGroup使用

UICheckBoxGroup里面可以存放多个UICheckBoxButton，并且可以动态的删除对应下标的元素，下面我们看具体用法。

```
local group = cc.ui.UICheckBoxButtonGroup.new(display.TOP_TO_BOTTOM)
        :addButton(cc.ui.UICheckBoxButton.new(MainScene.RADIO_BUTTON_IMAGES)
            :setButtonLabel(cc.ui.UILabel.new({text = "option 1", color = cc.c3b(255, 0, 0)}))
            :setButtonLabelOffset(20, 0)
            :align(display.LEFT_CENTER))
        :addButton(cc.ui.UICheckBoxButton.new(MainScene.RADIO_BUTTON_IMAGES)
            :setButtonLabel(cc.ui.UILabel.new({text = "option 2", color = cc.c3b(255, 0, 0)}))
            :setButtonLabelOffset(20, 0)
            :align(display.LEFT_CENTER))
        :addButton(cc.ui.UICheckBoxButton.new(MainScene.RADIO_BUTTON_IMAGES)
            :setButtonLabel(cc.ui.UILabel.new({text = "option 3", color = cc.c3b(255, 0, 0)}))
            :setButtonLabelOffset(20, 0)
            :align(display.LEFT_CENTER))
        :addButton(cc.ui.UICheckBoxButton.new(MainScene.RADIO_BUTTON_IMAGES)
            :setButtonLabel(cc.ui.UILabel.new({text = "option 4 disabled", color = cc.c3b(255, 0, 0)}))
            :setButtonEnabled(false)
            :setButtonLabelOffset(20, 0)
            :align(display.LEFT_CENTER))
        :setButtonsLayoutMargin(10, 10, 10, 10)
        :onButtonSelectChanged(function(event)
            printf("Option %d selected, Option %d unselected", event.selected, event.last)
        end)
        :align(display.LEFT_TOP, display.left + 40, display.top - 240)
        :addTo(self)
        group:getButtonAtIndex(1):setButtonSelected(true)
```

```
    cc.ui.UIPushButton.new("GreenButton.png", {scale9 = true})
        :setButtonSize(160, 40)
        :setButtonLabel(cc.ui.UILabel.new({text = "Remove option 2", size = 16, color =
display.COLOR_BLUE}))
        :onButtonPressed(function(event)
            event.target:getButtonLabel():setColor(display.COLOR_RED)
        end)
        :onButtonRelease(function(event)
            event.target:getButtonLabel():setColor(display.COLOR_BLUE)
        end)
        :onButtonClicked(function(event)
            if group:getButtonsCount() == 4 then
                group:removeButtonAtIndex(2)
                event.target:removeSelf()
            end
        end)
        :align(display.LEFT_CENTER, display.left + 200, display.top - 210)
        :addTo(self)
```

运行效果如下图所示：

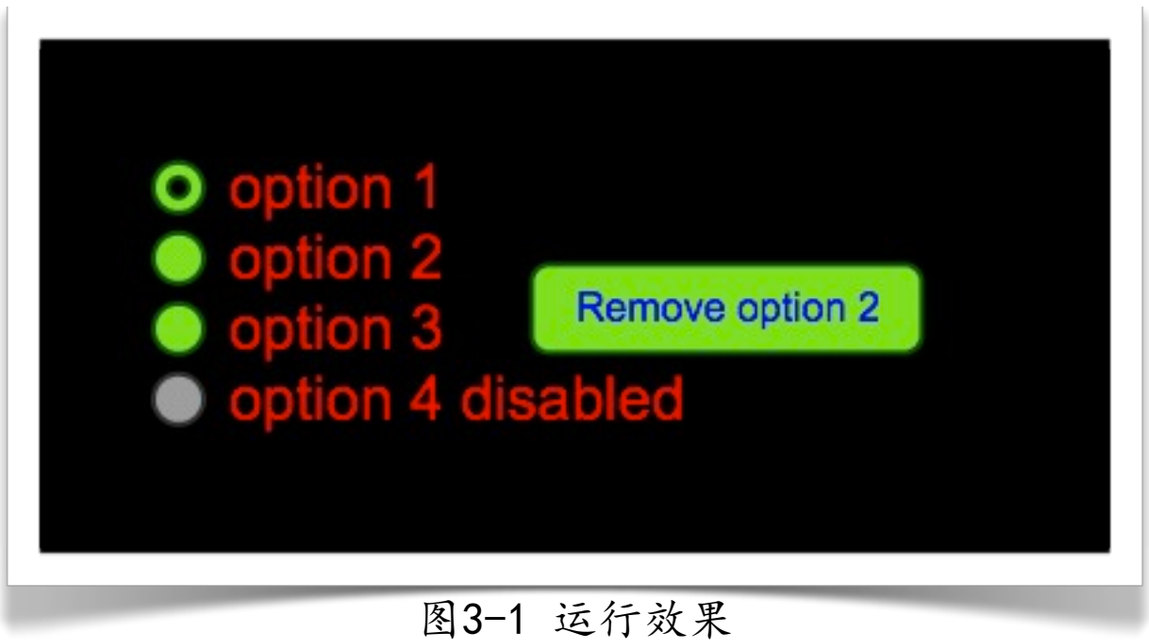


图3-1 运行效果

3.5 UIScrollView使用

```
local sp2 = display.newScale9Sprite("sunset.png")
sp2:setContentSize(300, 200)
sp2:pos(720, 460)

local emptyNode = cc.Node:create()
emptyNode:addChild(sp2)

local bound = sp2:getBoundingBox()
bound.width = 300
bound.height = 200

cc.ui.UIScrollView.new({viewRect = bound,
    scrollbarImgH = "SliderBarFixed.png",
    scrollbarImgV = "SliderBarFixedV.png",
    bgColor = cc.c4b(255,0,0,255)})
    :addScrollNode(emptyNode)
    :setDirection(cc.ui.UIScrollView.DIRECTION_HORIZONTAL)
    :onScroll(function (event)
    print("ScrollListener:" .. event.name)
end)
    :addTo(self)
    :setBounceable(true) -- 是否有回弹效果(默认支持)
```

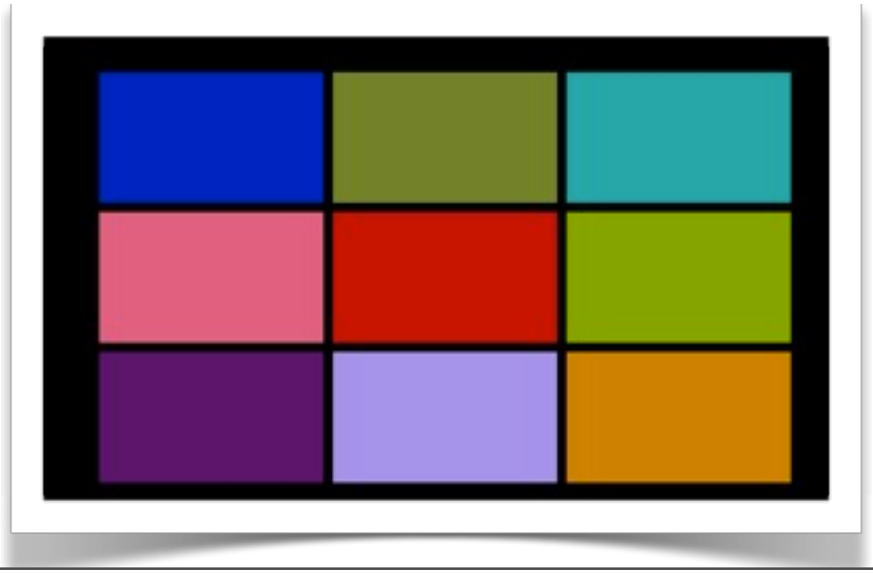
3.6 UIPageView

分页视图控件的内容是视窗区域的1到n倍，一个视窗的内容叫一页。创建分页视图后，默认显示第一页，用户可以左右滑动切换显示其它页。当用户松开触摸后， 控件自动判断当前应该显示哪一页，然后自动滚动这页使它居中显示在视窗中。分页视图非常适合用于做游戏的帮助引导界面，玩家可以左右滑动切换一页一页的查 看帮助引导。

Quick 封装的分页视图控件类是 UIPageView，源码位于framework/cc/ui/UIPageView.lua。同时能添加到 UIPageView 的项被抽象为 UIPageViewItem，源码位于framework/cc/ui/UIPageViewItem.lua。

在Quick中通过cc.ui.UIPageView获取UIPageView，然后可以通过new方法创建 UIPageView 控件每一页的显示内容由一个矩阵组成，矩阵的每个元素是 UIPageViewItem。可以指定每页显示几行几列，行间距，列间距等。在创建完成后可以设置 touch 事件监听函数。

UIPageViewItem 是个容器，它承载显示内容，然后再把它添加到 UIPageView 中。假如我们需要一个2页的分页视图控件，每页显示 3x3 的矩阵，那么你总共需要 2 * (3 * 3) = 18 个 UIPageViewItem。示例运行效果如下所示：



```
self.pv = cc.ui.UIPageView.new {
        -- bgColor = cc.c4b(200, 200, 200, 120),
        -- bg = "sunset.png",
        viewRect = cc.rect(80, 80, 780, 480),
        column = 3, row = 3,
        padding = {left = 20, right = 20, top = 20, bottom = 20},
        columnSpace = 10, rowSpace = 10}
        :onTouch(touchListener)
        :addTo(self)
    -- add items
    for i=1,18 do
        local item = self.pv:newItem()
        local content
        content = cc.LayerColor:create(
            cc.c4b(math.random(250),
                math.random(250),
                math.random(250),
                250))
        content:setContentSize(240, 140)
        content:setTouchEnabled(false)
        item:addChild(content)
        self.pv:addItem(item)
    end
    self.pv:reload()
```

3.7 UIListView

列表控件用于显示一串或一列具有共性的项。如游戏中的游戏道具展示，就需要列表视图控件。

Quick 封装的列表视图控件类是 UIListView，源码位于framework/cc/ui/UIListView.lua。同时能添加到 UIListView 的项被抽象为 UIListViewItem，源码位于framework/cc/ui/UIListViewItem.lua。

UIListView 是 UIScrollView 的扩展，与UIScrollView不同，它支持添加多个元素。

创建列表视图首先需要建立列表，然后再向列表里面添加项。

通过cc.ui.UIListView获取 UIListView，然后调用 new 方法创建一个列表。

```
cc.ui.UIListView.new(params)
```

创建列表项

你可以直接使用 UIListViewItem 的 new 方法来创建一个表项，但这不是推荐的方法。UIListView 内部封装了一个创建表项的成员函数，它做针对当前表的属性在 item 上做了额外的初始化。你应该使用下面的方式来创建表项。

```
local list = cc.ui.UIListView:new(params) -- 新建表
local item = list:newItem(contentNode) -- 新建表项
```

其中参数 contentNode 是一个可显示的节点类型，将被用来显示当前项的内容。也可以不传 contentNode，然后通过item:addContent(contentNode)来设置显示节点。

UIListViewItem 是为 UIListView 服务的，除了 contentNode，它还有一个重要的功能是约束当前列表项的宽高。它提供下面的接口：

1. UIListViewItem:setItemSize(w, h, bNoMargin) 默认情况下，可以用边距来控制 contentNode 在 UIListViewItem 上的布局。bNoMargin 设为 true，则忽略边距参数。
2. UIListViewItem:setMargin(margin) margin 是一个table类型。UIListViewItem 的默认 margin 为{left = 0, right = 0, top = 0, bottom = 0}。通过这个函数来改变 margin。
3. 注意：需要在setItemSize之前调用setMargin。

添加/移除列表项

- 添加列表项

UIListView 内部封装了添加列表项的方法：

UIListView:addItem(listItem, pos)
其中参数listItem为待添加的列表项，pos指定要添加的位置,默认添加到最后。

- 移除列表项

对应的也封装了移除列表项的方法：

UIListView:removeItem(listItem, bAni)
其中参数listItem为待移除的列表项，bAni指定是否播放移除动画。

UIListView中还提供了移除所有列表项的方法,调用removeAllItems()移除所有列表项。

事件

Quick 为UIListView封装了简单易用的事件处理方法。可以通过如下方式监听UIListView的Touch事件。

listView:onTouch(function (event) end)
可能会返回以下各种事件类型:

- "clicked" 点击事件，点击列表时触发。可以获取具体的点击点坐标x,y，被点击的列表项，位置等
- "began" 列表开始滚动事件
- "moved" 列表滚动事件
- "ended" 列表滚动结束事件
- "itemAppear" 列表项显示出现事件，列表滚动中某个项出现时触发。
- "itemAppearChange" 列表项的显示改变，如滚动、大小改变等
- "itemDisappear" 列表项显示消失事件， 列表滚动中某个项不可见时触发。

可以通过event.name判断返回的事件类型，然后进行相应的事件处理。完整代码如下所示:

```
--创建ListView
    self.lv=cc.ui.UIListView.new{
    -- bgColor = cc.c4b(200, 200, 200, 120),
    bg="sunset.png",
    bgScale9=true,
    viewRect=cc.rect(40, 80, 120, 400),
    direction=cc.ui.UIScrollView.DIRECTION_VERTICAL,
    scrollbarImgV = "bar.png"}
    :onTouch(handler(self, self.touchListener))
    :addTo(self)
```

```
--add items
     for i=1,20 do
          local item=self.lv:newItem()
          local content
     if 1==i then
          content=cc.ui.UILabel.new({text="item"..i,size=20,align=cc.ui.TEXT_ALIGN_CENTER,
               color=display.COLOR_RED})
     elseif 2==i then
               content=cc.ui.UIPushButton.new("GreenButton.png",{scale9 = true})
               :setButtonSize(120, 40)
               :setButtonLabel(cc.ui.UILabel.new({text = "点击大小改变" .. i,
                    size = 16, color = display.COLOR_BLUE}))
               :onButtonPressed(function(event)
                    event.target:getButtonLabel():setColor(display.COLOR_RED)
               end)
          :onButtonClicked(function(event)
                if self.bListViewMove then
                    print("TestUIListViewScene is scroll, not click")
                   return
                end
                print("TestUIListViewScene buttonclicked")
                local _,h=item:getItemSize()
                if 40 == h then
                       item:setItemSize(120, 80)
                 else
                       item:setItemSize(120, 40)
                 end
               end)
     content:setTouchSwallowEnabled(false)
```

```
    elseif 3==i then
        content = cc.ui.UILabel.new(
                    {text = "点击删除它"..i,
                    size = 20,
                    align = cc.ui.TEXT_ALIGN_CENTER,
                    color = display.COLOR_RED})
    elseif 4==i then
        content = cc.ui.UILabel.new(
                    {text = "有背景图"..i,
                    size = 20,
                    align = cc.ui.TEXT_ALIGN_CENTER,
                    color = display.COLOR_RED})
          item:setBg("YellowBlock.png")
  else
      content = cc.ui.UILabel.new(
                    {text = "item"..i,
                    size = 20,
                    align = cc.ui.TEXT_ALIGN_CENTER,
                    color = display.COLOR_BLACK})
     end
     item:addContent(content)
     item:setItemSize(120, 40)
     self.lv:addItem(item)
end
   self.lv:reload()
```

```
function TestListScene:touchListener(event)
    dump(event, "event:")
    local listView=event.listView
    if "clicked"==event.name then
        if 3==event.itemPos then
            listView:removeItem(event.item, true)
        end
    elseif "moved"==event.name then
        self.bListViewMove = true
    elseif "ended"==event.name then
        self.bListViewMove = false
    else
        print("event name:" .. event.name)
    end
end
```