第 3 章 Hello Cocos2d-JS

在开始详细介绍 Cocos2d-JS 引擎的 API 之前，有必要先了解一下手机游戏引擎有哪些，了解 Cocos2d-JS 的前世今生。还会介绍开发工具。然后从一个 HelloJS 入手，介绍 Cocos2d-JS 的基本开发流程，以及 Cocos2d-JS 生命周期和 Cocos2d-JS 核心知识体系。

3.1 移动平台游戏引擎

游戏引擎是指一些已编写好的游戏程序模块。游戏引擎包含以下子系统：渲染引擎（即“渲染器”，含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。

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 Unity3D、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 Unity3D。

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-JS 之前有必要先介绍一下 Cocos2d 的家谱，图 3-1 所示是 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

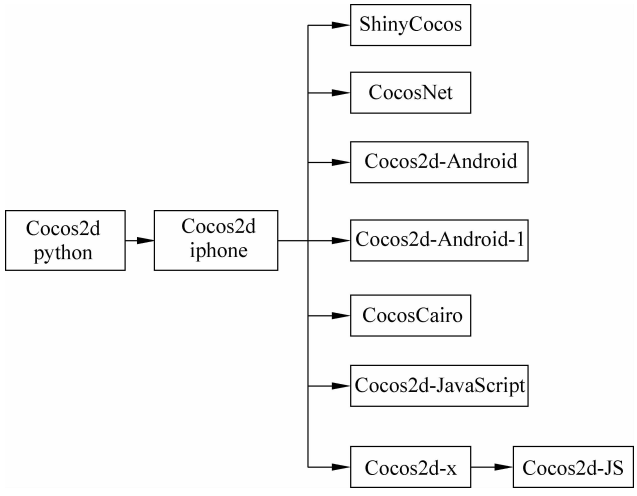


图 3-1　Cocos2d 的家谱

多元化。其中各个引擎介绍如下：

（1）ShinyCocos：使用 Ruby 对 Cocos2d-iphone 进行封装，使用 Ruby api 开发。

（2）CocosNet：是在 MonoTouch 平台上使用的 Cocos2d 引擎，采用.NET 实现。

（3）Cocos2d-android：是为 Android 平台使用的 Cocos2d 引擎，采用 Java 实现。

（4）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。

（5）Cocos2d-javascript：是采用 JavaScript 脚本语言实现的 Cocos2d 引擎。

（6）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。

（7）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；另一方面它依托浏览器运行，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。

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、WinPhone、Bada、BlackBerry 和 MeeGo 等。纵向方面向下能够支持 OpenGL ES 1.1、OpenGL ES 1.5、OpenGL ES 2.0 和 DirectX 11 等技术，向上支持 JavaScript 和 Lua 脚本绑定。

简单地说，Cocos2d-x 设计目标是为了实现跨平台，用户不再为同一款游戏在不同平台发布而进行编译。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，

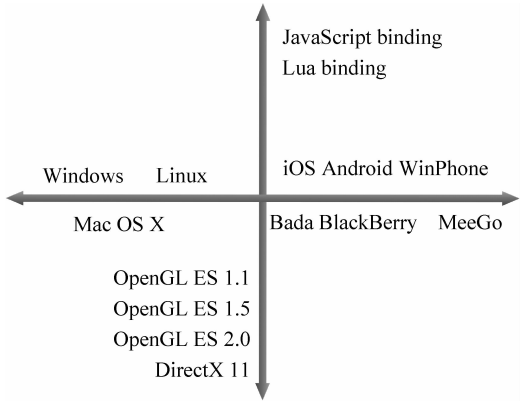


图 3-2 Cocos2d-x 设计目标

针对这种情况可以使用 JavaScript 和 Lua[①] 开发游戏。

3.2.3 Cocos2d-JS 引擎

Cocos2d-JS 设计得非常巧妙，使用的语言是 JavaScript，容易上手。基于 Cocos2d-JS 引擎开发的游戏程序，一方面是通过 Cocos2d-html5 引擎在 Web 浏览器上运行，另一方面是通过 JSB(JavaScript binding)技术通过 Cocos2d-x 引擎在本地运行。Cocos2d-JS 运行原理如图 3-3 所示。

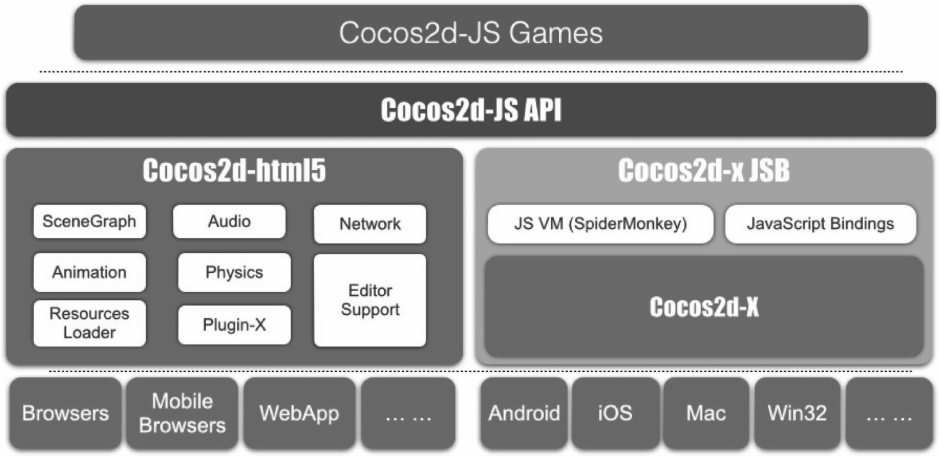


图 3-3 Cocos2d-JS 运行原理

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，由 Roberto Ierusalimschy、Waldemar Celes 和 Luiz Henrique de Figueiredo 所组成并于 1993 年开发。——引自于百度百科(http://baike.baidu.com/view/416116.htm?fr=wordsearch)。

Cocos2d-JS 与 Cocos2d-x 相比更先进，不仅可以在本地运行，还可以在 Web 浏览器上运行。

3.3 搭建 Cocos2d-JS 开发环境

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

3.3.1 搭建 WebStorm 开发环境

在上一章使用了 WebStorm 开发工具，它是非常优秀的 JavaScript 开发工具，WebStorm 工具可以开发和调试基于 Cocos2d-JS 引擎的 JavaScript 程序代码，但是测试和调试时只能运行在 Web 浏览器上。

WebStorm 安装过程在上一章已经介绍了，但是要想开发基于 Cocos2d-JS 引擎的 JavaScript 程序，还需要安装 Google Chrome 浏览器和 JetBrains IDE Support 插件。Google Chrome 浏览器安装不再介绍，这里重点介绍 JetBrains IDE Support 插件。

JetBrains IDE Support 是安装在 Google Chrome 浏览器上的插件，它是为了配合 WebStorm 工具调试使用的。JetBrains IDE Support 插件安装过程是在 Google Chrome 浏览器的网址中输入 https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomchmhgeddb 内容，安装页面如图 3-4 所示。该页面中可以单击“已添加至 CHROME”按钮安装插件。

安装成功后会在浏览器的地址栏后面出现 JB 图标，具体如何使用在后面章节再介绍。

3.3.2 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，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 下载和安装

图 3-5 所示是 JDK 7 下载界面，它的下载地址是 http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html，其中有很多版本。注意选择对应的操作系统，以及 32 位还是 64 位安装的文件。

---

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

图 3-4 安装 JetBrains IDE Support 插件

Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.04 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.38 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.92 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

图 3-5 下载 JDK

下载完 JDK 并安装完成后需要设置系统环境变量，主要是设置 JAVA_HOME 环境变量。打开环境变量设置对话框，如图 3-6 所示，可以在用户变量（上半部分，只影响当前用户）或系统变量（下半部分，影响所有用户）添加环境变量。一般情况下，在用户变量中设置环境变量。

图 3-6　环境变量设置对话框

在用户变量部分单击“新建”按钮，弹出对话框如图 3-7 所示，设置“变量名”为 JAVA_HOME，变量值为“C:\Program Files\Java\jdk1.7.0_21”。注意变量值的路径。

图 3-7　设置 JAVA_HOME

为了防止安装了多个 JDK 版本对于环境的影响，还可以在环境变量 PATH 追加 C:\Program Files\Java\jdk1.7.0_21\bin 路径，如图 3-8 所示，在用户变量中找到 PATH。双击打开 PATH 修改对话框，如图 3-9 所示，追加 C:\Program Files\Java\jdk1.7.0_21\bin。注意 PATH 之间用分号分隔。

图 3-8 环境变量 PATH 设置对话框

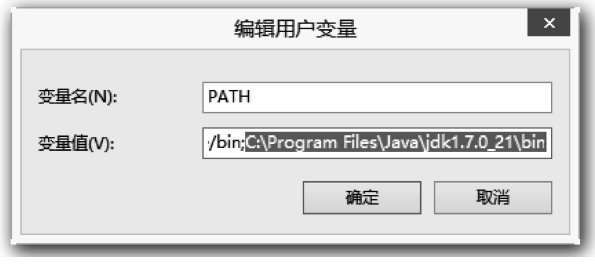


图 3-9 PATH 修改对话框

2. Cocos Code IDE 下载和安装

Cocos Code IDE 下载地址是 http://www.cocos2d-x.org/download，在浏览器中页面如图 3-10 所示。选择合适的文件下载，目前包括 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。

这里下载的是 cocos-code-ide-win64-1.0.0-rc1.zip 解压版本，解压之后找到 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 3-11 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，则创建。

Cocos Code IDE 具体如何使用在后面章节再介绍。

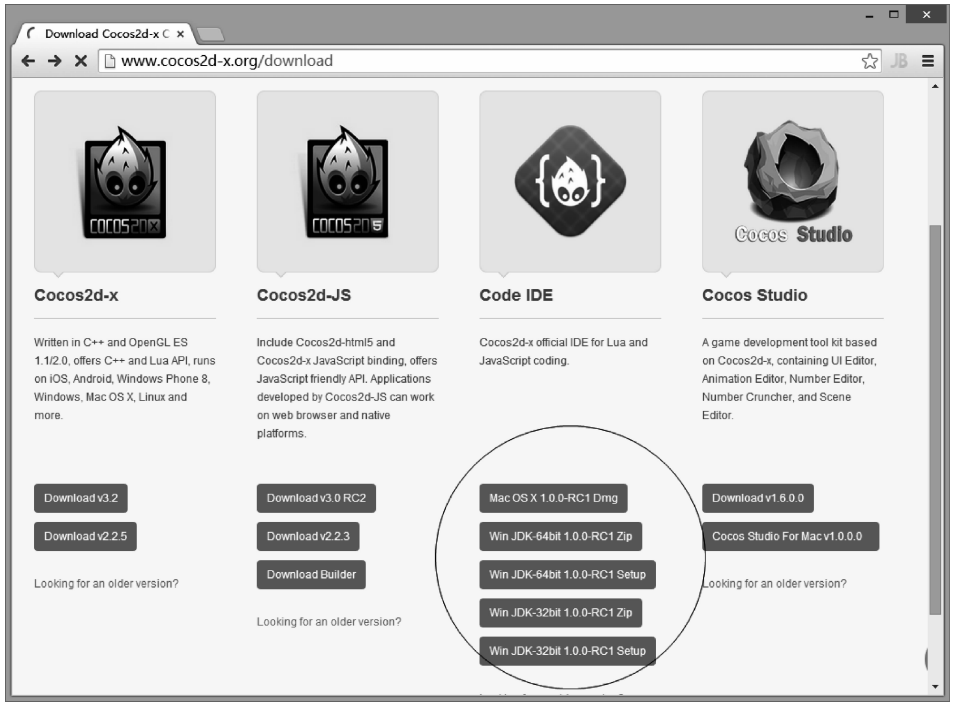


图 3-10　下载 Cocos Code IDE

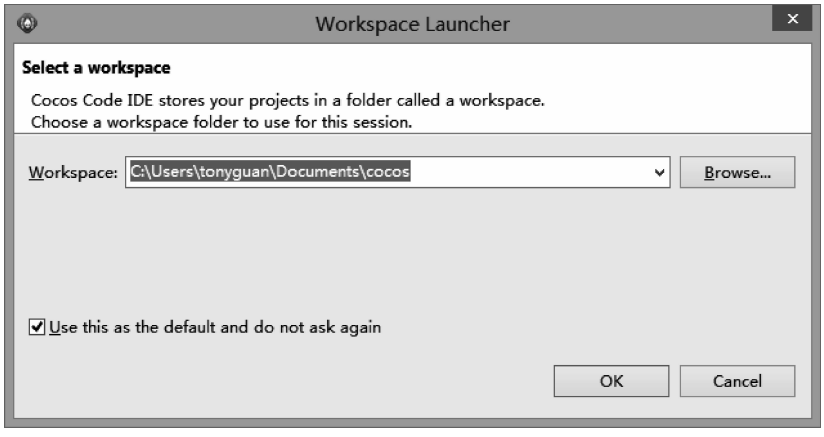


图 3-11　选择 Workspace

3.3.3　下载和使用 Cocos2d-JS 官方案例

首先到 Cocos2d-JS 官方网站下载 Cocos2d-JS 开发包，目前 Cocos2d-JS 3.0 最终版已经发布。Cocos2d-JS 3.0 下载解压后的目录结构如图 3-12 所示。

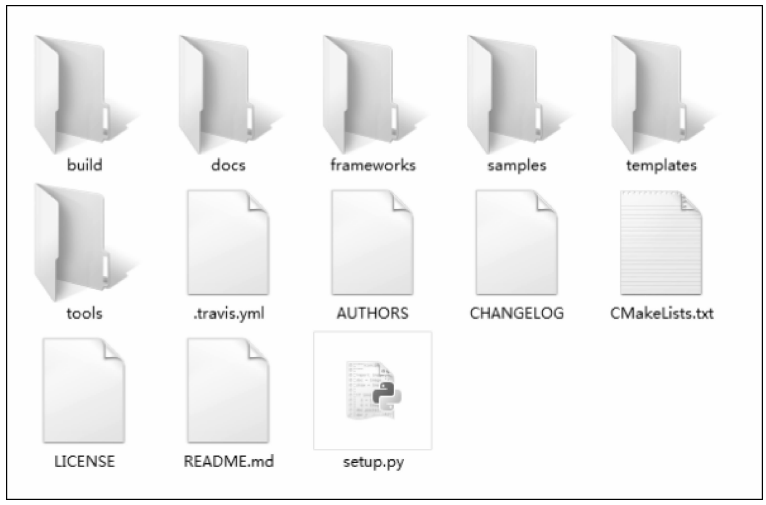


图 3-12 Cocos2d-JS 开发包内容

如果想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 3-13 所示，这里包含各个平台编译和运行案例的工程等文件。其中，cocos2d_jsb_samples. xcodeproj 文件是 Cocos2d-JS 案例的 Xcode 工程文件，cocos2d_jsb_samples. vc2012. sln 文件是 Cocos2d-JS 案例 Win32 平台下 Visual Studio 2012 解决方案文件，android-build. py 是在 Android 平台下编译和运行案例时使用的。

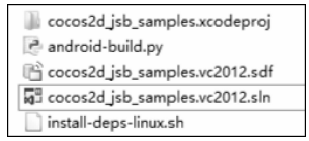


图 3-13 Cocos2d-JS 开发包 build 目录内容

如果在 Windows 下学习和开发，一般运行 cocos2d_jsb_samples. vc2012. sln 解决方案就可以了。如果启动 cocos2d_jsb_samples. vc2012. sln 解决方案，则进入如图 3-14 所示的 Visual Studio 2012 界面，其中的 js-tests 工程是 Cocos2d-JS 官方提供的案例工程。需要选中 js-tests 工程，在右击弹出的快捷菜单中选择“设置启动项目”命令，然后运行上方工具栏中的运行调试按钮▶，运行 js-tests 工程。

首次运行需要编译 Cocos2d-JS 时间会长一些，运行起来之后会出现一个 Windows 的窗口[如图 3-15(a)所示]，选择其中的一个菜单项可以运行相应的示例[如图 3-15(b)所示]。

如果想查看 js-tests 源代码，不能通过 Visual Studio 2012 查看，需要到＜Cocos2d-JS 引擎目录＞\samples\js-tests\src 目录下，使用文本编辑工具或者 WebStorm 工具。

事实上，＜Cocos2d-JS 引擎目录＞\build 目录工程文件只是编译 Cocos2d-x 库并使案例基于 JSB 方式运行，不能够通过这些工程修改案例中的 JavaScript 代码。为了能够查看、修改和运行案例中的 JavaScript 代码，可以在 WebStorm 工具中配置案例工程，管理案例。具体过程是启动 WebStorm，选择 File→New Project from Existing Files 命令，这样选择是为了从已经存在的文件创建 WebStorm 工程，弹出如图 3-16 所示对话框。选择最后一个选项，这个选项的意思是文件在本地，还没有配置 Web 服务器。

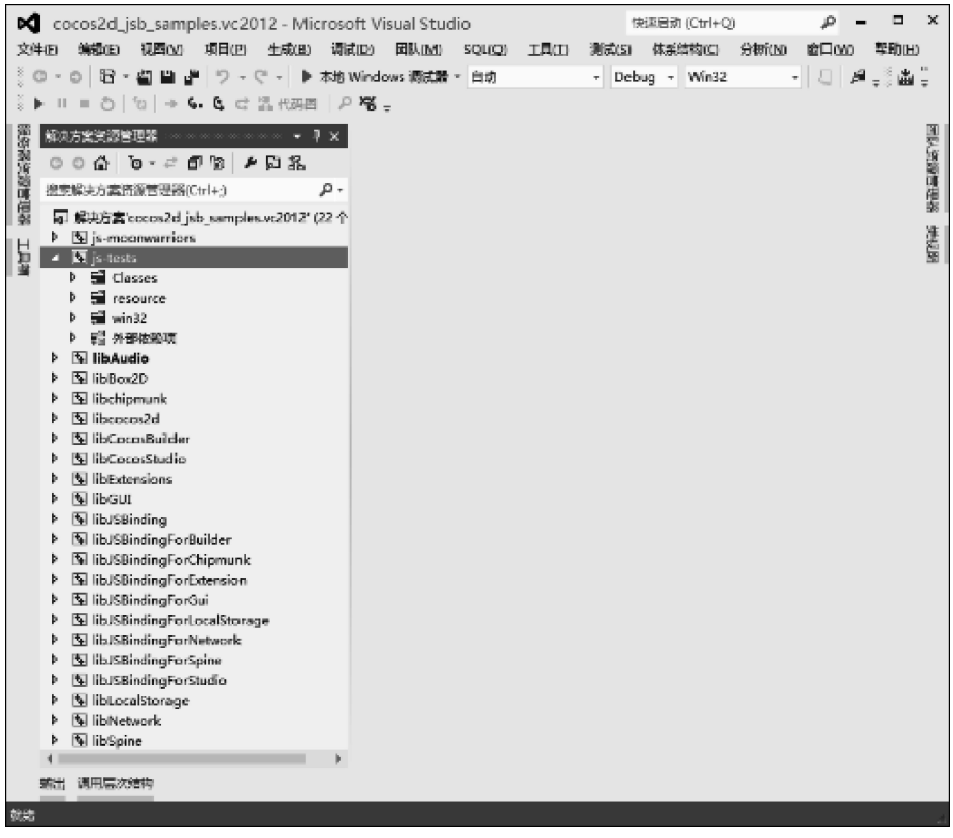
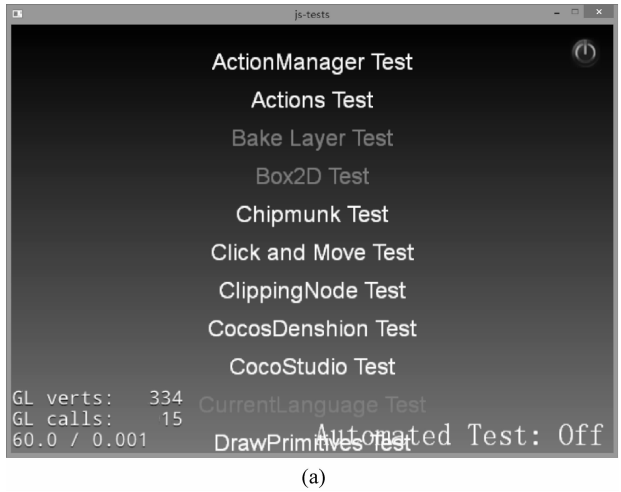
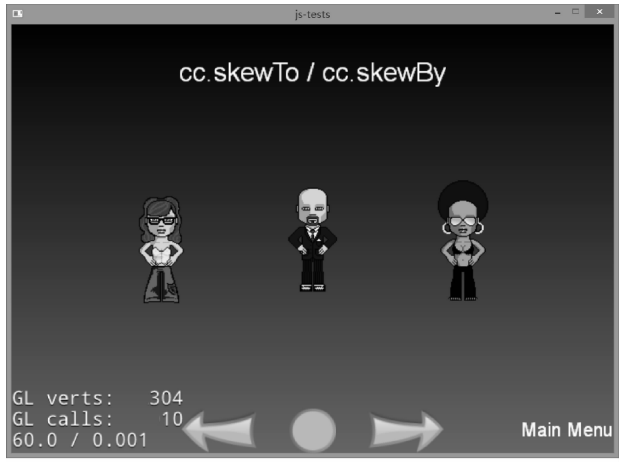


图 3-14　Cocos2d-JS 案例



(a)

图 3-15　运行案例



(b)

图 3-15 （续）

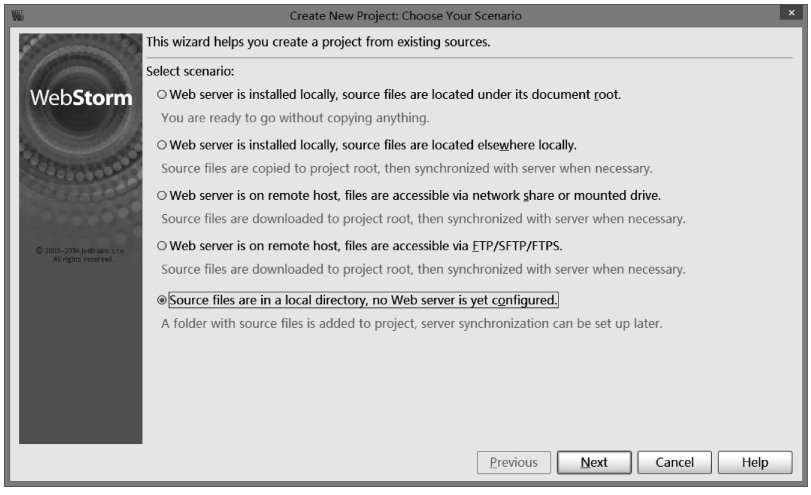


图 3-16 选择配置方案

提示 JavaScript 和 HTML 等 Web 文件运行，需要部署到一个 Web 服务器下。

在图 3-16 所示界面选择好后，单击 Next 按钮进入设置工程根目录对话框，如图 3-17 所示，选择＜Cocos2d-JS 引擎目录＞，然后单击 Project Root 按钮，设置无误后，单击 Finish 按钮完成设置过程。设置成功界面如图 3-18 所示。

在导航面板中选择 Samples→js-tests→index. html，从右击弹出的快捷菜单中选择 Debug “index. html”命令，WebStorm 会启动 Google Chrome 浏览器，如图 3-19 所示。此时发现在浏览器中已启动 js-tests 官方案例。

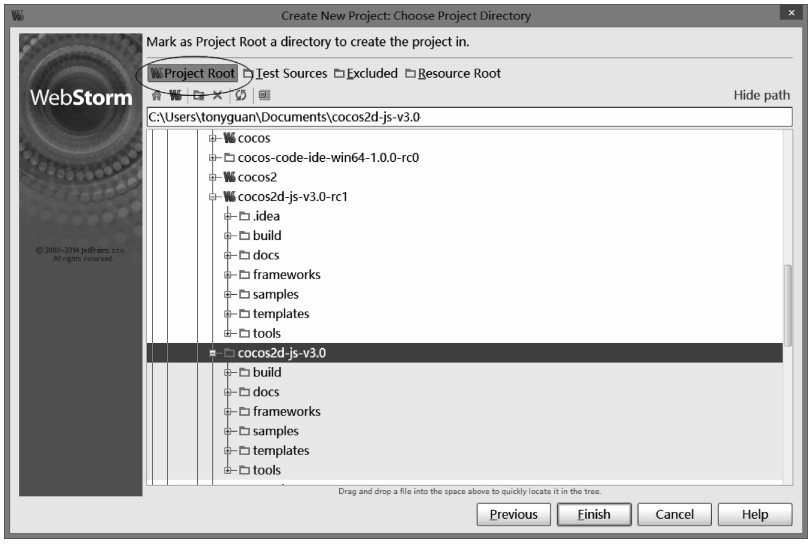


图 3-17　设置工程的根目录

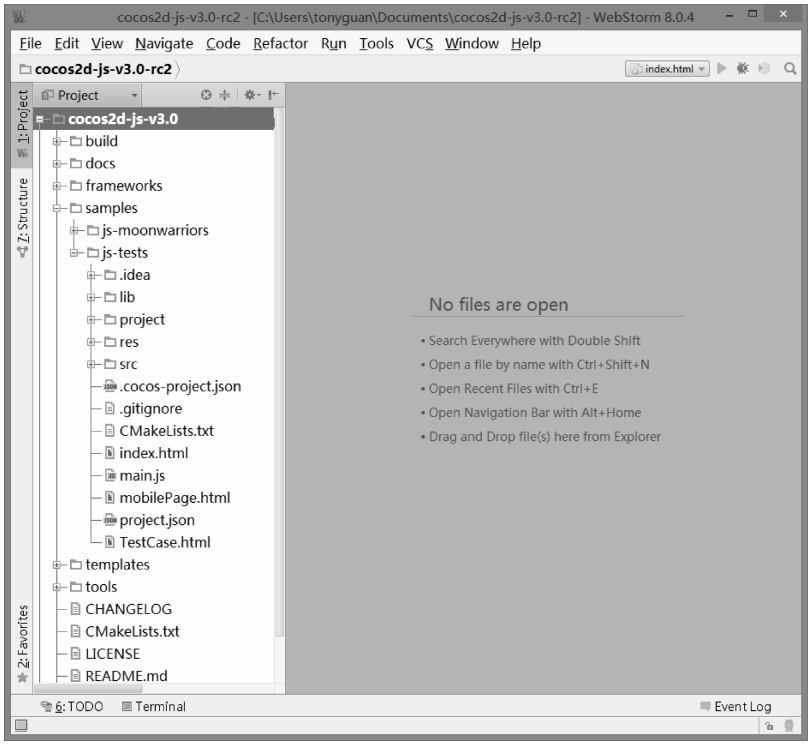


图 3-18　设置成功

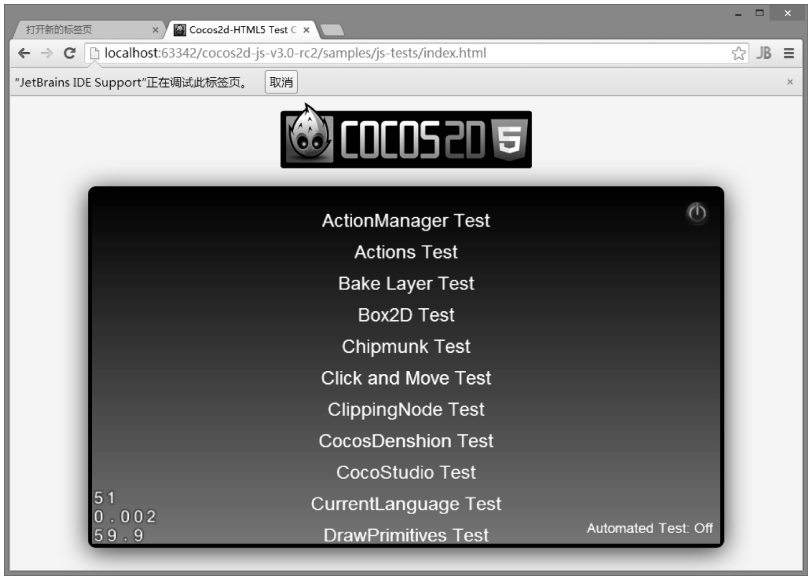


图 3-19 启动 Google Chrome 浏览器

3.3.4 使用 API 文档

从 Cocos2d-JS 官方下载的开发包中没有 API 文档，可以使用 Cocos2d-JS 官方的在线 API 文档，可以通过 http://www.cocos2d-x.org/wiki/Reference 选择 Cocos2d-JS Online API Documentation 进入在线 API 文档，如图 3-20 所示。可以在左边的文本框中输入查询条件，找到感兴趣的内容，如图 3-21 所示。

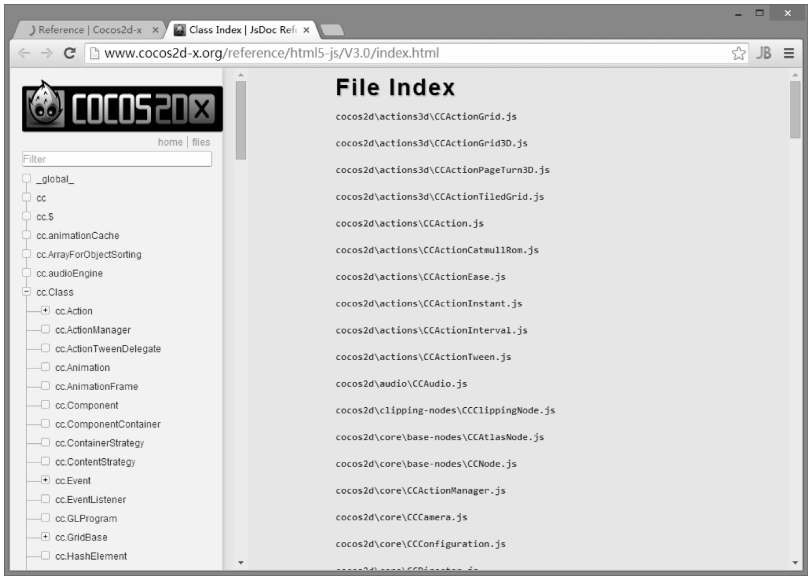


图 3-20 Cocos2d-JS 在线 API 文档

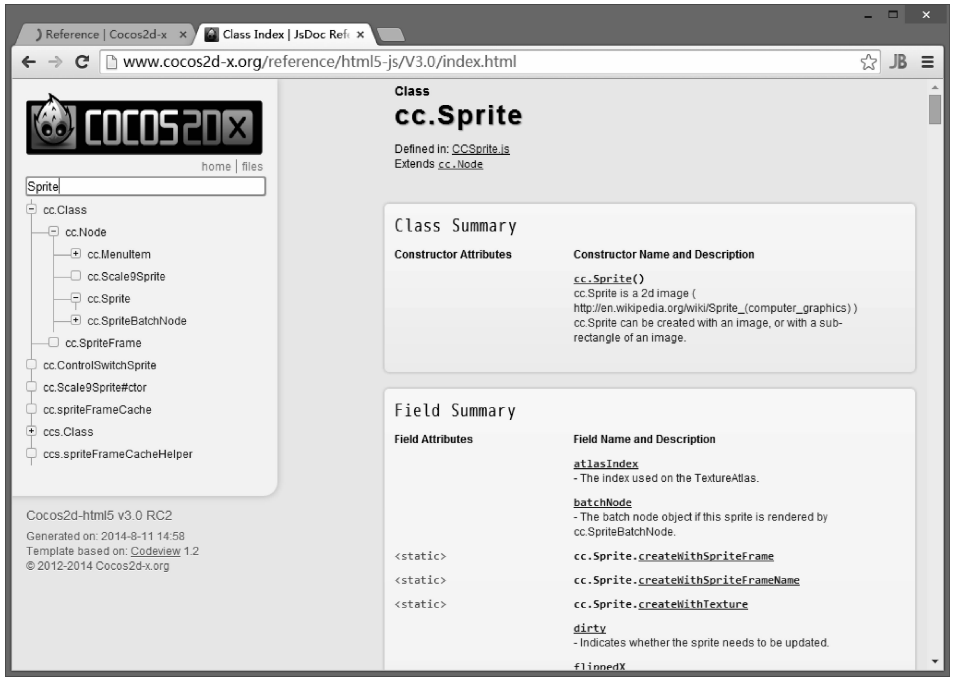


图 3-21　从在线 API 文档中搜索内容

3.4　第一个 Cocos2d-JS 游戏

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

3.4.1　创建工程

创建 Cocos2d-JS 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 WebStorm 或 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 工具很方便创建 Cocos2d-JS 工程。

首先需要在 Cocos Code IDE 工具中先配置 JavaScript 框架，打开 Cocos Code IDE 工具，选择 Window→Preferences 命令，弹出对话框，如图 3-22 所示，选择 Cocos→JavaScript，在右边的 JavaScript Frameworks 列表框中选择＜Cocos2d-JS 引擎目录＞。

JavaScript 框架配置不需要每次都进行，只是在最开始的时候配置一下，但创建工程的时候，Cocos Code IDE 工具会从这个 JavaScript 框架目录中创建工程文件。

接下来就可以创建 JavaScript 工程。选择 File→New Project 命令，如图 3-23 所示，弹出项目类型选择对话框。

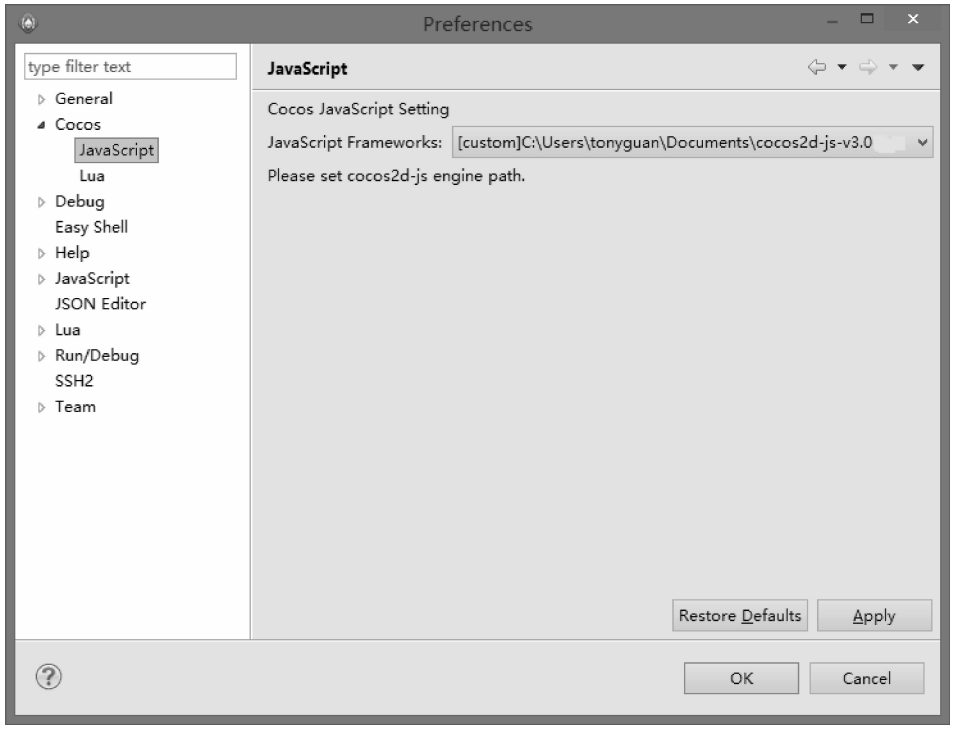


图 3-22 配置 JavaScript 框架

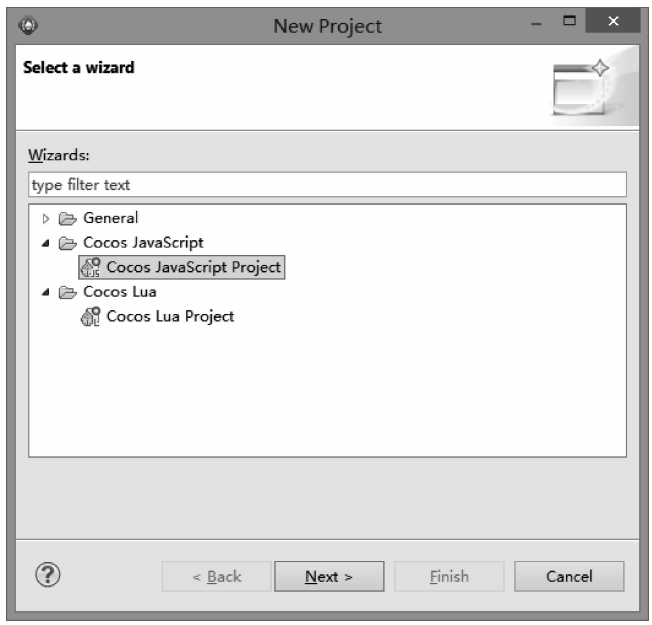


图 3-23 项目类型选择对话框

选中 CocosJavaScriptProject，然后单击 Next 按钮，弹出如图 3-24 所示的对话框。在 Project Name 文本框中输入工程名称，Create Project in Workspace 是在 Workspace 目录中创建工程，需要选中该复选框，选中 Create From Existing Resource 复选框可以从已经存在的工程创建。现在不需要选中该复选框。

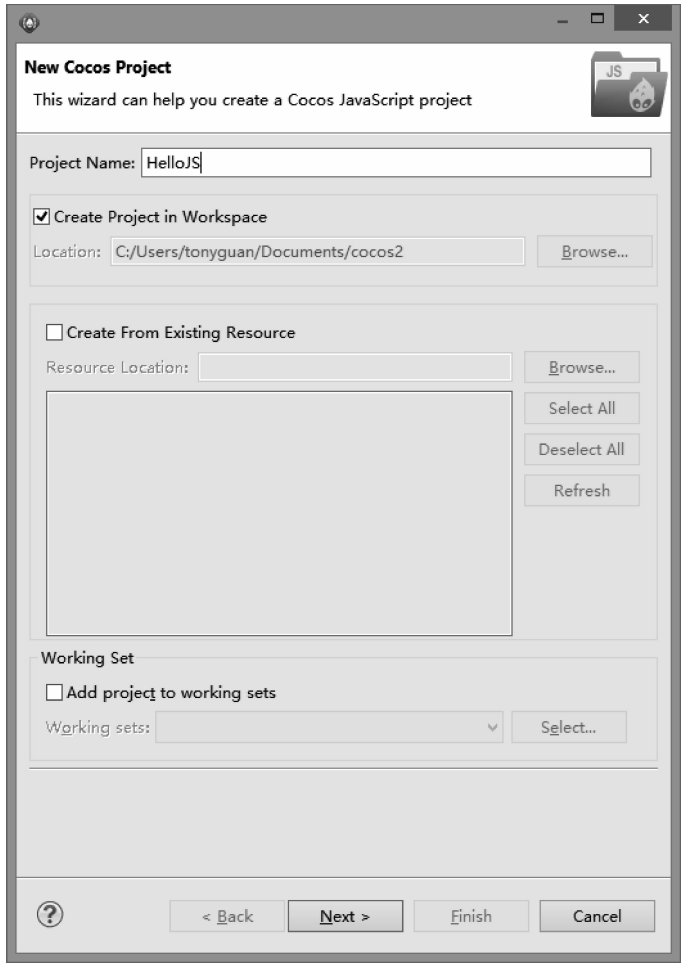


图 3-24　新建项目对话框

选择完成单击 Next 按钮进入到如图 3-25 所示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 项目是配置模拟器的朝向，其中 landscape 是横屏显示，portriat 是竖屏显示。Desktop Runtime Settings 中的 Title 文本框用于设置模拟器的标题，Desktop Windows initialize Size 用于设置模拟器的大小。Add Native Codes 用于设置添加本地代码到工程，这里不需要添加本地代码。最后单击 Finish 按钮完成创建操作，创建好工程之后，如图 3-26 所示。

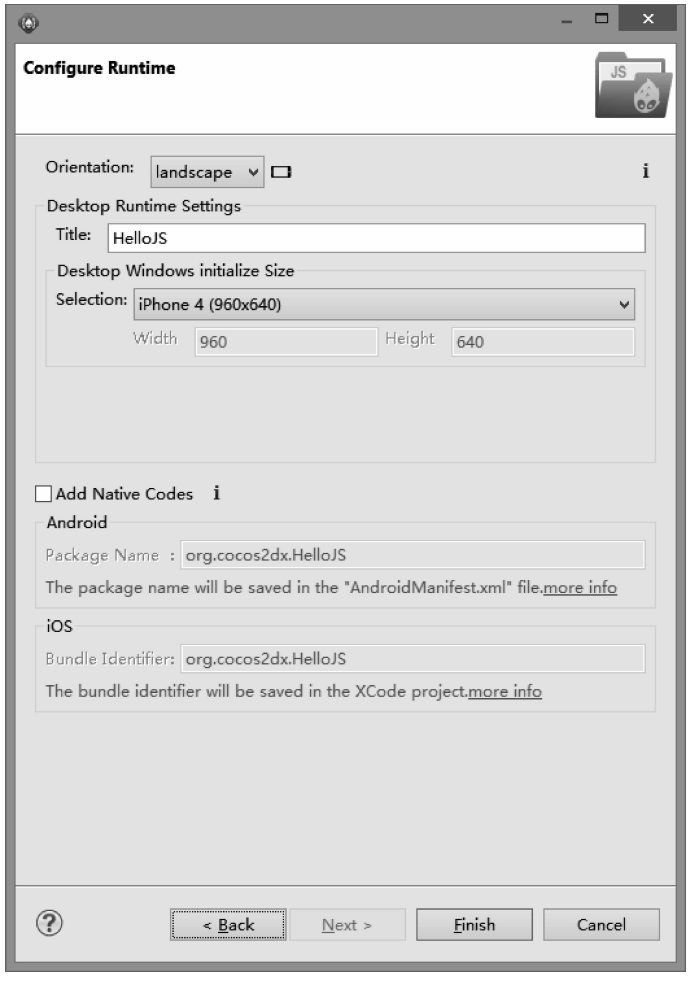


图 3-25 配置运行环境对话框

3.4.2 在 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 index.html 文件，从右击弹出的快捷菜单中选择 Run As→CocosJSBinding 命令运行刚刚创建的工程。运行结果如图 3-27 所示。

主要编写的程序代码是在 src 目录下，在本例中 app.js 文件负责处理主要的场景界面逻辑。如果想调试程序，可以设置断点，如图 3-28 所示，单击行号之前的位置，设置断点。

调试运行过程，从右击弹出的快捷菜单中选择 Debug As→CocosJSBinding 命令。如图 3-29 所示，程序运行到第 32 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。

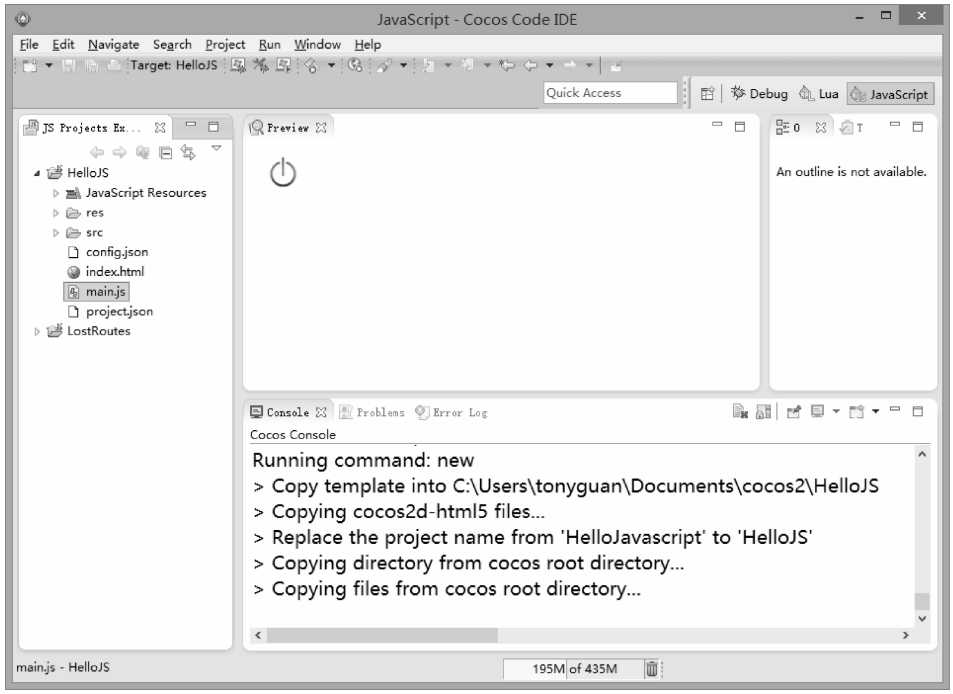


图 3-26　创建工程成功界面

图 3-27　运行工程界面

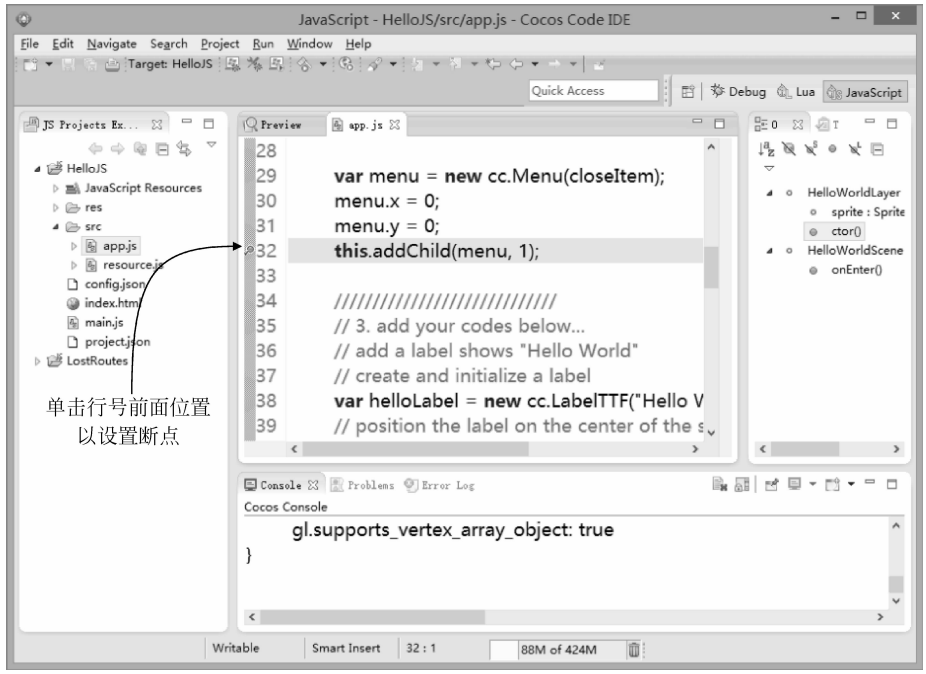


图 3-28　设置断点

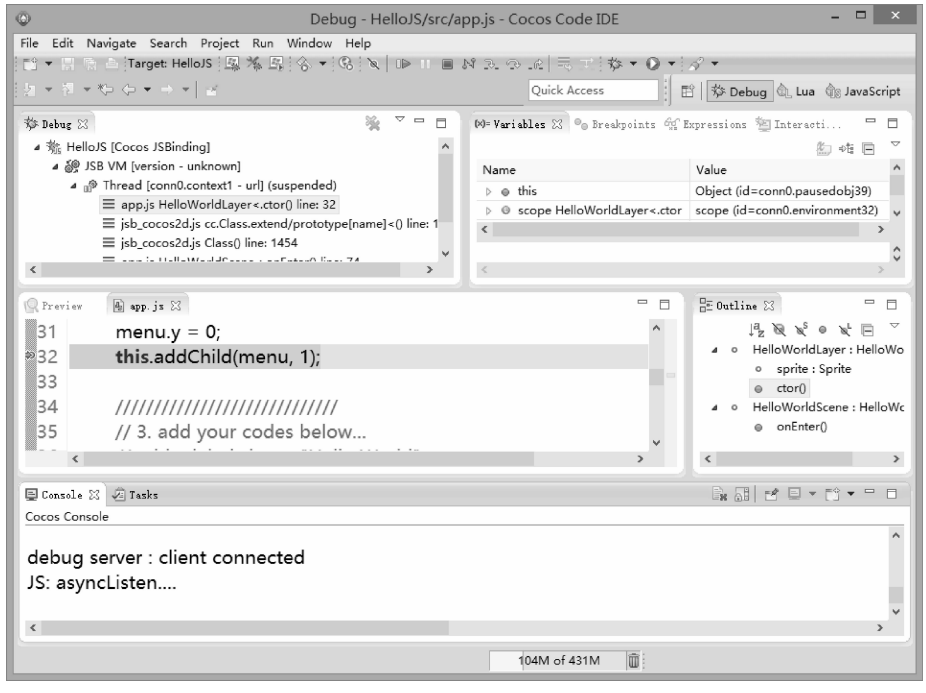


图 3-29　运行到断点挂起

在调试视图中，调试工具栏中的主要调试按钮，说明如图 3-30 所示。

3.4.3 在 WebStorm 中运行

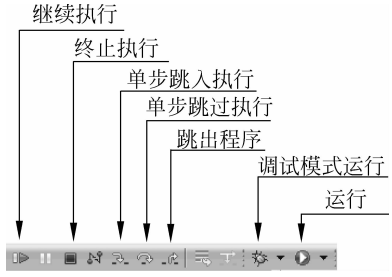


图 3-30 调试工具栏按钮

Cocos Code IDE 工具提供的运行是本地运行，即 Cocos2d-JS 程序通过 JSB 在本地运行。如果需要测试 Web 浏览器上运行情况，需要使用 WebStorm 工具。由于已经在 Cocos Code IDE 创建了工程，不需要再创建，但是需要进入到 WebStorm 中进行设置。具体设置过程参考 3.3.3 节，创建文件在本地 WebStorm 工程，为了能与 Cocos Code IDE 共用相同工程，需要设置 WebStorm 的 Project Root 为 Cocos Code IDE 的 Workspace 目录。

设置完成界面如图 3-31 所示。其中的 HelloJS 是要运行的工程，从右击弹出的快捷菜单中选择 HelloJS 中的 index.html 文件就可以运行。具体运行过程参考 3.3.3 节。运行结果如图 3-32 所示。

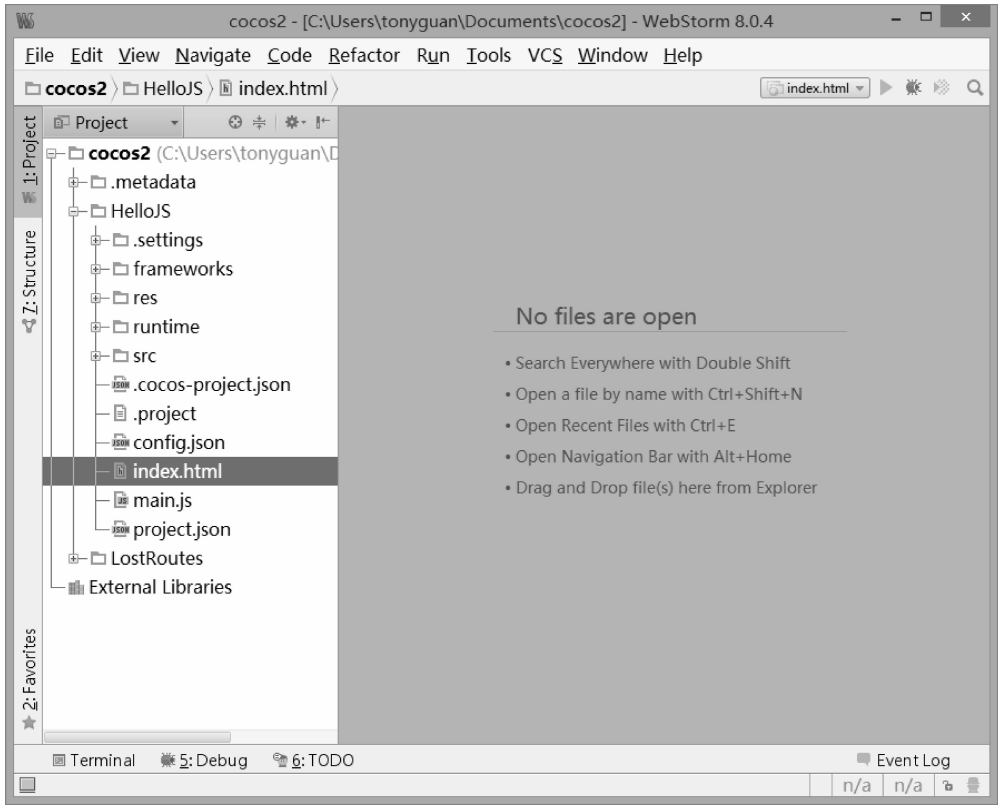


图 3-31 设置完成界面

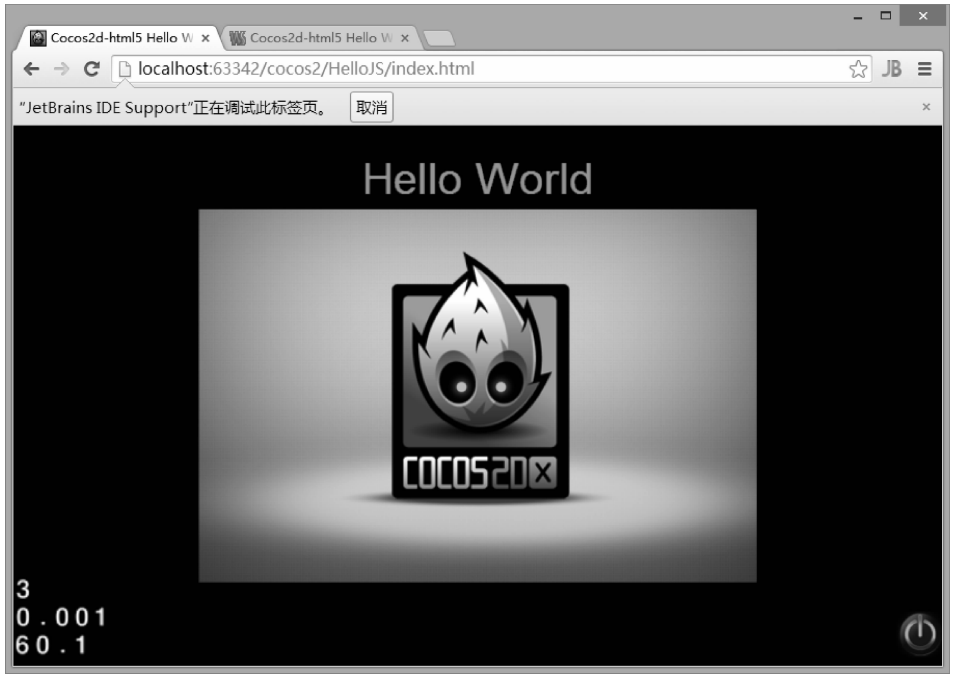


图 3-32 在浏览器中运行

3.4.4 工程文件结构

至此创建的 HelloJS 工程已经能够运行起来，下面介绍一下 HelloJS 工程中的文件结构。使用 Cocos Code IDE 打开 HelloJS 工程，左侧的导航面板如图 3-33 所示。

在图 3-33 所示导航面板中，res 文件夹存放资源文件，src 文件夹是主要的程序代码，其中的 app.js 是实现游戏场景的 JavaScript 文件，resource.js 定义资源对应的变量。HelloJS 根目录下还有 config.json、project.json、index.html 和 main.js，其中 config.json 保存模拟器运行配置信息，在创建工程时生成；project.json 是项目的配置信息；index.html 是 Web 工程的首页；main.js 是与首页 index.html 对应的 JavaScript 文件。

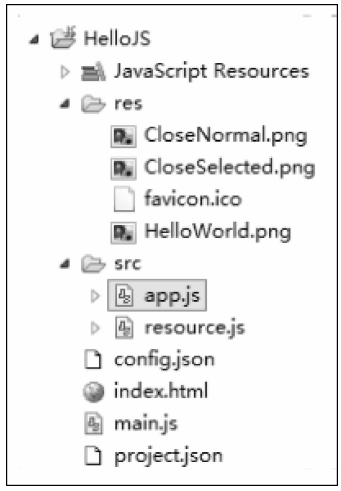


图 3-33 HelloJS 工程中的文件结构

3.4.5 代码解释

HelloJS 工程中有很多文件，下面详细解释一下它们内部的代码。

1. index.html 文件

index.html 文件只有在 Web 浏览器上运行才会启动它。index.html 代码如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Cocos2d-html5 Hello World test</title>
    <link rel="icon" type="image/GIF" href="res/favicon.ico"/>
    <meta name="apple-mobile-web-app-capable" content="yes"/>                    ①
    <meta name="full-screen" content="yes"/>
    <meta name="screen-orientation" content="portrait"/>
    <meta name="x5-fullscreen" content="true"/>
    <meta name="360-fullscreen" content="true"/>                                 ②
    <style>
        body, canvas, div {
            -moz-user-select: none;
            -webkit-user-select: none;
            -ms-user-select: none;
            -khtml-user-select: none;
            -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
        }
    </style>
</head>
<body style="padding:0; margin: 0; background: #000;">
<canvas id="gameCanvas" width="800" height="450"></canvas>                      ③
<script src="frameworks/cocos2d-html5/CCBoot.js"></script>                      ④
<script src="main.js"></script>                                                 ⑤
</body>
</html>
```

上述代码第①～②行是设置网页的 meta 信息，meta 信息是网页基本信息，这些设置能够使得 index.html 网页很好地在移动设备上显示。第③行代码放置一个 canvas 标签，canvas 标签是 HTML5 提供的，通过 JavaScript 可以在 Canvas 上绘制 2D 图形，Cocos2d-JS 在网页运行的游戏场景都是通过 Canvas 渲染出来的，Cocos2d-JS 的本地运行游戏场景的渲染是通过 OpenGL 渲染出来的。事实上，HTML5 也有类似于 OpenGL 渲染技术，它是 WebGL，但是考虑到浏览器的支持程度不同，Cocos2d-JS 没有采用 WebGL 渲染技术而是选择的 Canvas 渲染，虽然 Canvas 渲染速度不及 WebGL，但是一般的网页游戏都能满足要求。第④行代码是导入 JavaScript 文件 CCBoot.js，不需要维护该文件。第⑤行代码是导入 JavaScript 文件 main.js，需要维护该文件。

2. main.js 文件

main.js 负责启动游戏场景，无论 Web 浏览器运行还是本地运行都是通过该文件启动游戏场景的。main.js 代码如下：

```
cc.game.onStart = function(){                                                   ①
    cc.view.adjustViewPort(true);                                               ②
```

```
    cc.view.setDesignResolutionSize(800, 450, cc.ResolutionPolicy.SHOW_ALL);          ③
    cc.view.resizeWithBrowserSize(true);                                              ④
    //load resources
    cc.LoaderScene.preload(g_resources, function () {                                 ⑤
        cc.director.runScene(new HelloWorldScene());                                  ⑥
    }, this);
};
cc.game.run();                                                                        ⑦
```

上述代码第①行是启动游戏，cc.game 是一个游戏启动对象。代码第②～④行是设置游戏视图属性，其中第③行是设置游戏视图大小，它涉及屏幕适配问题，cc.ResolutionPolicy.SHOW_ALL 是屏幕适配策略。第⑤行代码是加载游戏场景所需资源，其中 g_resources 参数是加载资源的数组，该数组是在 src/resource.js 文件中定义的。第⑥行代码是运行 HelloWorldScene 场景，cc.director 是导演对象，运行 HelloWorldScene 场景会进入到该场景。第⑦行代码 cc.game.run()是运行游戏启动对象。

3. project.json 文件

项目配置信息 project.json 文件代码如下：

```
{
    "project_type": "javascript",

    "debugMode" : 1,
    "showFPS" : true,                                                                 ①
    "frameRate" : 60,                                                                 ②
    "id" : "gameCanvas",
    "renderMode" : 0,
    "engineDir":"frameworks/cocos2d-html5",

    "modules" : ["cocos2d"],                                                          ③

    "jsList" : [                                                                      ④
        "src/resource.js",                                                            ⑤
        "src/app.js"                                                                  ⑥
    ]
}
```

project.json 文件采用 JSON 字符串表示，重点关注有标号的语句，其中第①行代码设置是否显示帧率调试信息，帧率调试就是显示在左下角文字信息。第②行代码设置帧率为 60，即屏幕 1/60s 刷新一次。第③行代码是加载游戏引擎的模块，Cocos2d-JS 引擎有很多模块，模块的定义是在 HelloJS\frameworks\cocos2d-html5\moduleConfig.json，在资源管理器中才能看到该文件，这些模块在场景启动的时候加载，因此一定要根据需要导入，否则造成资源的浪费。例如，再添加一个 chipmunk 物理引擎模块，代码第③行可以修改为如下形式：

```
"modules" : ["cocos2d","chipmunk"]
```

代码第④～⑥行是声明需要加载的 JavaScript 文件，这里的文件主要是由用户编写的，

每次添加一个 JavaScript 文件到工程中，就需要在此处添加声明。

4. config.json 文件

只有在 Cocos Code IDE 中运行才需要该文件，它是配置模拟器运行信息的，该文件在工程发布时和 Web 环境下运行都没有用处。但如果想在 Cocos Code IDE 中运行，并改变模拟器大小和方向，可以修改该文件。config.json 文件代码如下：

```
{
  "init_cfg": {                                                    ①
    "isLandscape": true,                                           ②
    "name": "HelloJS",                                             ③
    "width": 960,                                                  ④
    "height": 640,                                                 ⑤
    "entry": "main.js",                                            ⑥
    "consolePort": 6050,
    "debugPort": 5086,
    "forwardConsolePort": 10088,
    "forwardUploadPort": 10090,
    "forwardDebugPort": 10086
  },
  "simulator_screen_size": [
    {
      "title": "iPhone 3Gs (480x320)",
      "width": 480,
      "height": 320
    },
    …
  ]
}
```

上述代码第①行是初始配置信息，其中第②行是设置横屏显示还是竖屏显示，第③行代码 name 属性是设置模拟器上显示的标题，第④和⑤行是设置屏幕的宽和高，第⑥行代码是设置入口文件。

5. resource.js 文件

resource.js 文件在 src 文件夹中，处于该文件夹中的文件是由用户来维护的。在 resource.js 文件中定义资源对应的变量。resource.js 文件代码如下：

```
var res = {                                                        ①
    HelloWorld_png : "res/HelloWorld.png",
    CloseNormal_png : "res/CloseNormal.png",
    CloseSelected_png : "res/CloseSelected.png"
};

var g_resources = [];                                              ②
for (var i in res) {
    g_resources.push(res[i]);                                      ③
}
```

上述代码第①行是定义 JSON 变量 res，它为每一个资源文件定义一个别名，在程序中

访问资源，资源名不要“写死”，[1]而是通过一个可配置的别名访问，这样当环境变化之后修改起来很方便。第②行代码是定义资源文件集合变量 g_resources，它的内容是通过代码第③行把 res 变量中的资源文件循环添加到 g_resources 中。当然，可以采用下面的形式逐一添加：

```
var g_resources = [
    //image
    res.HelloWorld_png,
    res.CloseNormal_png,
"res/CloseSelected.png"
];
```

放在 g_resources 变量中的资源，会在场景启动的时候加载。在 Web 浏览器下运行，如果加载的资源找不到，则会报出 404 错误。

6. app.js 文件

app.js 文件在 src 文件夹中，处于该文件夹中的文件是由用户来维护的。可以看到图 3-27 所示的场景是在 app.js 中实现的。app.js 代码如下：

```
var HelloWorldLayer = cc.Layer.extend({                                    ①
    sprite:null,                                  //定义一个精灵属性
    ctor:function () {                            //构造方法
        this._super();                            //初始化父类
        var size = cc.winSize;                    //获得屏幕大小
        var closeItem = new cc.MenuItemImage(                              ②
            res.CloseNormal_png,
            res.CloseSelected_png,
            function () {
                cc.log("Menu is clicked!");
            }, this);
        closeItem.attr({
            x: size.width - 20,
            y: 20,
            anchorX: 0.5,
            anchorY: 0.5
        });                                                                ③

        var menu = new cc.Menu(closeItem);        //通过 closeItem 菜单项创建菜单对象
        menu.x = 0;                                                        ④
        menu.y = 0;                                                        ⑤
        this.addChild(menu, 1);                   //把菜单添加到当前层上

        var helloLabel = new cc.LabelTTF("Hello World", "Arial", 38);      //创建标签对象
        helloLabel.x = size.width / 2;
```

---

① 写死被称为硬编码(英文为 Hard Code 或 Hard Coding)，硬编码指的是在软件实作上，把输出或输入的相关参数(如路径、输出的形式或格式)直接以常量的方式书写在源代码中，而非在运行时期由外界指定的设置、资源、数据或格式做出适当回应。——引自于维基百科(http://zh.wikipedia.org/zh-cn/%E5%AF%AB%E6%AD%BB)

```
        helloLabel.y = 0;
        this.addChild(helloLabel, 5);
        this.sprite = new cc.Sprite(res.HelloWorld_png);     //创建精灵对象
        this.sprite.attr({
            x: size.width / 2,
            y: size.height / 2,
            scale: 0.5,
            rotation: 180
        });
        this.addChild(this.sprite, 0);                                          ⑥

        this.sprite.runAction(
            cc.sequence(
                cc.rotateTo(2, 0),
                cc.scaleTo(2, 1, 1)
            )
        );                                          //在精灵对象上执行一个动画
        helloLabel.runAction(
            cc.spawn(
                cc.moveBy(2.5, cc.p(0, size.height - 40)),
                cc.tintTo(2.5,255,125,0)
            )
        );                                          //在标签对象上执行一个动画
        return true;
    }
});

var HelloWorldScene = cc.Scene.extend({                                         ⑦
    onEnter:function () {                                                       ⑧
        this._super();                                                          ⑨
        var layer = new HelloWorldLayer();                                      ⑩
        this.addChild(layer);            //把 HelloWorldLayer 层放到 HelloWorldScene 场景中
    }
});
```

在 app.js 文件中声明了两个类 HelloWorldLayer(见代码第①行)和 HelloWorldScene(见代码第⑦行),然后通过 HelloWorldScene 实例化 HelloWorldLayer,见代码第⑩行。HelloWorldScene 是场景,HelloWorldLayer 是层,场景包含若干个层。场景和层会在 3.5 节具体介绍。

另外,上述代码第②行是创建一个图片菜单项对象,单击该菜单项的时候回调 function 方法。

提示 cc.MenuItemImage 中 res.CloseNormal_png 和 res.CloseSelected_png 变量是在 resource.js 文件中定义的资源文件别名。在后文中,res.开头的变量都是资源文件的别名,这里不再详细解释。

第③行代码是菜单项对象的位置，其中 closeItem.attr({…})语句可以设置多个属性，类似的还有代码第⑥行，采用 JSON 格式表示，x 属性表示 x 轴坐标，y 属性表示 y 轴坐标，anchorX 表示 x 轴锚点，anchorY 表示 y 轴锚点。关于锚点的概念后面小节介绍。关于精灵 x 和 y 轴属性，也可以通过代码④和⑤方式设置。

代码第⑧行是声明 onEnter 方法，它是在进入 HelloWorldScene 场景时回调的。onEnter 方法是重写父类的方法，必须通过 this._super()语句调用父类的 onEnter 方法，见代码第⑨行所示。

3.5 Cocos2d-JS 核心概念

Cocos2d-JS 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，如导演、场景和层等概念。当然，也有些传统的游戏的概念。Cocos2d-JS 中核心概念包括导演、动作、场景、效果、层、粒子运动、节点、地图、精灵、物理引擎、菜单。

本节介绍导演、场景、层、精灵、菜单概念以及对应的类，由于节点概念很重要，会在下一节详细介绍。而其他的概念在后面的章节中介绍。

3.5.1 导演

导演类 cc.Director 用于管理场景，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：

```
var director = cc.Director._getInstance();
```

也可以在程序中直接使用 cc.director，该对象是在框架内使用如下语句进行赋值：

```
cc.director = cc.Director._getInstance();
```

所以，cc.director 是 cc.Director 的实例对象。

导演对象职责如下：

(1) 访问和改变场景。

(2) 访问 Cocos2d-JS 的配置信息。

(3) 暂停、继续和停止游戏。

(4) 转换坐标。

3.5.2 场景

场景类 cc.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：

(1) 展示类场景。播放视频或简单地在图像上输出文字，来实现游戏的开场介绍、胜利和失败提示、帮助介绍。

(2) 选项类场景。主菜单、设置游戏参数等。

(3) 游戏场景。这是游戏的主要内容。

场景类 cc. Scene 的类图如图 3-34 所示。从类图可见,Scene 继承了 Node 类,Node 是一个重要的类,很多类都从 Node 类派生而来,其中有 Scene、Layer 等。

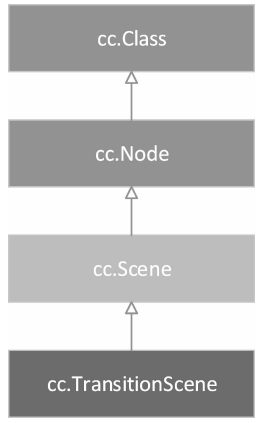


图 3-34 cc. Scene 类图

3.5.3 层

层是写游戏的重点,用户大约 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。图 3-35 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 3-35 所示,从上到下依次是菜单层、精灵层、背景层。Cocos2d-JS 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到该事件。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。

主菜单画面

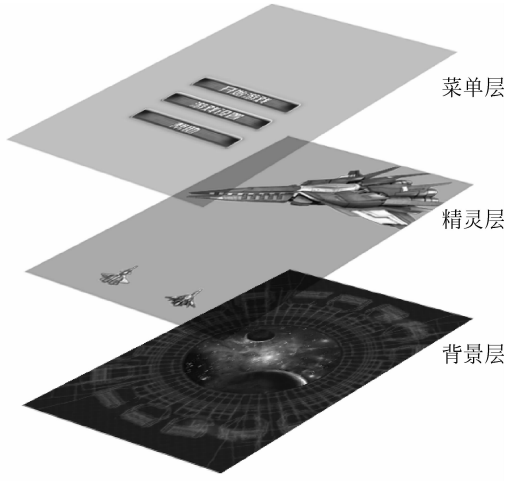


图 3-35 层叠加

层类 cc. Layer 的类图如图 3-36 所示。

3.5.4 精灵

精灵类 cc. Sprite 是游戏中非常重要的概念,它包括敌人、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。

cc. Sprite 类图如图 3-37 所示。从图中可见,cc. Sprite 是 cc. Node 子类,cc. Sprite 包含

很多类型。例如，物理引擎精灵类 PhysicsSprite 也都属于精灵。

3.5.5 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-JS 中菜单类是 cc.Menu。cc.Menu 类图如图 3-38 所示。从类图可见，cc.Menu 类派生于 cc.Layer。

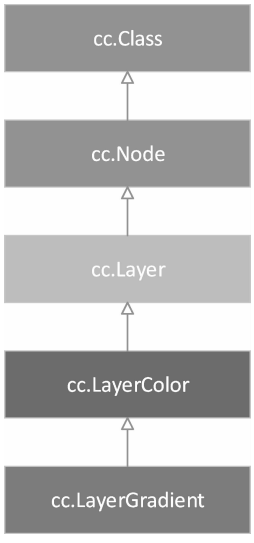


图 3-36 cc.Layer 类图

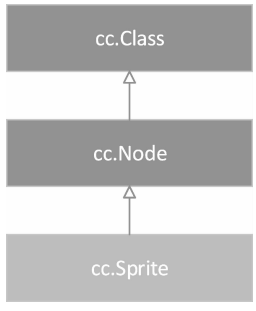


图 3-37 cc.Sprite 类图

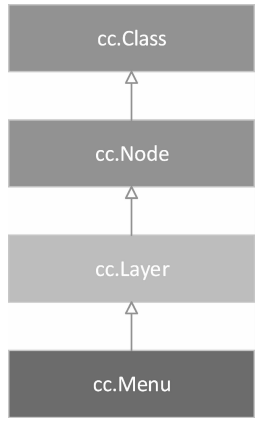


图 3-38 cc.Menu 类图

在菜单中又包含了菜单项 cc.MenuItem，cc.MenuItem 类图如图 3-39 所示。从图中可见，菜单项 cc.MenuItem 有很多种形式的子类，如 cc.MenuItemLabel、cc.MenuItemSprite 和 cc.MenuItemToggle，它表现出不同的效果。每个菜单项都有三个基本状态：正常、选中和禁止。

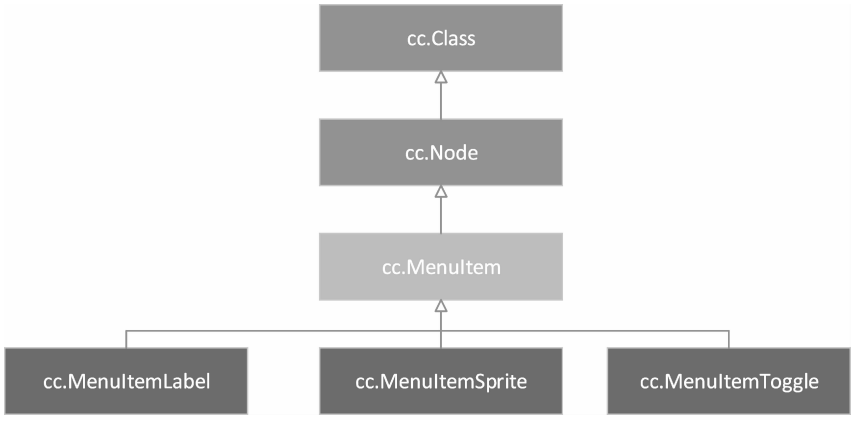


图 3-39 cc.MenuItem 类图

3.6 Node 与 Node 层级架构

Cocos2d-JS 采用层级(树形)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 3-40 所示。

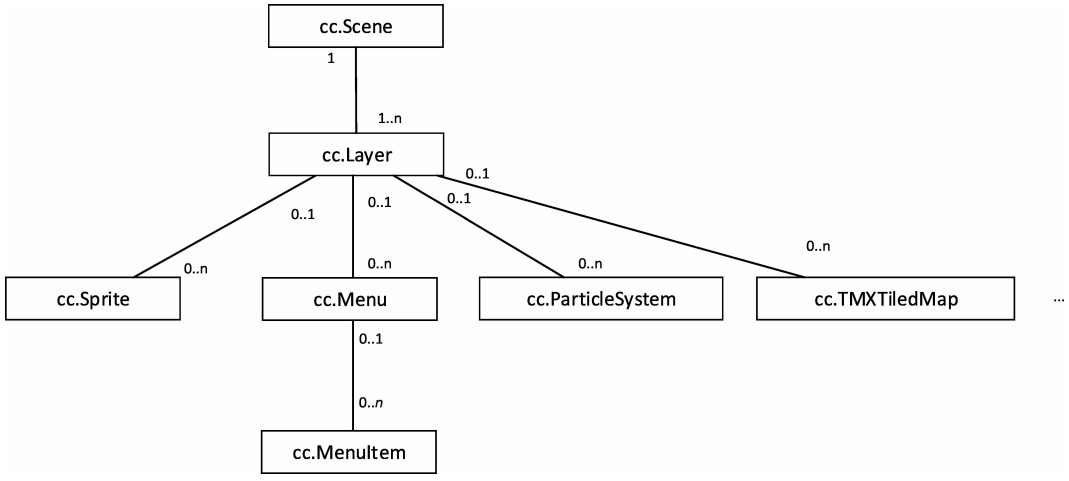


图 3-40 节点的层级结构

这些节点有一个共同的父类 cc. Node,部分子类 cc. Node 类图如图 3-41 所示。cc. Node 类是 Cocos2d-JS 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。

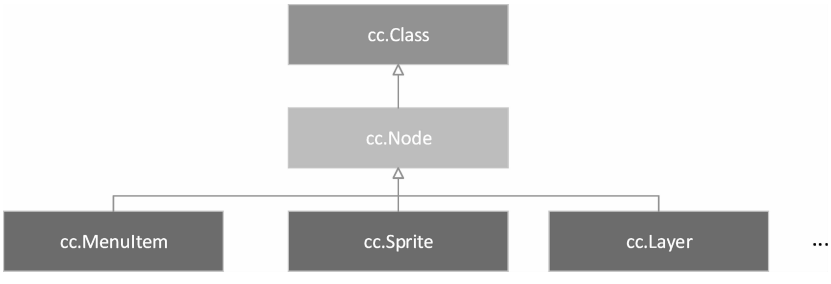


图 3-41 cc. Node 类图

3.6.1 Node 中重要的操作

cc. Node 作为根类,它有很多重要的方法。下面分别介绍一下:

(1) var childNode=new cc.Node(),创建节点。

(2) node.addChild(childNode, 0, 123),增加新的子节点。第二个参数是 Z 轴绘制顺序,第三个参数是标签。

(3) var childNode=node.getChildByTag(123),查找子节点。通过标签查找子节点。

(4) node.removeChildByTag(123, true),通过标签删除子节点,并停止所有该子节点上的一切动作。

(5) node.removeChild(childNode, true),删除 childNode 节点,并停止所有该子节点上的一切动作。

(6) 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。

(7) node.removeFromParentAndCleanup(true),从父节点删除 node 节点,并停止所有该节点上的一切动作。

3.6.2 Node 中重要的属性

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)。看看下面的几种情况:

(1) 图 3-42 所示是 anchorPoint 为(0.5,0.5)情况,这是默认情况。

(2) 图 3-43 所示是 anchorPoint 为(0.0,0.0)情况。

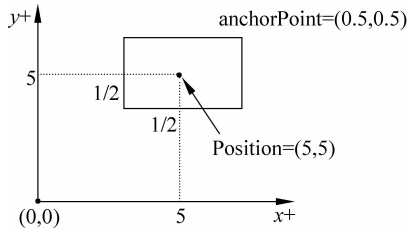


图 3-42 anchorPoint 为(0.5,0.5)

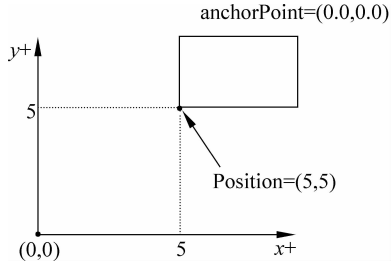


图 3-43 anchorPoint 为(0.0,0.0)

(3) 图 3-44 所示是 anchorPoint 为(1.0,1.0)情况。

(4) 图 3-45 所示是 anchorPoint 为(0.66, 0.5)情况。

为了进一步了解 anchorPoint 的使用情况,修改 HelloJS 实例。在 app.js 的 ctor 方法中修改 label 代码:

```
var helloLabel = new cc.LabelTTF("Hello World", "Arial", 38);

helloLabel.setPosition(size.width / 2, 0);                    ①
```

```
//helloLabel.x = size.width / 2;                                    ②
//helloLabel.y = 0;                                                 ③

helloLabel.setAnchorPoint(cc.p(1.0, 1.0));                          ④
//helloLabel.anchorX = 1.0;                                         ⑤
//helloLabel.anchorY = 1.0;                                         ⑥

this.addChild(helloLabel, 5);
```

上述代码第①行调用 setPosition(x, y)方法设置 position 属性,也可以直接通过属性 helloLabel.x 和 helloLabel.y 设置,见代码第②行和第③行。代码第④行调用 setAnchorPoint(x, y)方法设置 anchorPoint 属性,也可以直接通过属性 helloLabel.anchorX 和 helloLabel.anchorY 设置,见代码第⑤行和第⑥行。

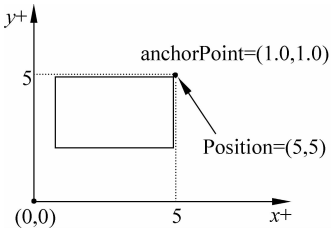


图 3-44 anchorPoint 为(1.0,1.0)

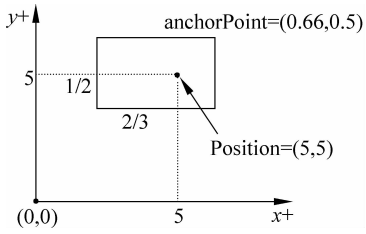


图 3-45 anchorPoint 为(0.66, 0.5)

此外,由于多个属性需要设置,可以通过 helloLabel.attr({…})语句进行设置。代码如下:

```
helloLabel.attr({
    x: size.width / 2,
    y: 0,
    anchorX: 1.0,
    anchorY: 1.0
});
```

运行结果如图 3-46 所示,helloLabel 设置了 anchorPoint 为(1.0,1.0)。

3.6.3 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、维护。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(cc.Scheduler)对精灵等对象的运行进行调度。因为 cc.Node 类封装了 cc.Scheduler 类,所以也可以直接使用 cc.Node 中定时器相关方法。

cc.Node 中定时器相关方法主要有:

(1) scheduleUpdate()。每个 Node 对象只要调用该方法,那么这个 Node 对象就会定时地每帧回调用一次自己的 update(dt)方法。

(2) schedule(callback_fn, interval, repeat, delay)。与 scheduleUpdate 方法功能一

图 3-46　helloLabel 的 anchorPoint 为(1.0,1.0)

样,不同的是,可以指定回调方法(通过 callback_fn 指定)。interval 是时间间隔,repeat 是执行的次数,delay 是延迟执行的时间。

(3) unscheduleUpdate()。停止 update(dt)方法调度。

(4) unschedule(callback_fn)。可以指定具体方法停止调度。

(5) unscheduleAllCallbacks()。可以停止所有的调度。

为了进一步了解游戏循环与调度的使用,修改 HelloJS 实例。

修改 app.js 代码,添加 update(dt)声明。代码如下:

```
var HelloWorldLayer = cc.Layer.extend({
    sprite: null,
    ctor: function () {
        …
        var closeItem = new cc.MenuItemImage(
            res.CloseNormal_png,
            res.CloseSelected_png,
            function () {
                cc.log("Menu is clicked!");
                this.unscheduleUpdate();
            }, this);

        …
        var helloLabel = new cc.LabelTTF("Hello World", "Arial", 38);
```

```
            helloLabel.attr({
                x: size.width / 2,
                y: 0,
                anchorX: 1.0,
                anchorY: 1.0
            });

            helloLabel.setTag(123);                                                    ①
            //更新方法
            this.scheduleUpdate();                                                     ②
            //this.schedule(this.update, 1.0/60, cc.REPEAT_FOREVER, 0.1);              ③

            this.addChild(helloLabel, 5);

            //add "HelloWorld" splash screen
            this.sprite = new cc.Sprite(res.HelloWorld_png);
            this.sprite.attr({
                x: size.width / 2,
                y: size.height / 2,
                scale: 0.5,
                rotation: 180
            });
            this.addChild(this.sprite, 0);

            this.sprite.runAction(
                cc.sequence(
                    cc.rotateTo(2, 0),
                    cc.scaleTo(2, 1, 1)
                )
            );
            helloLabel.runAction(
                cc.spawn(
                    cc.moveBy(2.5, cc.p(0, size.height - 40)),
                    cc.tintTo(2.5, 255, 125, 0)
                )
            );
            return true;
        },
        update: function (dt) {                                                        ④
            var label = this.getChildByTag(123);                                       ⑤
            label.x = label.x + 0.2;                                                   ⑥
            label.y = label.y + 0.2;                                                   ⑦
        }
    });
```

为了能够在 ctor 方法之外访问标签对象 helloLabel，需要为标签对象设置 Tag 属性，其中的第①行代码就是设置 Tag 属性为 123，第⑤行代码是通过 Tag 属性重新获得这个标签对象。

为了能够开始调度，还需要在 ctor 方法中调用 scheduleUpdate（见第②行代码）或

schedule(见第③行代码)。

代码第④行的 update(dt)方法是在调度方法,精灵等对象的变化逻辑都是在这个方法中编写的。这个例子很简单,只是让标签对象动起来,第⑥行代码就是改变它的位置。

为了省电等,如果不再使用调度,一定不要忘记停止调度。可以在 Close 菜单项的单击事件中停止调度。代码如下:

```
var closeItem = new cc.MenuItemImage(
            res.CloseNormal_png,
            res.CloseSelected_png,
            function () {
                cc.log("Menu is clicked!");
                this.unscheduleUpdate();
            }, this);
```

代码 this.unscheduleUpdate()就是停止调度 update,如果是其他的调度方法,则可以采用 unschedule 或 unscheduleAllSelectors 停止。

3.7 Cocos2d-JS 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 等平台应用开发的时候使用的二维坐标系的原点在左上角,而在 Cocos2d-JS 坐标系中其原点在左下角,而且 Cocos2d-JS 坐标系又可以分为世界坐标系和模型坐标系。

3.7.1 UI 坐标

UI 坐标就是 Android 和 iOS 等应用开发的时候使用的二维坐标系。它的原点位于左上角,如图 3-47 所示。

UI 坐标原点是在左上角,x 轴向右为正,y 轴向下为正。在 Android 和 iOS 等平台使用的视图、控件等都遵守这个坐标系。然而在 Cocos2d-JS 中默认不是采用 UI 坐标,但是有的时候也会用到 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)。触摸对象提供了很多获得位置信息的方法,如下面代码所示:

```
vartouchLocation = touch.getLocationInView();
```

使用 getLocationInView()方法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。

3.7.2 OpenGL 坐标

在上面提到了 OpenGL 坐标,OpenGL 坐标是一种三维坐标。由于 Cocos2d-JS 的 JSB 是采用 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,如图 3-48 所示。

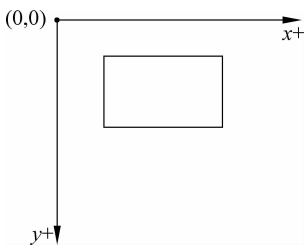


图 3-47　UI 坐标

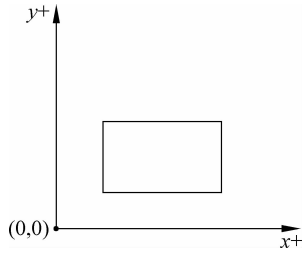


图 3-48　OpenGL 坐标

一般会通过一个触摸对象获得 OpenGL 坐标位置，如下面代码所示：

```
var touchLocation = touch.getLocation();
```

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 3-49(a)所示。左手坐标是 z 轴指向屏幕里，如图 3-49(b)所示。OpenGL 坐标是右手坐标，而微软平台的 Direct3D[①] 是左手坐标。

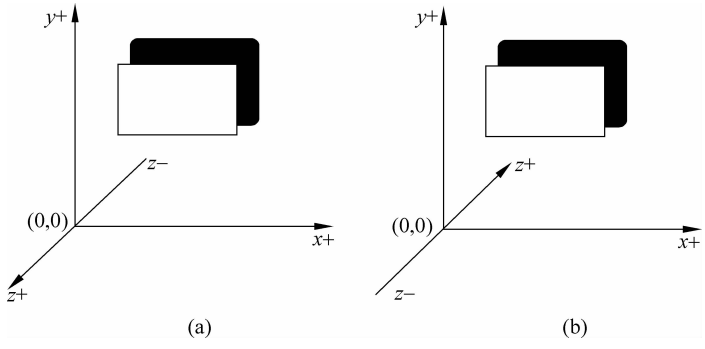


图 3-49　三维坐标

3.7.3　世界坐标和模型坐标

由于 OpenGL 坐标又可以分为世界坐标和模型坐标，所以 Cocos2d-JS 的坐标也有世界坐标和模型坐标。

你是否有过这样的问路经历：张三会告诉你向南走 1km，再向东走 500m。而李四会告诉你向右走 1km，再向左走 500m。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，他把地球作为参照物，表述位置使用地理的东、南、西和北。而李四

① Direct3D(简称 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。—— 引自于维基百科(http://zh.wikipedia.org/wiki/Direct3D)

采用的坐标是模型坐标，他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。

从图 3-50 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)。

有时需要将世界坐标与模型坐标互相转换。可以通过 Node 对象的如下方法实现：

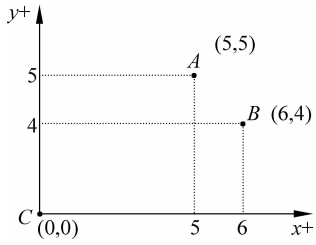


图 3-50　世界坐标和模型坐标

(1) {cc. Point} convertToNodeSpace(worldPoint)。将世界坐标转换为模型坐标。

(2) {cc. Point} convertToNodeSpaceAR(worldPoint)。将世界坐标转换为模型坐标。AR 表示相对于锚点。

(3) {cc. Point} convertTouchToNodeSpace(touch)。将世界坐标中触摸点转换为模型坐标。

(4) {cc. Point} 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。

(5) {cc. Point} convertToWorldSpace(nodePoint)。将模型坐标转换为世界坐标。

(6) {cc. Point} convertToWorldSpaceAR(nodePoint)。将模型坐标转换为世界坐标。AR 表示相对于锚点。

下面通过两个例子了解一下世界坐标与模型坐标的互相转换。

1. 世界坐标转换为模型坐标

图 3-51 所示是世界坐标转换为模型坐标实例运行结果。

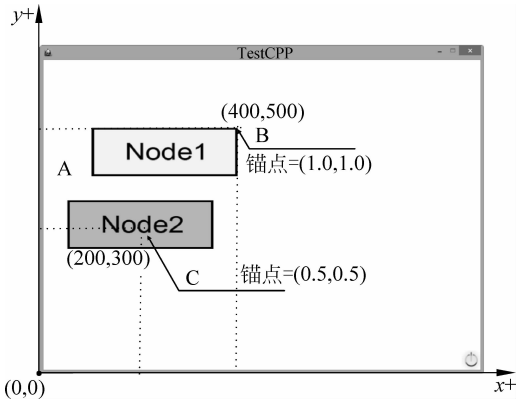


图 3-51　世界坐标转换为模型坐标

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。

编写代码如下：

```
var HelloWorldLayer = cc.Layer.extend({
    sprite: null,
    ctor: function () {
        this._super();

        var size = cc.winSize;
        var closeItem = new cc.MenuItemImage(
            res.CloseNormal_png,
            res.CloseSelected_png,
            function () {
                cc.log("Menu is clicked!");
            }, this);
        closeItem.attr({
            x: size.width - 20,
            y: 20,
            anchorX: 0.5,
            anchorY: 0.5
        });
        var menu = new cc.Menu(closeItem);
        menu.x = 0;
        menu.y = 0;
        this.addChild(menu, 1);

        //创建背景
        var bg = new cc.Sprite(res.bg_png);                                    ①
        bg.setPosition(size.width / 2, size.height / 2);
        this.addChild(bg, 2);                                                  ②

        //创建 Node1
        var node1 = new cc.Sprite(res.node1_png);                              ③
        node1.setPosition(400, 500);
        node1.setAnchorPoint(1.0, 1.0);
        this.addChild(node1, 2);                                               ④

        //创建 Node2
        var node2 = new cc.Sprite(res.node2_png);                              ⑤
        node2.setPosition(200, 300);
        node2.setAnchorPoint(0.5, 0.5);
        this.addChild(node2, 2);                                               ⑥

        var point1 = node1.convertToNodeSpace(node2.getPosition());            ⑦
        var point3 = node1.convertToNodeSpaceAR(node2.getPosition());          ⑧

        cc.log("Node2 NodeSpace = (" + point1.x + "," + point1.y + ")");
        cc.log("Node2 NodeSpaceAR =  (" + point3.x + "," + point3.y + ")");

        return true;
    }
});
```

代码第①～②行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。代码第③～④行是创建 Node1 对象，并设置了位置和锚点属性。代码第⑤～⑥行是创建 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：

```
JS: Node2 NodeSpace = (100, - 100)
JS: Node2 NodeSpaceAR =   ( - 200, - 200)
```

结合图 3-51 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 3-52 中的 A 点），不难计算出 A 点的世界坐标是（100，400），那么 convertToNodeSpace 方法就是 A 点坐标减去 C 点坐标，结果是（－100，100）。而 convertToNodeSpaceAR 方法要考虑锚点，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

图 3-52 所示是模型坐标转换为世界坐标实例运行结果。

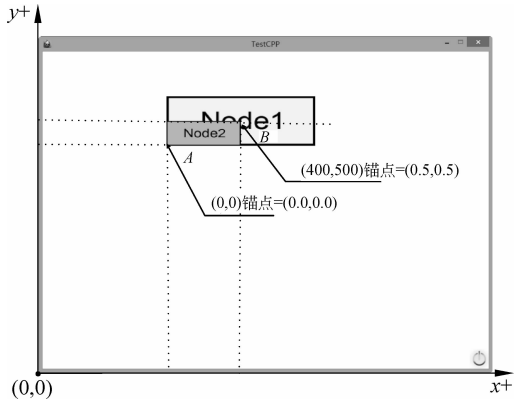


图 3-52 模型坐标转换为世界坐标

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：

```
var HelloWorldLayer = cc.Layer.extend({
    sprite: null,
    ctor: function () {
        this._super();
        var size = cc.winSize;
        var closeItem = new cc.MenuItemImage(
            res.CloseNormal_png,
            res.CloseSelected_png,
            function () {
```

```
                cc.log("Menu is clicked!");
            }, this);
        closeItem.attr({
            x: size.width - 20,
            y: 20,
            anchorX: 0.5,
            anchorY: 0.5
        });
        var menu = new cc.Menu(closeItem);
        menu.x = 0;
        menu.y = 0;
        this.addChild(menu, 1);

        //创建背景
        var bg = new cc.Sprite(res.bg_png);
        bg.setPosition(size.width / 2, size.height / 2);
        this.addChild(bg, 2);

        //创建 Node1
        var node1 = new cc.Sprite(res.node1_png);
        node1.setPosition(400, 500);
        node1.setAnchorPoint(0.5, 0.5);
        this.addChild(node1, 2);

        //创建 Node2
        var node2 = new cc.Sprite(res.node2_png);
        node2.setPosition(0, 0);                                                    ①
        node2.setAnchorPoint(0, 0); ;                                               ②

        node1.addChild(node2, 2);                                                   ③

        var point2 = node1.convertToWorldSpace(node2.getPosition());               ④
        var point4 = node1.convertToWorldSpaceAR(node2.getPosition());             ⑤

        cc.log("Node2 WorldSpace = (" + point2.x + "," + point2.y + ")");
        cc.log("Node2 WorldSpaceAR =  (" + point4.x + "," + point4.y + ")");

        return true;
    }
});
```

上述代码主要关注第③行，它是将 Node2 放到 Node1 中，这是与之前的代码的区别。这样第①行设置的坐标就变成了相对于 Node1 的模型坐标了。

第④行代码将 Node2 的模型坐标转换为世界坐标。而第⑤行代码是类似的，它是相对于锚点的位置。

运行结果如下：

```
JS: Node2 WorldSpace = (250,450)
JS: Node2 WorldSpaceAR =  (400,500)
```

图 3-52 所示的位置可以用世界坐标描述。将代码第①～③行修改如下：

```
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，可以了解 Cocos2d-JS 开发环境的搭建，熟悉 Cocos2d-JS 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，还介绍了 Node 和 Node 层级架构，以及 Cocos2d-JS 的坐标系。