第 3 章 Hello Cocos2d-x

从这一章开始介绍 Cocos2d-x 引擎开发，本书重点介绍基于 Windows 下使用 Visual Studio 2012 进行开发。本章先从一个 HelloWorld 入手，介绍 Cocos2d-x 的基本开发流程，以及 Cocos2d-x 生命周期、Cocos2d-x 核心知识体系。

3.1 第一个 Cocos2d-x 游戏

我们编写的第一个程序一般习惯上都命名为 HelloWorld，从它开始再学习其他的内容。本节介绍的第一个 Cocos2d-x 游戏也命名为 HelloWorld。

3.1.1 创建工程

在 Cocos2d-x 早期版本中，创建工程是通过安装在 Visual Studio 中的工程模板而创建的，而 Cocos2d-x 3.x 创建工程是通过 Cocos2d-x 提供的命令工具 cocos 实现的。

cocos 是使用 Python[①] 脚本编写的，cocos 工具的运行需要安装 Python 环境。Python 下载地址为 https://www.python.org/的，需要注意它目前有 Python 3 和 Python 2 可以下载。要选择 Python 2 版本，因为 cocos 是使用 Python 2 编写的，这两个版本在语法上有区别，使用 Python 3 导致无法使用 Cocos2d-x 3.x 提供的工具。

为了能够在任何目录下执行 Python，需要配置系统环境变量 Path，在 Windows 系统中打开“控制面板”→“系统和安全”→“系统”→“高级系统设置”，弹出如图 3-1 所示系统属性对话框，选择“高级”选项卡，然后单击“环境变量”按钮，弹出如图 3-2 所示环境变量设置对话框。

---

① Python(英国发音：/'paiθən/，美国发音：/'paiθɑn/)，是一种面向对象、直译式计算机编程语言，具有近二十年的发展历史，成熟且稳定。它包含了一组完善而且容易理解的标准库，能够轻松完成很多常见的任务。它的语法简洁而清晰，尽量使用无异议的英语单词，与其他大多数程序设计语言使用大括号不一样，它使用缩进来定义语句块。——引自于维基百科 http://zh.wikipedia.org/wiki/Python。

图 3-1　系统属性对话框

图 3-2　环境变量设置对话框

在图 3-2 所示环境变量设置对话框中需要设置 Path 变量，然而 Path 变量在用户变量(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)都有，读者可以根据自己的喜好自己决定在哪里设置。本例中是在系统变量中设置的，双击系统变量中 Path 变量，弹出如图 3-3 所示对话框，在 Path 后面追加 C:\Python27，注意两个 Path 之间要用英文分号“;”分隔。

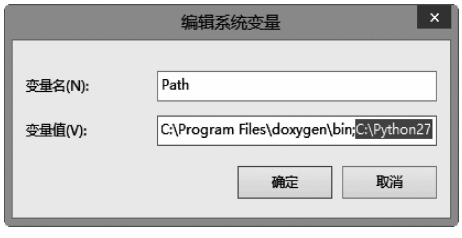


图 3-3　设置 Path 变量

配置好环境后，能够通过 DOS 等终端进入 cocos 目录(<Cocos2d-x 安装目录>\tools\cocos2d-console\bin)，然后在 DOS 等终端中执行如下指令：

```
cocos new HelloWorld -p com.work6 -l cpp -d D:/projects
```

其中，D:/projects 为 HelloWorld 的工程生成目录。通过上面的指令在 D:/projects 目录下面生成了名为 HelloWorld 的 Cocos2d-x 工程。

打开 HelloWorld 目录，其中的内容如图 3-4 所示。

从图 3-4 中可以看出生成的工程代码是适合于多平台的，其中 Classes 目录是放置一些通用类(与平台无关的)，我们编写的 C++代码主要放置在该目录下。图 3-4 中 cocos2d 目录是放置 Cocos2d-x 引擎的源代码，其中包括音效引擎和物理引擎等。

图 3-4 中 proj.android、proj.ios_mac、proj.win32、proj.wp8-xaml 和 proj.linux 目录是

图 3-4 HelloWorld 工程中的内容

放置与特定平台有关系的代码。其中，proj. android 是 android 平台特定代码；proj. ios_mac 是 iOS 和 Mac OS 运行需要的特定代码；proj. win32 是 Win32 平台运行需要的特定代码，它可以在 Windows 下运行，模拟器是 Win32 窗口；proj. wp8-xaml 是 Windows Phone 8 平台运行需要的特定代码；proj. linux 是 Linux 平台运行需要的特定代码。

考虑到广大读者对 Windows 比较熟悉，而且学习容易上手，不需要更多的设备投入，所以本书介绍平台移植（第 21 章）之前的实例主要都是基于 Win32 平台的，可以通过 proj. win32 目录下的 Visual Studio 解决方案 HelloWorld. sln 来进行编译和运行。

图 3-4 中 Resources 目录是放置工程需要的资源文件，这个目录中的内容是共享于全部平台下的。

3.1.2 工程文件结构

进入到 proj. win32 目录下，双击 HelloWorld. sln 解决方案文件，启动 HelloWorld 界面，如图 3-5 所示。

图 3-5 在 Visual Studio 中启动 HelloWorld. sln 解决方案

图 3-5 所示解决方案中 HelloWorld 工程的 Classes 文件夹中的内容是与图 3-4 所示 Classes 目录的内容对应的。HelloWorld 工程的 win32 文件夹中的 main. cpp 和 main. h 是 win32 平台特有程序代码，通过它启动 Win32 窗口。

图 3-5 中的 libAudio 工程对应于图 3-4 的 cocos2d 目录中的音效引擎，libchipmunk 工程是物理引擎，libcocos2d 工程是 Cocos2d-x 引擎。

如果想看一下效果，可以单击 ▶ 本地 Windows 调试器 按钮，运行结果如图 3-6 所示。

图 3-6　程序运行结果

3.1.3　代码解释

下面解释一下图 3-5 中的 Classes 文件夹的 4 个文件：AppDelegate. h、AppDelegate. cpp、HelloWorldScene. h 和 HelloWorldScene. cpp。

1. AppDelegate 类

在 AppDelegate. h 和 AppDelegate. cpp 中分别声明和定义了 AppDelegate 类，AppDelegate 类是 Cocos2d-x 引擎要求实现的游戏应用委托对象，在 Cocos2d-x 游戏运行的不同生命周期阶段会触发它的不同函数。

AppDelegate. h 代码如下：

```
#ifndef _APP_DELEGATE_H_
#define _APP_DELEGATE_H_

#include "cocos2d.h"

class AppDelegate : private cocos2d::Application
{
public:
    AppDelegate();
    virtual ~AppDelegate();
```

```
    /*
        * 游戏启动时调用的函数,在这里可以初始化导演对象和场景对象
     */
    virtual bool applicationDidFinishLaunching();

    /*
        * 游戏进入后台时调用的函数
     */
    virtual void applicationDidEnterBackground();

    /*
        * 游戏进入前台时调用的函数
     */
    virtual void applicationWillEnterForeground();
};

#endif // _APP_DELEGATE_H_
```

从上面的代码看到 AppDelegate 继承了 cocos2d::Application，cocos2d::Application 是 Cocos2d-x 引擎提供的基类。

AppDelegate.cpp 代码如下：

```
#include "AppDelegate.h"
#include "HelloWorldScene.h"

USING_NS_CC;                                                    ①

AppDelegate::AppDelegate()
{
}
AppDelegate::~AppDelegate()
{
}

bool AppDelegate::applicationDidFinishLaunching() {             ②
    //初始化 director
    auto director = Director::getInstance();                    ③
    auto glview = director->getOpenGLView();
    if(!glview) {
        glview = GLView::create("My Game");
        director->setOpenGLView(glview);                        ④
    }
    director->setDisplayStats(false);                           ⑤
    director->setAnimationInterval(1.0 / 60);                   ⑥

    auto scene = HelloWorld::createScene();                     ⑦
```

```
    //run
    director->runWithScene(scene);                                          ⑧

    return true;
}
void AppDelegate::applicationDidEnterBackground() {                         ⑨
    Director::getInstance()->stopAnimation();                               ⑩
    //SimpleAudioEngine::sharedEngine()->pauseBackgroundMusic();            ⑪
}

void AppDelegate::applicationWillEnterForeground() {                        ⑫
    Director::getInstance()->startAnimation();                              ⑬
    //SimpleAudioEngine::sharedEngine()->resumeBackgroundMusic();           ⑭
}
```

在上述代码第①行的 USING_NS_CC 是 Cocos2d-x 提供的一个宏，它是用来替换 using namespace cocos2d 语句的，随着学习的深入会发现 Cocos2d-x 定义了很多宏，使用起来很方便。

代码第②行定义 applicationDidEnterBackground()函数是游戏程序进入到后台时调用的函数。第③行代码是初始化导演类 Director，第④行代码是设置导演类的 OpenGL 视图。第⑤行代码是设置是否在屏幕上显示帧率等信息。显示帧率一般是为了测试，实际发布时是设置为不显示的，它会影响游戏的外观。第⑥行代码是设定计时器 1.0/60 秒间隔一次，即设定帧率为 60。第⑦行代码创建场景对象 Scene，第⑧行代码是运行该场景，这会使游戏进入该场景。

代码第⑨行定义的 applicationDidEnterBackground()函数是游戏进入后台时调用的函数。第⑩行代码是停止场景中的动画。第⑪行代码是停止背景音乐，默认是注释掉的。

代码第⑫行定义的 applicationWillEnterForeground()函数是游戏进入前台时调用的函数。第⑬行代码是开始场景中的动画。第⑭行代码是继续背景音乐，默认是注释掉的。

2. HelloWorld 类

在 HelloWorldScene.h 和 HelloWorldScene.cpp 中分别声明和定义了 HelloWorld 类，HelloWorld 类继承了 cocos2d::Layer 类，它被称为层(layer)，这些层被放到场景(scene)中，场景类是 cocos2d::Scene。注意，HelloWorldScene.h 虽然命名为场景，但是它内部定义的 HelloWorld 类是一个层。

HelloWorldScene.h 的代码如下：

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__

#include "cocos2d.h"

class HelloWorld : public cocos2d::Layer                                    ①
{
```

```
public:

    static cocos2d::Scene * createScene();                                    ②
    virtual bool init();                                                      ③

    void menuCloseCallback(cocos2d::Ref * pSender);                           ④

    CREATE_FUNC(HelloWorld);                                                  ⑤
};

#endif // __HELLOWORLD_SCENE_H__
```

从上面的代码第①行可以看出 HelloWorld 继承了 cocos2d::Layer，HelloWorld 是一个层，而不是场景。第②行代码是声明创建当前层 HelloWorld 所在场景的静态函数 createScene()。第③行代码是声明初始化层 HelloWorld 实例函数。第④行代码是声明菜单回调函数 menuCloseCallback，用于触摸菜单事件的回调。第⑤行代码中的 CREATE_FUNC 是一个 Cocos2d-x 定义的宏，它的作用相当于如下代码：

```
static HelloWorld * create()
{
    HelloWorld * pRet = new HelloWorld();
    if (pRet && pRet->init())
    {
        pRet->autorelease();
        return pRet;
    }
    else
    {
        delete pRet;
        pRet = NULL;
        return NULL;
    }
}
```

从上述代码可见 CREATE_FUNC 宏的作用是创建一个静态函数 create，该函数可以用来创建层。

下面分别解释一下 HelloWorldScene.cpp 中的几个函数。

HelloWorldScene.cpp 中 createScene()代码如下：

```
Scene * HelloWorld::createScene()
{
    auto scene = Scene::create();                                             ①
    auto layer = HelloWorld::create();                                        ②
    scene->addChild(layer);                                                   ③
    return scene;
}
```

createScene()函数是在游戏应用启动的时候,在AppDelegate的applicationDidFinishLaunching()函数中通过auto scene = HelloWorld::createScene()语句调用的。createScene()中做了三件事情,首先创建了HelloWorld层所在的场景对象(见代码第①行),其次创建了HelloWorld层(见代码第②行),最后将HelloWorld层添加到场景scene中(见代码第③行)。

当调用HelloWorld::create()语句创建层的时候,会调用HelloWorld的实例函数init(),达到初始化HelloWorld层的目的。init ()代码如下:

```
bool HelloWorld::init()
{
    //////////////////////////////
    // 1. 初始化父类
    if (!Layer::init())                                                          ①
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();               ②
    Vec2 origin = Director::getInstance()->getVisibleOrigin();                  ③

    //////////////////////////////
    // 2. 增加一个菜单项,单击它的时候退出程序
    auto closeItem = MenuItemImage::create(
                                        "CloseNormal.png",
                                        "CloseSelected.png",
                                CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));   ④

    closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize().
width/2 ,origin.y + closeItem->getContentSize().height/2));                      ⑤
    auto menu = Menu::create(closeItem, NULL);                                   ⑥
    menu->setPosition(Vec2::ZERO);                                               ⑦
    this->addChild(menu, 1);                                                     ⑧

    //////////////////////////////
    // 3. 在下面添加自己的代码

    auto label = LabelTTF::create("Hello World", "Arial", 24);                   ⑨
    label->setPosition(Vec2(origin.x + visibleSize.width/2,
                            origin.y + visibleSize.height - label->getContentSize().height)); ⑩
    this->addChild(label, 1);                                                    ⑪

    auto sprite = Sprite::create("HelloWorld.png");                              ⑫
    sprite->setPosition(Vec2(visibleSize.width/2 + origin.x, visibleSize.height/2 + origin.y)); ⑬
    this->addChild(sprite, 0);                                                   ⑭

    return true;
}
```

在 init()函数中主要是初始化 HelloWorld 层,其中包括了里面的精灵、菜单和文字等内容。其中,第①代码是初始化父类 Layer,返回 true 则初始化成功,返回 false 则初始化失败。

第②行代码是定义视图的可视化尺寸,第③行代码是定义视图的可视化原点。

第④行代码是创建一个图片菜单项对象,单击该菜单项的时候回调 menuCloseCallback 函数。第⑤行代码是菜单项对象的位置,第⑥行代码是创建 Menu 菜单对象。第⑦行代码是定义菜单对象的位置。第⑧行代码是把菜单对象添加到当前 HelloWorld 层上,如图 3-6 所示右下角的⏻就是刚刚添加的菜单对象。

第⑨~⑪行代码是将一个 Hello World 标签对象放置到层中。这个过程是:创建对象→设置对象的位置→把对象添加到层上。第⑨行代码创建一个 LabelTTF 标签对象。第⑩行代码是设置标签对象位置为水平居中,在垂直方向上与屏幕顶对齐。第⑪行代码将文本对象添加到层 HelloWorld 上。

第⑫行代码是创建精灵 Sprite 对象,它是图 3-6 中的 Cocos2d-x 的 logo 图标。第⑬行代码是设置精灵对象的位置,这个位置是屏幕的中央。第⑭行代码是将精灵对象添加到层 HelloWorld 上。

菜单回调函数 menuCloseCallback 代码如下:

```
void HelloWorld::menuCloseCallback(Ref * pSender)
{
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WP8) || (CC_TARGET_PLATFORM == CC_PLATFORM_WINRT)    ①
    MessageBox("You pressed the close button. Windows Store Apps do not implement a close
button.","Alert");
    return;
#endif

    Director::getInstance()->end();

#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)                                                  ②
    exit(0);
#endif
}
```

menuCloseCallback 函数中使用了条件编译语句代码①和②,用来判断是哪个当前程序,运行的是哪个平台。其中,CC_TARGET_PLATFORM 读取当前运行平台的宏,CC_PLATFORM_WP8 表示 Windows Phone 8 平台;CC_PLATFORM_WINRT 表示 Windows Runtime[①] 平台;CC_PLATFORM_IOS 表示 iOS 平台。此外,还有定义很多平台的宏,如果需要则可以查询 API。

① Windows Runtime 或 WinRT,是 Windows 8 中的一种跨平台应用程序架构。

3.2 Cocos2d-x 核心概念

Cocos2d-x 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，当然也有传统的游戏的概念。Cocos2d-x 中核心概念：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、菜单概念以及对应的类，由于节点概念很重要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。

3.2.1 导演

导演类 Director(v3.0 之前是 CCDirector)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：

```
auto director = Director::getInstance();
```

导演对象职责如下：

(1) 访问和改变场景。

(2) 访问 Cocos2d-x 的配置信息。

(3) 暂停、继续和停止游戏。

(4) 转换坐标。

Director 类图如图 3-7 所示。

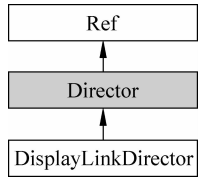


图 3-7 Director 类图运行结果

从图 3-4 中还可以看到，它有一个子类是 DisplayLinkDirector。

3.2.2 场景

场景类 Scene(v3.0 之前是 CCScene)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。包括主菜单、设置游戏参数等。

(3) 游戏场景。这是游戏的主要内容。

场景类 Scene 的类图如图 3-8 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。

3.2.3 层

层是写游戏的重点，开发者大约 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 3-9 所示，一个简单的主菜单界

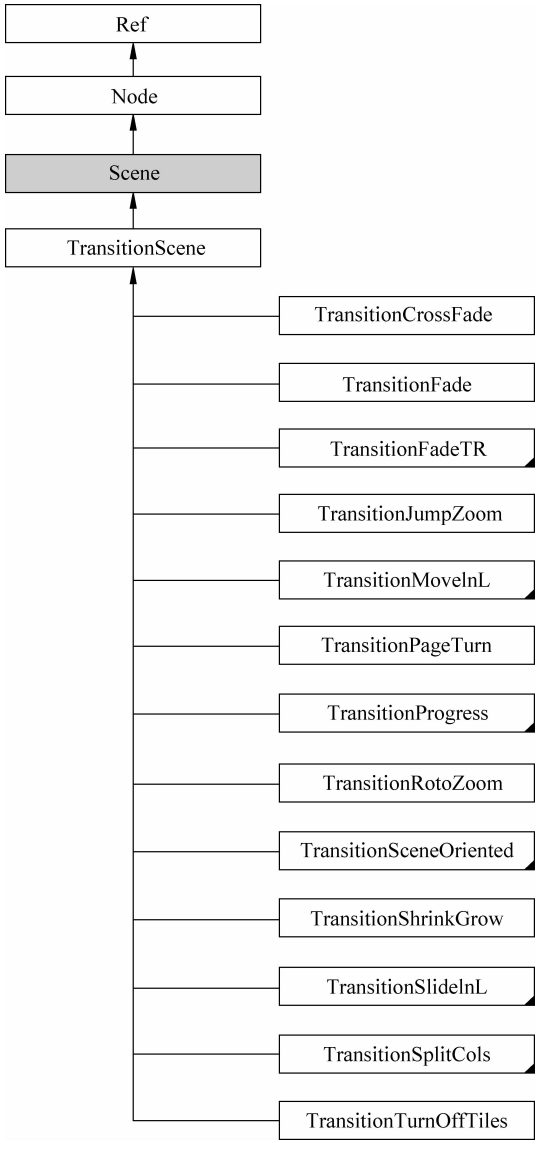
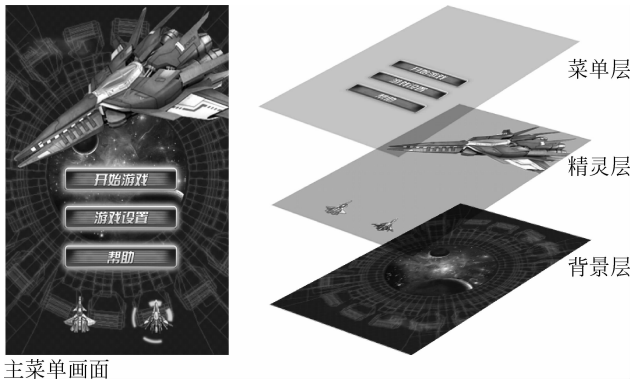


图 3-8 Scene 类图

面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，从上到下依次是：菜单层→精灵层→背景层。Cocos2d-x 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：文本、链接、精灵、



主菜单画面

图 3-9　层叠加

地图等内容。

层类 Layer 的类图如图 3-10 所示。

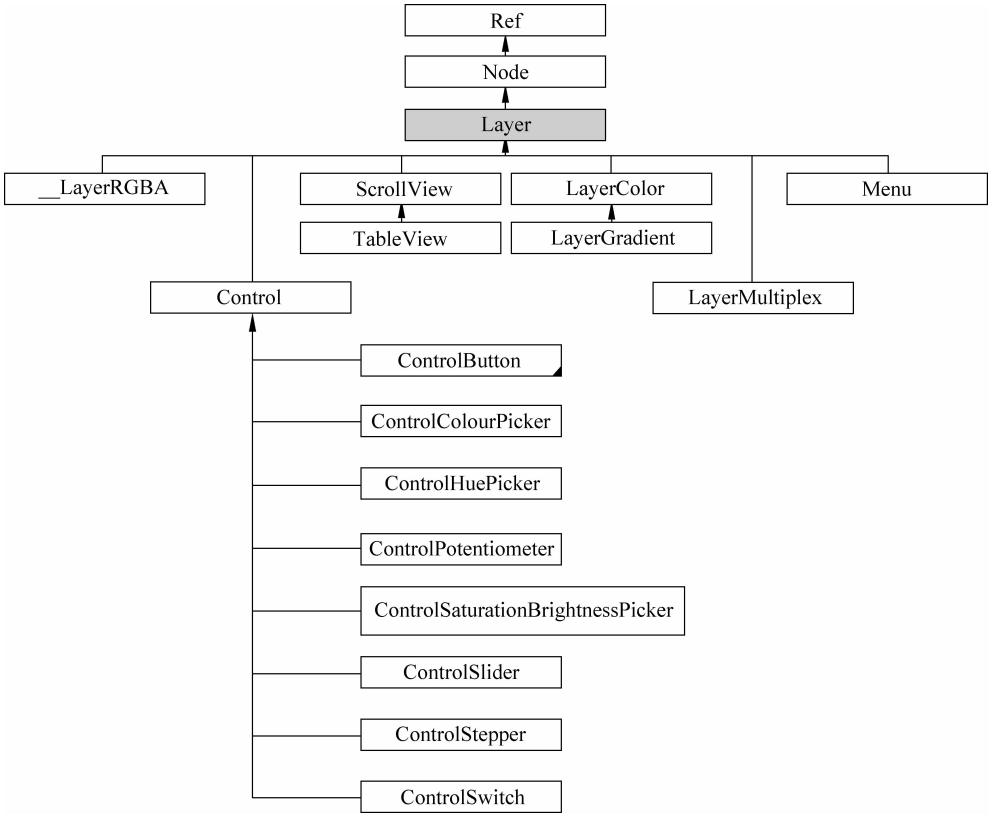


图 3-10　Layer 类图

3.2.4 精灵

精灵类 Sprite（v3.0 之前是 CCSprite）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。

Sprite 类图如图 3-11 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。

3.2.5 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 中菜单类是 Menu，Menu 类图如图 3-12 所示。从类图可见，Menu 类派生于 Layer。

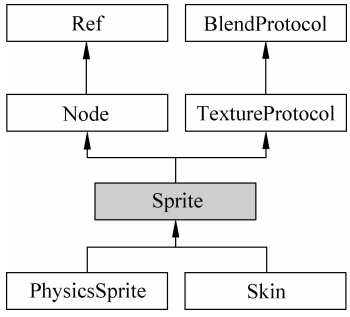


图 3-11 Sprite 类图

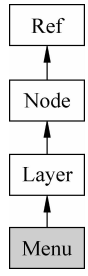


图 3-12 Menu 类图

在菜单中又包含了菜单项 MenuItem，MenuItem 类图如图 3-13 所示。从图中可见，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。每个菜单项都有三个基本状态：正常、选中和禁止。

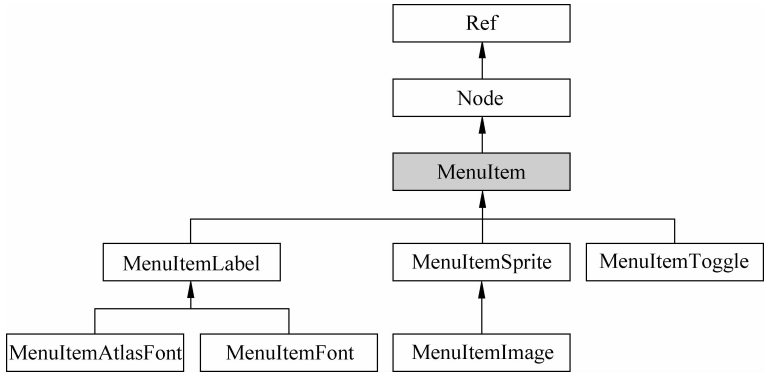


图 3-13 MenuItem 类图

3.3 Node 与 Node 层级架构

Cocos2d-x 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 3-14 所示。

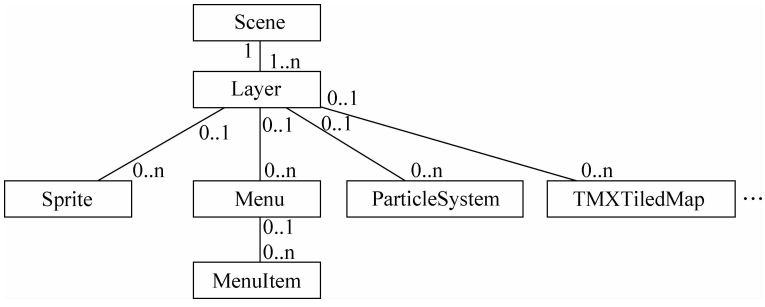


图 3-14 节点的层级结构

这些节点有一个共同的父类 Node,Node 类图如图 3-15 所示。Node 类是 Cocos2d-x 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。

3.3.1 Node 中重要的操作

Node 作为根类它有很多重要的函数,下面分别介绍一下:

(1) 创建节点。

```
Node * childNode = Node::create()
```

(2) 增加新的子节点。

```
node->addChild(childNode, 0, 123)
```

(3) 查找子节点。

```
Node * node = node->getChildByTag(123)
```

(4) 删除子节点,并停止该节点上的一切动作。

```
node->removeChildByTag(123, true)
```

(5) 通过 Node 指针删除节点。

```
node->removeChild()
```

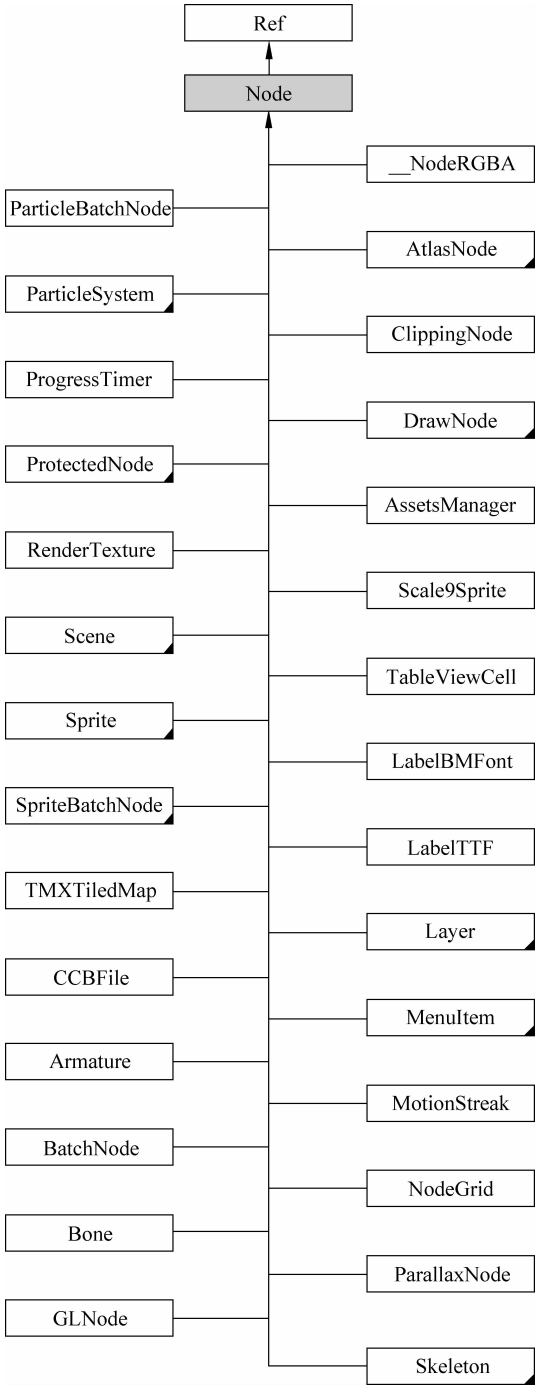

Ref
Node
__NodeRGBA
ParticleBatchNode
AtlasNode
ParticleSystem
ClippingNode
ProgressTimer
DrawNode
ProtectedNode
AssetsManager
RenderTexture
Scale9Sprite
Scene
TableViewCell
Sprite
LabelBMFont
SpriteBatchNode
LabelTTF
TMXTiledMap
Layer
CCBFile
MenuItem
Armature
MotionStreak
BatchNode
NodeGrid
Bone
ParallaxNode
GLNode
Skeleton

图 3-15 Node 类图

(6) 删除所有子节点,并停止这些子节点上的一切动作。

```
node->removeAllChildrenWithCleanup(true)
```

(7) 从父节点中删除 node 节点,并停止该节点上的一切动作。

```
node->removeFromParentAndCleanup(true)
```

3.3.2 Node 中重要的属性

Node 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 3-16 所示是 anchorPoint 为(0.5,0.5)的情况,这是默认情况。

(2) 图 3-17 所示是 anchorPoint 为(0.0,0.0)的情况。

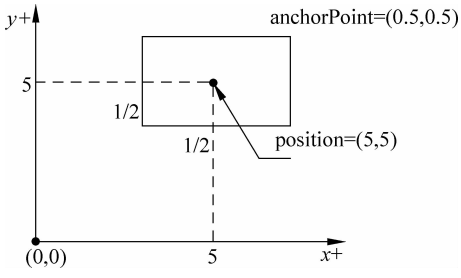


图 3-16 anchorPoint 为(0.5,0.5)

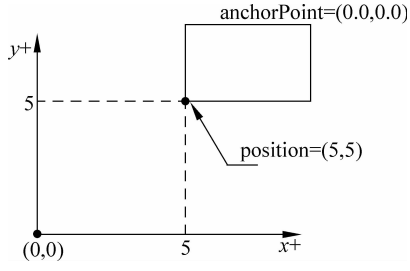


图 3-17 anchorPoint 为(0.0,0.0)

(3) 图 3-18 所示是 anchorPoint 为(1.0,1.0)的情况。

(4) 图 3-19 所示是 anchorPoint 为(0.66, 0.5)的情况。

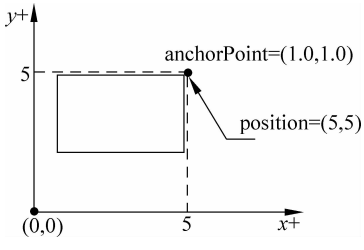


图 3-18 anchorPoint 为(1.0,1.0)

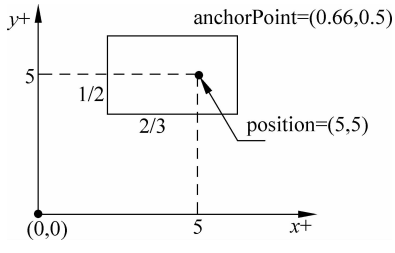


图 3-19 anchorPoint 为(0.66, 0.5)

为了进一步了解 anchorPoint 的使用,可修改 HelloWorld 实例。修改 HelloWorldScene.cpp 的 HelloWorld::init()函数如下,其中加黑体字体显示的是添加的代码。

```
bool HelloWorld::init()
{
    ...

    auto label = LabelTTF::create("Hello World", "Arial", 24);
    label->setPosition(Point(origin.x + visibleSize.width/2,
                             origin.y + visibleSize.height - label->getContentSize().height));

    label->setAnchorPoint(Vec2(1.0, 1.0));

    this->addChild(label, 1);

    auto sprite = Sprite::create("HelloWorld.png");
    sprite->setPosition(Vec2(visibleSize.width/2 + origin.x, visibleSize.height/2 +
origin.y));
    this->addChild(sprite, 0);

    return true;
}
```

运行结果如图 3-20 所示，Hello World 设置了 anchorPoint 为(1.0,1.0)。

图 3-20　Hello World 的 anchorPoint 为(1.0,1.0)

3.3.3 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与维护。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(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 中调用函数。

Node 中调用函数主要有:

(1) void scheduleUpdate(void)。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自己的 update(float dt)函数。

(2) void schedule(SEL_SCHEDULE selector, float interval)。与 scheduleUpdate 函数功能一样,不同的是可以指定回调函数(通过 selector 指定),也可以更改需要指定回调时间间隔。

(3) void unscheduleUpdate(void)。停止 update(float dt)函数调度。

(4) void unschedule(SEL_SCHEDULE selector)。可以指定具体函数停止调度。

(5) void unscheduleAllSelectors(void)。可以停止调度。

为了进一步了解游戏循环与调度的使用,可修改 HelloWorld 实例。

修改 HelloWorldScene.h 代码,添加 update(float dt)声明。代码如下:

```
class HelloWorld : public cocos2d::Layer
{
public:
    …

    virtual void update(float dt);

    CREATE_FUNC(HelloWorld);

};
```

修改 HelloWorldScene.cpp 代码如下:

```
bool HelloWorld::init()
{
    …

    auto label = LabelTTF::create("Hello World", "Arial", 24);
    label->setTag(123);                                                    ①
    …

    //更新函数
    this->scheduleUpdate();                                                ②
    //this->schedule(schedule_selector(HelloWorld::update),1.0f/60);       ③

    return true;
```

```
}

void HelloWorld::update(float dt)                                                    ④
{
    auto label = this->getChildByTag(123);                                           ⑤
    label->setPosition(label->getPosition() + Vec2(2, -2));                          ⑥
}

void HelloWorld::menuCloseCallback(Ref* pSender)
{
    //停止更新
    unscheduleUpdate();                                                              ⑦
    Director::getInstance()->end();

    #if (CC_TARGET_PLATFORM == CC_PLATFORM_WP8) || (CC_TARGET_PLATFORM == CC_PLATFORM_WINRT)
    MessageBox("You pressed the close button. Windows Store Apps do not implement a close
button.","Alert");
    return;
#endif

    Director::getInstance()->end();

#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    exit(0);
#endif

}
```

为了能够在 init 函数之外访问标签对象 label,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 init 函数中调用 scheduleUpdate(见第②行代码)或 schedule(见第③行代码)。

代码第④行的 HelloWorld::update(float dt)函数是在调度函数,精灵等对象的变化逻辑都是在这个函数中编写的。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。第⑦行代码 unscheduleUpdate()就是停止调度 update,如果是其他的调度函数则可以采用 unschedule 或 unscheduleAllSelectors 停止。

3.4 Cocos2d-x 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它的原点是在左上角的。而在 Cocos2d-x 坐标系中其原点是在

左下角的，而且 Cocos2d-x 坐标系又可以分为世界坐标和模型坐标。

3.4.1 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。它的原点是在左上角的，如图 3-21 所示。

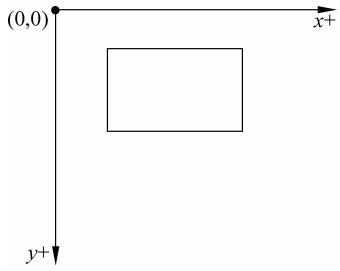


图 3-21 UI 坐标

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、控件等都是遵守这个坐标系。然而在 Cocos2d-x 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函数，如下面代码所示：

```
Vec2 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 默认坐标，可以采用下面的语句进行转换：

```
Vec2 touchLocation2 = 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 默认坐标。

3.4.2 OpenGL 坐标

在上面提到了 OpenGL 坐标，OpenGL 坐标是三维坐标。由于 Cocos2d-x 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，如图 3-22 所示。

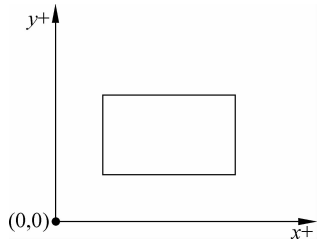


图 3-22 OpenGL 坐标

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 3-23(a)所示。左手坐标是 z 轴指向屏幕里，如图 3-23(b)所示。OpenGL 坐标是右手坐标，而微软平台的 Direct3D[①] 是左手坐标。

3.4.3 世界坐标和模型坐标

由于 OpenGL 坐标可以分为世界坐标和模型坐标，所以 Cocos2d-x 的坐标也有世界坐

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之一。——引自于维基百科 http://zh.wikipedia.org/wiki/Direct3D。

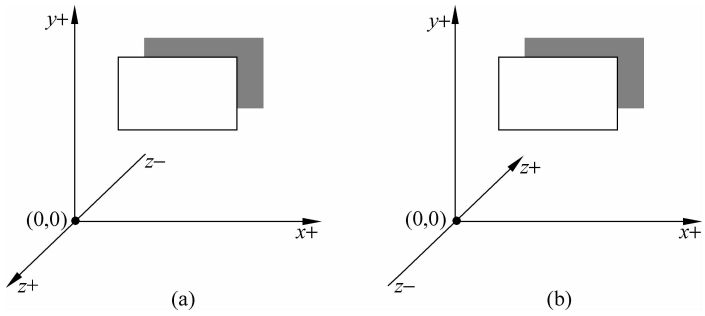


图 3-23 三维坐标

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。

看看图 3-24，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)。

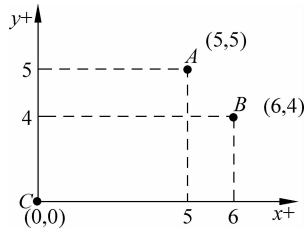


图 3-24 世界坐标和模型坐标

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

(1) Vec2 convertToNodeSpace(const Vec2& worldPoint)。将世界坐标转换为模型坐标。

(2) Vec2 convertToNodeSpaceAR(const Vec2& worldPoint)。将世界坐标转换为模型坐标。AR 表示相对于锚点。

(3) Vec2 convertTouchToNodeSpace(Touch * touch)。将世界坐标中触摸点转换为模型坐标。

(4) Vec2 convertTouchToNodeSpaceAR(Touch * 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) Vec2 convertToWorldSpace(const Vec2& nodePoint)。将模型坐标转换为世界坐标。

(6) Vec2 convertToWorldSpaceAR(const Vec2& 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转换。

1. 世界坐标转换为模型坐标

图 3-25 所示是世界坐标转换为模型坐标实例运行结果。

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。

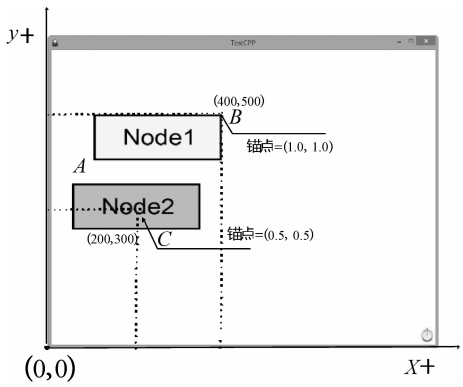


图 3-25　世界坐标转换为模型坐标

编写代码如下：

```
bool HelloWorld::init()
{

    if (!Layer::init())
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    auto closeItem = MenuItemImage::create(
        "CloseNormal.png",
        "CloseSelected.png",
        CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

    closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize()
.width/2 ,origin.y + closeItem->getContentSize().height/2));

    auto menu = Menu::create(closeItem, NULL);
    menu->setPosition(Vec2::ZERO);
    this->addChild(menu, 1);
    //创建背景
    auto bg = Sprite::create("bg.png");                                    ①
    bg->setPosition(Vec2(origin.x + visibleSize.width/2,
        origin.y + visibleSize.height/2));

    this->addChild(bg, 0);                                                 ②
    //创建 Node1
    auto node1 = Sprite::create("node1.png");                              ③
    node1->setPosition(Vec2(400,500));
    node1->setAnchorPoint(Vec2(1.0, 1.0));

    this->addChild(node1, 0);                                              ④
    //创建 Node2
    auto node2 = Sprite::create("node2.png");                              ⑤
    node2->setPosition(Vec2(200,300));
    node2->setAnchorPoint(Vec2(0.5, 0.5));
```

```
    this->addChild(node2, 0);                                                    ⑥

    Vec2 point1 = node1->convertToNodeSpace(node2->getPosition());               ⑦
    Vec2 point3 = node1->convertToNodeSpaceAR(node2->getPosition());             ⑧

    log("Node2 NodeSpace = (%f, %f)",point1.x,point1.y);
    log("Node2 NodeSpaceAR = (%f, %f)", 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：

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

结合图 3-25 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 3-25 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

图 3-26 所示是模型坐标转换为世界坐标实例运行结果。

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。

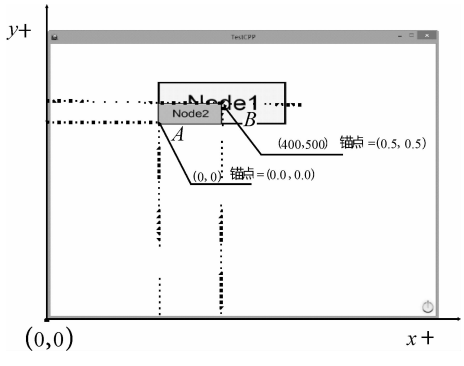


图 3-26　模型坐标转换为世界坐标

编写代码如下：

```
bool HelloWorld::init()
{
    if (!Layer::init())
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
```

```
    auto closeItem = MenuItemImage::create(
        "CloseNormal.png",
        "CloseSelected.png",
        CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

    closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize()
.width/2,origin.y + closeItem->getContentSize().height/2));

    auto menu = Menu::create(closeItem, NULL);
    menu->setPosition(Vec2::ZERO);
    this->addChild(menu, 1);

    //创建背景
    auto bg = Sprite::create("bg.png");
    bg->setPosition(Vec2(origin.x + visibleSize.width/2,
        origin.y + visibleSize.height/2));
    this->addChild(bg, 0);

    //创建 Node1
    auto node1 = Sprite::create("node1.png");
    node1->setPosition(Vec2(400,500));
    this->addChild(node1, 0);

    //创建 Node2
    auto node2 = Sprite::create("node2.png");
    node2->setPosition(Vec2(0.0, 0.0));                                    ①
    node2->setAnchorPoint(Vec2(0.0, 0.0));                                 ②
    node1->addChild(node2, 0);                                             ③

    Vec2 point2 = node1->convertToWorldSpace(node2->getPosition());        ④
    Vec2 point4 = node1->convertToWorldSpaceAR(node2->getPosition());      ⑤
    log("Node2 WorldSpace = (%f,%f)",point2.x,point2.y);
    log("Node2 WorldSpaceAR = (%f,%f)",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:

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

图 3-23 所示的位置，可以用世界坐标描述。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);
```

3.5 Win32 平台下设置屏幕

在 Win32 平台运行游戏是为了进行测试游戏应用，那么如何改变屏幕的大小呢？修改屏幕大小代码如下：

```
bool AppDelegate::applicationDidFinishLaunching() {
    // initialize director
    auto director = Director::getInstance();
    auto glview = director->getOpenGLView();
    if(!glview) {
        glview = GLView::create("My Game");
        glview->setFrameSize(900, 640);                    ①
        director->setOpenGLView(glview);
    }
        ...
    return true;
}
```

在 applicationDidFinishLaunching()函数第①行代码 glview->setFrameSize(900, 640)可以设置窗口的大小，如果是竖屏显示的，可以设置 glview->setFrameSize(640,900)。

本章小结

通过对本章的学习，可以了解 Cocos2d-x 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，学习的重点是 Node 和 Node 层级架构。本章最后介绍了 Cocos2d-x 的坐标系和 Win32 平台下设置屏幕显示。