第二部分

基 础 篇

本部分内容

- 第 3 章 Cocos2D-x 中的核心类
- 第 4 章 Cocos2D-x 中的动作、特效与动画
- 第 5 章 Cocos2D-x 中的菜单项和文本渲染系统
- 第 6 章 Cocos2D-x 中的事件处理机制
- 第 7 章 Cocos2D-x 中的瓦片地图集
- 第 8 章 Cocos2D-x 中的声音、存储和网络
- 第 9 章 可视化场景编辑器
- 第 10 章 Cocos2D-x 中的物理引擎

第 3 章　Cocos2D–x 中的核心类

Cocos2D-x 引擎的设计思路是将游戏的各个部分抽象成几个概念，包括导演、场景、布景层和人物精灵，它们之间的关系如图 3-1 所示。

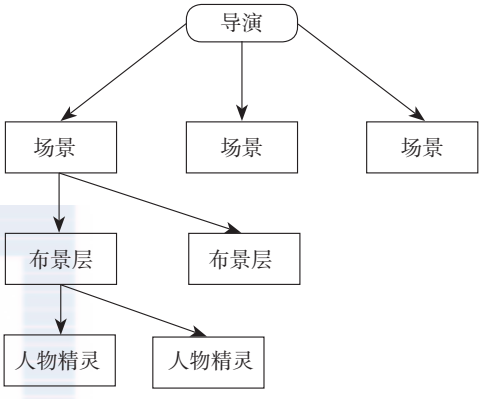

图 3-1　Cocos2D-x 的基本部分间的关系图

几乎任何一款游戏中都会有这些概念，而游戏的复杂程度也就决定这些部分之间的关系的复杂程度。具体说明如下：

- ❑ 导演（CCDirector）：顾名思义，导演类是游戏中的组织者和领导者，是整个游戏的负责人、总指挥。导演类可以制定游戏的运行规则，从而让游戏内的场景、布景类和精灵类有序地进行。
- ❑ 场景（CCScene）：场景就是一个关卡，或者是一个游戏界面。这样的一个一个场景确定了整个的游戏。
- ❑ 布景层（CCLayer）：一个场景可以由多个布景层构成。布景层就是关卡里的背景，关卡不同也就是场景需要的布景层不同。有时候，为了游戏的不同模块的管理更加方便，会把一个场景分为多个布景层，如 UI 布景层、游戏布景层；有些游戏需要更细致的细分，可以分为游戏对象布景层和游戏地图布景图。
- ❑ 人物精灵（CCSprite）：人物精灵可以分为玩家控制的主角类、敌人类；更复杂的游戏可以分为 NPC（Non-Player-Controlled Character，非玩家控制角色）类、机关类等。它们是构成游戏的关键要素。

本章主要介绍这些 Cocos2D-x 基础类，首先，节点类是场景、布景层、人物精灵的基类，然后分别讲解导演、场景、布景层、人物精灵等。另外由于 Cocos2D-x 由 Cocos2D-iPhone 移植而来，很多功能和 Objective-C 相关，所以移植到 C++ 版本时，涉及一些 Objective-C 的类别和特性的移植。对于没有使用或学习过 Objective-C 的开发者来说，这些会有些陌生。本章将在后面介绍这些内容，从而为接下来的学习打下基础。

3.1　节点类

节点类（CCNode）是 Cocos2D-x 中的主要类，继承自 CCObject，继承关系如图 3-2 所示。

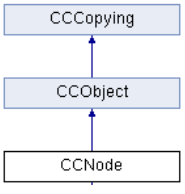

图 3-2　CCNode 类的继承关系

任何需要画在屏幕上的对象都是节点类。最常用的节点类包括场景

类（CCScene）、布景层类（CCLayer）、人物精灵类（CCSprite）、菜单类（CCMenu）。

CCNode 类包括的主要功能如下：

- ❑ 每个节点都可以含有子节点，这点本书也会在后面给出示例。
- ❑ 节点含有周期性回调的方法（Schedule、Unschedule 等）。关于周期性回调方法，本章将会有一节单独讲解，如果现在不了解，可以跳过这段内容。
- ❑ 可以含有动作（CCAction）。

CCNode 可以为它自己和它的子节点添加额外的功能。无论是 CCNode 运行的动作（CCAction），还是设置的旋转角度和位置等属性，父节点的设置都可以传递到子节点上，这点在一些游戏的开发中可以使我们的管理更轻松。比如某些纵版射击游戏，玩家控制的主角飞机需要携带子机，子机的移动位置要随着主机一起移动，我们就可以把子机设置为主机的子节点，这样，在设置位置的时候，只需要设置主机的位置就可以了，大大减少了程序员需要处理的内容，提高了代码的清晰度和可读性。

由于 CCNode 类不自带贴图，其实在屏幕上看不到任何节点类的效果，所以一般使用 CCNode 类的场合有两个：第一个情况就是，需要一个父节点来管理一批子节点，这时候可以设置一个“无形”的子节点来管理子节点；另一种情况就是有时需要自己定义一个在屏幕上显示的对象，这时候让新定义的这个类继承自 CCNode。一个类继承自 CCNode 类，说明它有如下特点：

- ❑ 重写初始化的方法和周期性回调方法。
- ❑ 在时间线上控制回调。
- ❑ 重写渲染的绘制方法。

CCNode 类不含有贴图，它可以进行位置的平移、大小的伸缩变化、旋转变化。在网格特效（会在后面的章节介绍）使用的时候，网格特效可以获得屏幕中绘制的内容，并且对获得的屏幕内容进行渲染。这点在游戏中需要一些全屏特效的时候可以使用。

下面分别介绍 CCNode 类的成员数据和函数。

3.1.1 CCNode 类的成员数据

CCNode 类的主要公共成员数据，如表 3-1 所示。

表 3-1 CCNode 类的主要公共成员数据

名　称	类　型	描　述
m_bIsRunning	布尔型	这个节点是否在运行
m_bIgnoreAnchorPointForPosition	布尔型	是否忽略锚点的位置

CCNode 类的主要保护成员数据（子类可见，Protected 关键字）如表 3-2 所示。

表 3-2 CCNode 类的主要保护成员数据

名 称	类 型	描 述
m_nZOrder	整型	该节点兄弟节点的 z 轴顺序。在二维游戏中，z 轴顺序决定遮挡关系
m_fVertexZ	浮点型	在 OpenGL 的真正的 z 轴值
m_fRotation	浮点型	角度制的节点旋转的角度值
m_fScaleX	浮点值	x 轴的缩放系数
m_fScaleY	浮点值	y 轴的缩放系数
m_tPosition	点坐标（CCPoint）	位置坐标
m_fSkewX	浮点值	x 轴的扭曲效果的系数
m_fSkewY	浮点值	y 轴的扭曲效果的系数
m_pChildren	数组	子节点数组
m_pCamera	摄像机	跟随节点的摄像机对象
m_pGrid	网格特效	允许节点拥有网格特效
m_bIsVisible	布尔值	节点是否显示
m_tAnchorPoint	点坐标	节点平移或移位时的锚点，（0，0）为左下，（1，1）为右上，（0.5，0.5）为中心
m_tAnchorPointInPoints	点坐标	节点平移或移位时的锚点的绝对坐标，可读，如果需要修改，请修改上一个属性
m_tContentSize	尺寸（CCSize）	获得节点的大小（未缩放、旋转等），所有节点都有大小，场景类和布景层类的尺寸是屏幕大小
m_tPosition	点坐标	节点坐标
m_pParent	节点	父节点
m_nTag	整型	节点标记
m_pUserData	空	用户数据指针
m_pUserObject	对象（CCObject）	类似上一个属性，存储了 ID 号
m_pShaderProgram	OpenGL 程序	渲染参数
m_nOrderOfArrival	整型	内部 z 轴排序，不改变
m_glServerState	OpenGL 服务状态	OpenGL 附带的服务状态
m_pActionManager	动作管理	用于管理所有动作
m_pScheduler	调度类	调度所有的周期性更新

3.1.2 CCNode 类的函数

CCNode 类的主要函数如表 3-3 所示。

表 3-3 CCNode 类的主要函数

函 数 名	返 回 类 型	描 述
getZOrder	整型	获得兄弟节点间 z 轴顺序
getVertexZ	浮点型	获得 z 轴坐标

（续）

函 数 名	返 回 类 型	描　　述
setVertexZ	空	设置 z 坐标
getRotation	浮点型	获得旋转角度（角度制）
setRotation	空	设置旋转角度（角度制）
getScale	浮点型	获得缩放系数
setScale	空	设置缩放系数
getScaleX	浮点型	获得 x 轴缩放系数
setScaleX	空	设置 x 轴缩放系数
getScaleY	浮点型	获得 y 轴缩放系数
setScaleY	空	设置 y 轴缩放系数
getPosition	点坐标	获得坐标位置
setPosition	空	设置坐标位置
getPositionX	浮点值	获得 x 坐标值
setPositionX	空	设置 x 坐标值
getPositionY	浮点值	获得 y 坐标值
setPositionY	空	设置 y 坐标值
getSkewX	浮点值	获得 x 轴扭曲效果系数
setSkewX	空	设置 x 轴扭曲效果系数
getSkewY	浮点值	获得 y 轴扭曲效果系数
setSkewY	空	设置 y 轴扭曲效果系数
getChildren	数组	获得子节点数组
getChildrenCount	整型	获得子节点数量
getCamera	摄像机	获得节点摄像机
getGrid	网格	获得网格对象
setGrid	空	设置网格对象
isVisible	布尔型	获得是否可见
setVisible	空	设置是否可见
getAnchorPoint	点坐标	获得锚点相对坐标
setAnchorPoint	空	设置锚点相对坐标
getAnchorPointInPoints	点坐标	获得锚点绝对坐标
getContentSize	尺寸	获得节点的尺寸
setContentSize	空	设置节点的尺寸
isRunning	布尔型	获得节点是否在运行
getParent	节点	获得父节点
setParent	空	设置父节点
isIgnoreAnchorPointForPosition	布尔型	是否忽略锚点位置
ignoreAnchorPointForPosition	空	设置是否忽略锚点位置
getTag	整型	获得标签值

（续）

函 数 名	返 回 类 型	描　　述
setTag	空	设置标签值
getUserData	空	获得用户数据
setUserData	空	设置用户数据
getUserObject	对象	获得用户数据对象
setUserObject	空	设置用户数据对象
getActionManager	动作管理	获得动作管理对象
setActionManager	空	设置动作管理对象
getScheduler	调度	获得调度对象
setScheduler	空	设置调度对象
getShaderProgram	渲染参数	获得渲染参数对象
setShaderProgram	空	设置渲染参数对象
onEnter	空	进入节点（场景类）的对象
onEnterTransitionDidFinish	空	场景等切换动画播放完毕进入
onExit	空	离开节点（场景类）的对象
onExitTransitionDidStart	空	场景等切换动画播放完毕离开
addChild	空	添加子节点，参数可以加入 z 轴排序参数，标签值
removeFromParentAndCleanup	空	从父节点删除本节点，参数决定是否清除本节点
removeChild	空	删除自动节点，参数为节点对象和是否清除本节点
removeChildByTag	空	根据标签值删除节点，参数为标签和是否清除本节点
removeAllChildrenWithCleanup	空	删除所有子节点，参数决定是否清除本节点
getChildByTag	节点	根据标签值获得子节点
reorderChild	空	根据 z 轴值重新排列子节点，参数为节点和 z 轴值
sortAllChildren	空	在渲染前排列所有节点，可以被 reorderChild 和 addChild 代替。除非在某一帧里有节点的添加和删除，否则不会自动调用
cleanup	空	停止所有的动作和调度
draw	空	渲染函数
visit	空	递归方法遍历到本节点和子节点并绘制它们
runAction	动作	运行动作
stopAllActions	空	结束所有动作
getActionByTag	动作	根据动作标签获得动作
stopAction	空	结束动作，传入的参数是动作指针
stopActionByTag	空	根据标签值结束动作
numberOfRunningActions	整型	获得运行动作数量
transform	空	运行矩阵变化
convertToNodeSpace	点坐标	转换为节点空间坐标，相对于节点的左下角，与锚点无关
convertToWorldSpace	点坐标	转换为世界空间（全局绝对）坐标，与锚点无关

（续）

函 数 名	返 回 类 型	描　　述
convertToNodeSpaceAR	点坐标	转换为节点空间坐标，传入值和输出值都相对于锚点
convertToWorldSpaceAR	点坐标	转换为世界空间坐标，传入值和输出值都相对于锚点
convertTouchToNodeSpace	点坐标	从触屏对象转换为节点空间坐标
convertTouchToNodeSpaceAR	点坐标	从触屏对象转换为节点空间坐标，传入值和输出值都相对于锚点
nodeToParentTransform	仿射变换矩	返回从本地节点坐标到父节点空间坐标的矩阵变换仿射矩阵
parentToNodeTransform	仿射变换矩	返回从父节点空间坐标到本地节点坐标的矩阵变换仿射矩阵
nodeToWorldTransform	仿射变换矩	返回从本地节点坐标到世界坐标的矩阵变换仿射矩阵
worldToNodeTransform	仿射变换矩	返回从世界坐标到本地节点坐标的矩阵变换仿射矩阵

以上就是 CCNode 类的部分函数，其中不包括本章后面介绍的调度相关的函数。

在 2.0 版本之前，CCNode 类的构造函数是 CCNode：：node（）；在 2.0 版本以后可以使用 create 函数进行创建。

注意 非 C++ 程序员可能会在 API 文档中遇到陌生的 virtual 修饰符。其修饰的结果是使此函数为虚函数。虚函数必须是基类的非静态成员函数，其访问权限可以是 protected 或 public，实现多态性，通过指向派生类的基类指针，访问派生类中同名覆盖成员函数。

在定义了虚函数后，可以在基类的派生类中对虚函数重新定义。在派生类中重新定义的函数应与虚函数具有相同的形参个数和形参类型，以实现统一的接口，不同定义过程。如果在派生类中没有对虚函数重新定义，则它继承其基类的虚函数。简而言之，虚函数允许在程序运行过程中根据指针的类型动态地选择它调用指针类型的函数。

3.1.3 坐标系简介

通过前面的学习，大家会对 CCNode 类的属性和方法有所了解，但是可能有些名词会令你困惑，如 OpenGL 坐标系、世界坐标系、节点相对坐标系、仿射变换等。本节就来解决这些问题。

1. OpenGL 坐标系

Cocos2D-x 以 OpenGL 和 OpenGL ES 为基础，所以自然支持 OpenGL 坐标系。该坐标系原点在屏幕左下角，x 轴向右，y 轴向上。

屏幕坐标系使用的是不同的坐标系统，原点在屏幕左上角，x 轴向右，y 轴向下。iOS 的屏幕触摸事件 CCTouch 传入的位置信息使用的是该坐标系。因此在 Cocos2D-x 中对触摸事件做出响应前，需要首先把触摸点转化到 OpenGL 坐标系。这一点在后面的触屏信息中会详细介绍，可以使用 CCDirector 的 convertToGL 方法来完成这一转化。

2. 世界坐标系

世界坐标系也叫作绝对坐标系，是游戏开发中建立的概念，因此，“世界”即是游戏世界。它建立了描述其他坐标系所需要的参考标准。我们能够用世界坐标系来描述其他坐标系的位置。它是 Cocos2D-x 中一个比较大的概念。

Cocos2D-x 中的元素是有父子关系的层级结构。通过 CCNode 设置位置使用的是相对其父节点的本地坐标系，而非世界坐标系。最后在绘制屏幕的时候，Cocos2D-x 会把这些元素的本地节点坐标映射成世界坐标系坐标。世界坐标系和 OpenGL 坐标系方向一致，原点在屏幕左下角，x 轴向右，y 轴向上。

下面让我们暂时从纷乱的坐标系中抽出来，看一个重要概念：锚点。

3. 锚点

锚点指定了贴图上和所在节点原点（也就是设置位置的点）重合的点的位置，因此只有在 CCNode 类节点使用贴图的情况下，锚点才有意义。

锚点的默认值是（0.5，0.5），表示的并不是一个像素点，而是一个乘数因子。（0.5，0.5）表示锚点位于贴图长度乘 0.5 和宽度乘 0.5 的地方，即贴图的中心。

改变锚点的值虽然可能看起来节点的图像位置发生了变化，但其实并不会改变节点的位置，其实变化的只是贴图相对于你设置的位置的相对位置，相当于你在移动节点里面的贴图，而非节点本身。如果把锚点设置成（0，0），贴图的左下角就会和节点的位置重合，这可能使得元素定位更为方便，但会影响到元素的缩放和旋转等一系列变换。因此并没有一种锚点设置是放之四海而皆准的，要根据你这个对象的使用情况来定义。在 Cocos2D-x 中锚点为默认值（0.5，0.5），这样的锚点设置要把一个节点放置到贴图的中央。

4. 节点坐标系

节点坐标系是和特定节点相关联的坐标系。每个节点都有独立的坐标系。当节点移动或改变方向时，和该节点关联的坐标系（它的子节点）将随之移动或改变方向。这一切都是相对的，相对于基准的，只有在节点坐标系中才有意义。

CCNode 类的设置位置使用的就是父节点的节点坐标系。它和 OpenGL 坐标系的方向也是一致的，x 轴向右，y 轴向上，原点在父节点的左下角。如果父节点是场景树中的顶层节点，那么它使用的节点坐标系就和世界坐标系重合了。

CCNode 类对象中有两个方便的函数可以做坐标转换：

- convertToWorldSpace：把基于当前节点的本地坐标系下的坐标转换到世界坐标系中。
- convertToNodeSpace：把世界坐标转换到当前节点的本地坐标系中。

这两种转换都是不考虑锚点的，都以当前节点父类的左下角的坐标为标准。另外，CCNode 还提供了 convertToWorldSpaceAR 和 convertToNodeSpaceAR。这两个方法完成同样的功能，但是它们的基准坐标是基于坐标锚点的。几乎所有的游戏引擎都会使用类似的本地坐标系而非世界坐标系来指定元素的位置。

这样做的好处是，当计算物体运动的时候，使用同一本地坐标系的元素可以作为一个子

系统独立计算，最后再加上坐标系的运动即可，这是物理研究中常用的思路。例如，一个在行驶的车厢内上下跳动的人，只需要在每帧绘制的时候计算他在车厢坐标系中的位置，然后加上车的位置就可以计算出人在世界坐标系中的位置。如果使用单一的世界坐标系，人的运动轨迹就变复杂了，就涉及中学所学到的运动轨迹的合成与分解。

5. 仿射变换

最后介绍仿射变换。游戏大量使用的旋转、缩放、平移等都是仿射变换。所谓仿射变换是指在线性变换的基础上加上平移。平移不是线性变换。

二维计算机图形学中的仿射变换通常是通过和 3 × 3 齐次矩阵相乘来实现的。Cocos2D-x 的 CCAffineTransform 结构体的定义如代码清单 3-1 所示。

代码清单 3-1　CCAffineTransform 结构体定义

```
struct CCAffineTransform {
  CCFloat a, b, c, d;
  CCFloat tx, ty;
}
```

其中定义了 a、b、c、d、t_x、t_y，它们在矩阵中的位置如图 3-3 所示。

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

图 3-3　参数对应矩阵位置

3.1.4　实例：通过节点控制屏幕中的全体渲染对象

本节，已经可以利用我们目前所学的知识做出一些有趣的东西。之前已经说过，CCNode 类没有贴图，也就是说在屏幕上单独建立一个节点是没有任何效果的，但是可以通过这个“无形”的节点来控制屏幕上的节点。现在就开始吧！

1. 加入节点

新建一个项目，并在 HelloWorldScene.cpp 文件中的 init 函数中做如代码清单 3-2 的代码所示的修改。

代码清单 3-2　加入节点

```
bool HelloWorld::init()
{

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

    // 创建一个节点
    CCNode *anode = CCNode::create();
    // 将节点作为子节点加入场景类中
    this->addChild(anode,0);
```

```
    CCMenuItemImage *pCloseItem = CCMenuItemImage::create(
                                        "CloseNormal.png",
                                        "CloseSelected.png",
                                        this,
    menu_selector(HelloWorld::menuCloseCallback) );
   pCloseItem->setPosition( ccp(CCDirector::sharedDirector()->getWinSize().width
   - 20, 20) );

    CCMenu* pMenu = CCMenu::create(pCloseItem, NULL);
    pMenu->setPosition( CCPointZero );
    // 将菜单作为子节点加入之前定义的节点中
    anode->addChild(pMenu, 1);

    CCLabelTTF* pLabel = CCLabelTTF::create("Hello World", "Thonburi", 34);

    CCSize size = CCDirector::sharedDirector()->getWinSize();

    pLabel->setPosition( ccp(size.width / 2, size.height - 20) );

    // 将标签作为子节点加入之前定义的节点中
    anode->addChild(pLabel, 1);

    CCSprite* pSprite = CCSprite::create("HelloWorld.png");

    pSprite->setPosition( ccp(size.width/2, size.height/2) );

    // 将背景图片作为子节点加入之前定义的节点中
    anode->addChild(pSprite, 0);
    return true;
}
```

首先通过 create 函数创建一个节点 anode，把 anode 作为子节点使用 addChild 函数加入 HelloWorld 场景类对象中；然后把本来作为子节点加入 HelloWorld 场景类对象中的对象，使用 addChild 函数作为子节点加入 anode 中。这些对象包括菜单类对象、标签类对象、人物精灵类对象。这些对象的类都是 CCNode 类的子类。如果目前你对这些对象的使用不了解的话，请跳过这些对象的使用，因为对于这些本书会在后面做相应的讲解。

图 3-4 加入节点控制的项目运行效果

运行项目，发现和修改前的项目并无区别，如图 3-4 所示。

2. 改变位置

修改 HelloWorldScene.cpp 文件中的 init 方法，如代码清单 3-3 所示。

代码清单 3-3　设置节点位置

```
// 设置节点位置
anode->setPosition(ccp(50,50));
```

运行效果如图 3-5 所示。

图 3-5　修改节点位置后的运行效果

可以看到，整体都跟着节点移动了。

3. 设置缩放

下面修改这段代码如代码清单 3-4 所示，看一下缩放的效果。

代码清单 3-4　设置缩放代码

```
// 设置缩放
anode->setScale(0.5);
```

运行效果如图 3-6 所示。

图 3-6　整体缩放后的运行效果

4. 整体旋转

最后再来试验整体旋转，如代码清单 3-5 所示。因为默认锚点在左下角，因此需要首先

移动一下整体的位置，否则整体一转，屏幕中将只显示黑屏。注意角度设置为角度制。

代码清单 3-5 设置旋转

```
// 设置旋转
anode->setPosition(ccp(200,200));
anode->setRotation(90.0);
```

运行效果如图 3-7 所示。

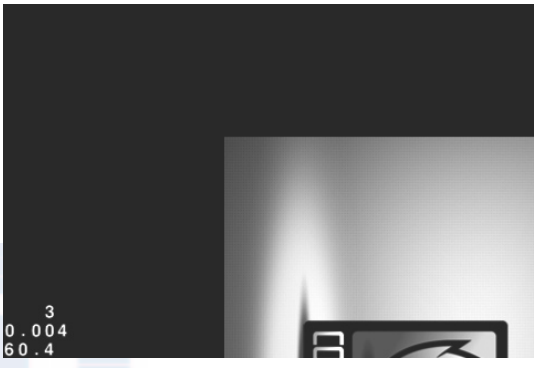

图 3-7 整体旋转后的运行效果

本节介绍了屏幕中渲染对象的基础类 CCNode，下一节将介绍控制游戏显示的导演类 CCDirector。

3.2 导演类

CCDirector 类是 Cocos2D-x 游戏引擎的核心，它用来创建并且控制着主屏幕的显示，同时控制场景的显示时间和显示方式。在整个游戏里一般只有一个导演。游戏的开始、结束、暂停都会调用 CCDirector 类的方法。CCDirector 类具有如下功能。

- ❑ 初始化 OpenGL 会话。
- ❑ 设置 OpenGL 的一些参数和方式。
- ❑ 访问和改变场景以及访问 Cocos2D-x 的配置细节。
- ❑ 访问视图。
- ❑ 设置投影和朝向。

需要说明的是，CCDirector 是单例模式，调用 CCDirector 方法的标准方式如下：

```
CCDirector::sharedDirector()->函数名
```

CCDirector 类的继承关系如图 3-8 所示。

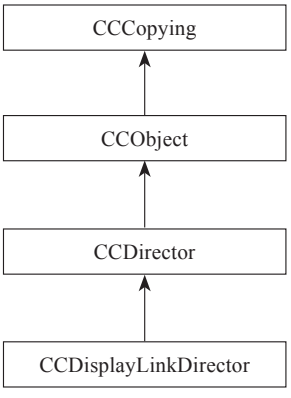

图 3-8 CCDirector 类的继承关系

CCDisplayLinkDirector 继承了 CCDirector，是一个可以自动刷新的导演类。它只支持 1/60 、1/30 和 1/15 三种动画间隔（帧间隔）。

在 Cocos2D-x 里面，在游戏的任何时间，只有一个场景对象实例处于运行状态，而导演就是流程的总指挥，它负责游戏全过程的场景切换，这也是典型的面向对象和分层的设计原则。下面分别介绍 CCDirector 类的成员数据和函数。

3.2.1 CCDirector 类的成员数据

CCDirector 类的主要保护成员数据如表 3-4 所示。

表 3-4　CCDirector 类的主要保护成员数据

名　　称	类　　型	描　　述
m_pScheduler	调度	绑定导演类的调度对象
m_pActionManager	动作管理	绑定导演类的动作管理对象
m_pTouchDispatcher	触屏调度	绑定导演类的触屏调度对象
m_pKeypadDispatcher	键盘调度	绑定导演类的键盘调度对象
m_pAccelerometer	加速度传感器	绑定导演类的加速度传感器对象
m_bPaused	布尔型	导演是运行还是暂停
m_pRunningScene	场景类	正在运行的场景类
m_pNextScene	场景类	下一个场景类
m_fDeltaTime	浮点型	帧间隔的时间
m_obWinSizeInPoints	尺寸	坐标点型的屏幕尺寸
m_obWinSizeInPixels	尺寸	像素级的屏幕尺寸

CCDirector 类没有公共的成员数据，这些属性都不可以直接得到。

3.2.2　CCDirector 类的函数

CCDirector 类的主要公共函数如表 3-5 所示。

表 3-5　CCDirector 类的主要公共函数

函 数 名	返回类型	描　　述
getRunningScene	场景类	获得当前正在运行的场景
getAnimationInterval	小数	获得每帧的时间
setAnimationInterval	空	设置每帧的时间
isDisplayStats	布尔型	返回是否在屏幕左下角显示每帧的时间
setDisplayStats	空	设置是否在屏幕左下角显示每帧的时间
getSecondsPerFrame	浮点值	获得每帧的时间（单位为秒）
getOpenGLView	OpenGL 视图	获得绘制所有对象的 OpenGL 视图
setOpenGLView	空	设置绘制所有对象的 OpenGL 视图
isPaused	布尔型	获得导演类对象是否暂停
getTotalFrames	整型	获得从导演类开始运行的帧数
getProjection	导演投影	获得 OpenGL 的投影
setProjection	空	设置 OpenGL 的投影
getNotificationNode	节点	获得一个节点对象，这个节点对象在主场景被遍历后被遍历
enableRetinaDisplay	布尔型	是否是视网膜版的显示
getWinSize	尺寸	获得屏幕大小（单位为点）
getWinSizeInPixels	尺寸	获得像素级的屏幕大小，单位为像素，只是视网膜版本遇上一个返回值不同

（续）

函 数 名	返回类型	描　　述
reshapeProjection	空	改变投影的大小
convertToGL	点坐标	从 UI 体系的坐标转换为 OpenGL 的坐标
convertToUI	点坐标	从 OpenGL 的坐标转换为 UI 体系的坐标
runWithScene	空	运行当前的场景
popScene	空	弹出当前场景，将它从栈顶弹出
pushScene	空	悬挂当前场景，压入栈中
popToRootScene	空	弹出所有场景，直到根场景
replaceScene	空	替换当前场景
end	空	结束游戏
pause	空	暂停场景
resume	空	重启被暂停的场景，被暂停的时间调度也重新激活
stopAnimation	空	停止动画
startAnimation	空	开始动画
drawScene	空	绘制场景
purgeCachedData	空	清除缓存数据
setGLDefaultValues	空	将 OpenGL 参数设置为默认值
setAlphaBlending	空	设置 OpenGL 是否采用 Alpha 通道
setDepthTest	空	设置是否测试景深
getTouchDispatcher	触屏调度	获得触屏调度对象
setTouchDispatcher	空	设置触屏调度对象
getKeypadDispatcher	键盘调度	获得键盘调度
setKeypadDispatcher	空	设置键盘调度
getAccelerometer	加速度传感器	获得加速度传感器
setAccelerometer	空	设置加速度传感器
getActionManager	动作管理	获得动作管理对象
setActionManager	空	设置动作管理对象
getScheduler	调度	获得调度对象
setScheduler	空	设置调度对象

在 2.0 之前的版本中，曾经有设置导演类的函数，有 4 个导演类型，分别是 kCCDirectorTypeNSTimer、kCCDirectorTypeMainLoop、kCCDirectorType-Thread MainLoop、kCCDirectorTypeDisplayLink。但是在 2.0 版本以后，这些函数已经成为历史了。

3.2.3 实例：CCDirector 类的使用

由于 CCDirector 类是一个控制的类别，从创建项目的模板中，就可以看到 CCDirector 类在游戏初始化的应用，如代码清单 3-6 所示。

代码清单 3-6　CCDirector 类在游戏初始化的应用

```
bool AppDelegate::applicationDidFinishLaunching() {
    // 获得导演类
    CCDirector *pDirector = CCDirector::sharedDirector();

    // 设置 OpenGL 视图
    pDirector->setOpenGLView(&CCEGLView::sharedOpenGLView());

    // 设置是否显示每帧时间
    pDirector->setDisplayStats(true);

    // 设置每帧时间
    pDirector->setAnimationInterval(1.0 / 60);

    // 创建场景
    CCScene *pScene = HelloWorld::scene();

    // 运行场景
    pDirector->runWithScene(pScene);

    return true;
}
```

这段代码出自 AppDelegate.cpp 文件中的 applicationDidFinishLaunching 函数，首先获得导演类指针，然后设置 OpenGL 视图，设置是否显示每帧时间，设置每帧时间，然后创建并运行场景。这样初始化工作就完成了。

在游戏进入后台或者从后台返回时，分别调用相应的方法停止动画和开始动画，如代码清单 3-7 所示。

代码清单 3-7　游戏进入后台或者从后台返回时导演类的工作

```
void AppDelegate::applicationDidEnterBackground() {
    CCDirector::sharedDirector()->stopAnimation();
}


void AppDelegate::applicationWillEnterForeground() {
    CCDirector::sharedDirector()->startAnimation();
}
```

在手机有外部事件进入时，也会将当前界面暂停，比如来电话的时候。

在游戏结束的时候，同样需要使用导演类的结束函数，如代码清单 3-8 所示。

代码清单 3-8　导演类的结束方法

```
void HelloWorld::menuCloseCallback(CCObject* pSender)
{
    CCDirector::sharedDirector()->end();
```

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

从以上例子可以看出，导演类就是一个管理游戏的指挥官。之后还会接触到一些CCDirector类的用法，包括坐标的转换等，这些会在触摸事件那部分做详细介绍。

3.3 场景类

CCScene类是CCNode的子类。和CCNode相比，它只是添加了一个特性，那就是拥有自己的锚点，位置在屏幕的正中央。除此之外，它目前还没有额外的功能，只是一个抽象的概念。

3.1节中介绍CCNode类时，把屏幕上所有显示对象的父节点设置为我们定义的节点，这个父节点的角色一般由场景承担。CCScene类的继承关系如图3-9所示。

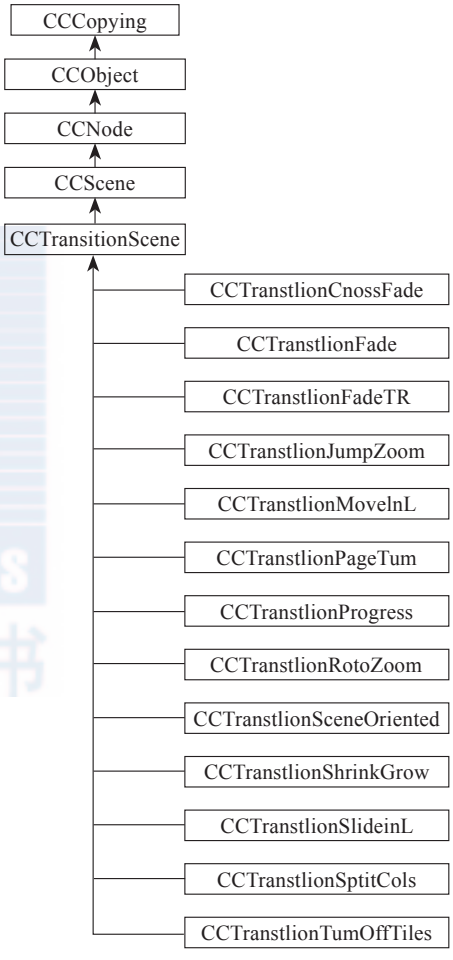

图3-9 CCScene类的继承关系

可以看到，CCScene类有CCTransitionScene（切换场景类），并且CCTransitionScene类有很多子类，这些类都用于切换场景的特效，将在本节的后半段介绍。

注意 游戏通常都会出现这种情况，当切换场景时，程序会由于旧场景的内存没有释放并且新场景已经载入，出现短暂的“峰值”，但是不必为此而做额外的工作，因为Cocos2D-x引擎会清除旧场景的内存。但是要注意，正确使用内存的保留与释放（这点会在后面介绍），尤其是使用场景切换特效的时候，你要做的工作是尽早上机测试和合理的内存使用。

下面来看CCScene类在游戏中的使用和场景切换特效。

3.3.1 如何新建一个场景

下面通过HelloWorld项目中新建场景的过程来总结使用场景的过程。

1）定义一个场景类实例。

Cocos2D-x 引擎中的 HelloWorld 范例中，HelloWorldScene.cpp 中的 scene 函数即是新建场景的地方，如代码清单 3-9 所示。

代码清单 3-9　HelloWorld 新建场景类实例

```
CCScene* HelloWorld::scene()
{
    // 新建场景类实例
    CCScene *scene = CCScene::create();
    //CCScene *scene = CCScene::node();
    // 定义布景层
    HelloWorld *layer = HelloWorld::create();

    // 将布景层加入场景
    scene->addChild(layer);

    // 返回场景类
    return scene;
}
```

通过 create 方法构造一个场景，当然，create 函数是 Cocos2D-x 2.0 以后的版本　才有的方式，2.0 之前的版本采用的是后面被注释掉的方法，然后将定义的布景层作为子项加入场景中，要显示的成员都在布景层中。至于布景层的定义，将在后面做介绍。

2）获得并运行场景。

在游戏入口 AppDelegate 类中的 applicationDidFinishLaunching 函数中调用上一步定义的方法，并使用导演类的 runWithScene 方法运行场景，如代码清单 3-10 所示。

代码清单 3-10　获得并运行场景

```
bool AppDelegate::applicationDidFinishLaunching() {
    // 初始化导演类
    CCDirector *pDirector = CCDirector::sharedDirector();

   // 设置 OpenGL 视图
    pDirector->setOpenGLView(&CCEGLView::sharedOpenGLView());

    // 设置是否显示每帧时间
    pDirector->setDisplayStats(true);

    // 设置每帧时间
    pDirector->setAnimationInterval(1.0 / 60);

    // 创建场景
    CCScene *pScene = HelloWorld::scene();

    // 运行场景
    pDirector->runWithScene(pScene);

    return true;
}
```

到这里已经可以新建一个场景并把它加入游戏中了。

这时也许有人会有疑问：一般来讲，游戏都会由多个场景组成，至少会有菜单场景和游戏场景，更复杂的游戏甚至会有更多场景，那么如何在场景间切换呢？下一节将通过引擎自带的 tests 例子来学习如何在场景间切换。

3.3.2 场景的切换

tests 项目是一个 Cocos2D-x 的功能示例。对于 Cocos2D-x 的初学者来说，看 tests 项目中的示例并理解它们，是个不错的学习方式。本节就通过介绍 tests 的基本结构来介绍场景间的切换。

游戏入口 AppDelegate 类中的 applicationDidFinishLaunching 函数如代码清单 3-11 所示。

代码清单 3-11 tests 项目的 AppDelegate 类的 applicationDidFinishLaunching 函数

```
bool AppDelegate::applicationDidFinishLaunching()
{

    CCDirector *pDirector = CCDirector::sharedDirector();
    pDirector->setOpenGLView(&CCEGLView::sharedOpenGLView());

    pDirector->setDisplayStats(true);

    pDirector->setAnimationInterval(1.0 / 60);

    CCScene * pScene = CCScene::create();
    CCLayer * pLayer = new TestController();
    pLayer->autorelease();

    pScene->addChild(pLayer);
    pDirector->runWithScene(pScene);

    return true;
}
```

这里和上一节的 applicationDidFinishLaunching 有一点区别，它的定义场景和给场景加层的操作都在这个函数里完成，但是本质上是类似的。下面就来看 TestController 布景层里面是如何调用下一级场景的。

切换场景这个事件应该是在按下主菜单上的按键后发生的，于是找到 controller.cpp 文件中的 menuCallback 函数。它是自定义的一个回调函数，在定义布景层时定义。如果不理解这一点，没关系，请跳过，后面会讲解这个问题。

menuCallback 函数如代码清单 3-12 所示。

代码清单 3-12 menuCallback 函数

```
void TestController::menuCallback(CCObject * pSender)
{

    CCMenuItem* pMenuItem = (CCMenuItem *)(pSender);
```

```
    int nIdx = pMenuItem->getZOrder() - 10000;

    TestScene* pScene = CreateTestScene(nIdx);
    if (pScene)
    {
        pScene->runThisTest();
        pScene->release();
    }
}
```

除了处理菜单类的内容，这部分可以暂时跳过。后面调用在 controller.cpp 文件中定义的 CreateTestScene 函数获得场景，然后调用场景的 runThisTest 函数，这是自定义的。

首先来看 CreateTestScene 函数，如代码清单 3-13 所示。

代码清单 3-13　CreateTestScene 函数

```
static TestScene* CreateTestScene(int nIdx)
{
    // 清空缓存
    CCDirector::sharedDirector()->purgeCachedData();

    TestScene* pScene = NULL;

    switch (nIdx)
    {
    case TEST_ACTION_MANAGER:
        pScene = new ActionManagerTestScene();
        break;
    // 中间部分的代码过长，省略到这部分。如果需要完整代码，请参考 Cocos2D-x 引擎目录下 tests
    // 项目的 controller.cpp 文件
    default:
        break;
    }

    return pScene;
}
```

这里省略中间的部分，首先调用导演类的清除缓存函数，然后新建场景，并返回场景。

关于场景的定义，来看 ActionManagerTestScene 场景的定义。ActionManagerTest.h 文件中的 ActionManagerTestScene 类的定义如代码清单 3-14 所示。

代码清单 3-14　ActionManagerTestScene 类的定义

```
class ActionManagerTestScene : public TestScene
{
public:
    virtual void runThisTest();
};
```

ActionManagerTest.cpp 文件中的 runThisTest 函数的定义如代码清单 3-15 所示。

代码清单 3-15 runThisTest 函数的定义

```
void ActionManagerTestScene::runThisTest()
{
    CCLayer* pLayer = nextActionManagerAction();
    addChild(pLayer);

    CCDirector::sharedDirector()->replaceScene(this);
}
```

这里做的就是把布景层加入场景中，并把场景通过导演类的 replaceScene 将当场景替换成场景。

以上过程分为以下三步：

1）调用 CCDirector::sharedDirector()->purgeCachedData() 清空缓存。

2）新建场景。

3）调用 CCDirector::sharedDirector()->replaceScene(this) 替换新场景。Cocos2D-x 提供了场景间切换的特效，下一节将会介绍这些内容。

注意 不要在节点初始化的 init 函数中调用 replaceScene 函数。导演类不允许在一个节点初始化的调用场景替换，否则会导致程序崩溃。

这里说一下压入场景（pushScene）和弹出场景（popScene）。它们都可以用来显示场景和保留当前场景并显示新场景；不同的是它们不把旧场景从内存中释放掉，这样可以提高加载速度，这时需要注意，如果内存不足以支撑的话，建议采用 replaceScene 函数。

3.3.3 场景间切换的动画

上一节介绍了使用 replaceScene 切换场景。很多时候，为了游戏效果，使 UI 更有动态，会在切换界面的过程中加入一些过程动画，如平移切换场景动画（见图 3-10）、旋转切换场景动画（见图 3-11）。

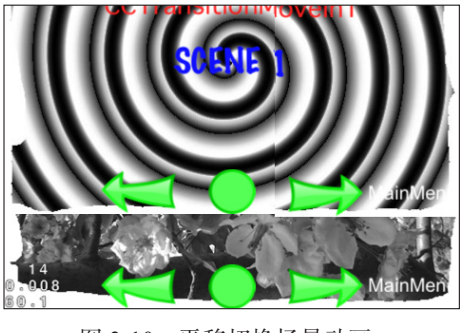

图 3-10 平移切换场景动画

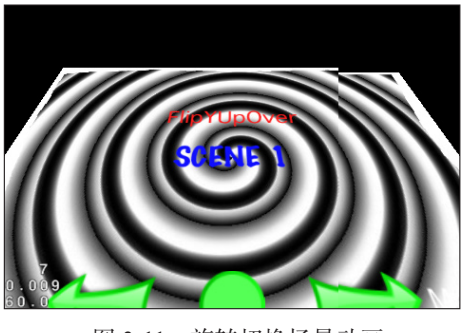

图 3-11 旋转切换场景动画

正常的无过渡场景，如果尚未建立场景（游戏中的第一个场景），使用 runWithScene 函数即可以使用相应场景；如果是替换场景，则使用 replaceScene 函数替换相应的场景即可；而如果要使用场景间的切换效果，则需要使用相应的切换类，即在 3.1 节中介绍的 CCTransitionScene 子类的 create 函数（2.0 版本之前是 transitionWithDuration 函数），生成相应场景。当然，不同效果的使用方法略有不同。

然后，通过 replaceScene 函数启动场景，也就是说给这个场景加了一个外包装类，然后再启动。那么，这个场景就不是直接显示了，而是在场景的效果动画播完以后进入场景，起到过渡的效果。

一般情况，CCTransitionScene 子类的 create 函数有两个参数。第一个参数是特效的切换时间，直接生成一个 CCTime 类即可，例子中设定的时间是 1 ~ 2s，对于很多场景的显示都很舒服，你也可以根据你的要求修改切换时间；第二个参数是要进入的场景。有的类会有第三个参数，如表 3-6 所示。

表 3-6 切换场景动画类

动 画 效 果	类 名	是否有第三个函数和第三个函数功能
跳跃式，原场景先缩小，然后新场景跳跃进来	CCTransitionJumpZoom	无
淡出淡入，原场景淡出，新场景淡入	CCTransitionFade	为渐变的颜色，如 ccWHITE
x 轴平移移动	CCTransitionFlipX	kOrientationLeftOver：向左平移 kOrientationRightOver：向右平移
y 轴平移移动	CCTransitionFlipY	kOrientationUpOver：向上平移 kOrientationDownOver：向下平移
水平角度翻转	CCTransitionFlipAngular	kOrientationLeftOver：向左翻 kOrientationRightOver：向右翻
带缩放效果的 x 轴平移	CCTransitionZoomFlipX	kOrientationLeftOver：向左平移 kOrientationRightOver：向右平移
带缩放效果的 y 轴平移	CCTransitionZoomFlipY	kOrientationUpOver：向上平移 kOrientationDownOver：向下平移
带缩放效果的旋转	CCTransitionZoomFlipAngular	kOrientationLeftOver：向左翻 kOrientationRightOver：向右翻
交错切换	CCTransitionShrinkGrow	无
转角切换	CCTransitionRotoZoom	无
新场景从左移入覆盖	CCTransitionMoveInL	无
新场景从右移入覆盖	CCTransitionMoveInR	无
新场景从上移入覆盖	CCTransitionMoveInT	无
新场景从下移入覆盖	CCTransitionMoveInB	无
新场景从左移入推出原场景	CCTransitionSlideInL	无
新场景从右移入推出原场景	CCTransitionSlideInR	无
新场景从上移入推出原场景	CCTransitionSlideInT	无
新场景从下移入推出原场景	CCTransitionSlideInB	无

（续）

动 画 效 果	类　　名	是否有第三个函数和第三个函数功能
向右上波浪	CCTransitionFadeTR	无
向左下波浪	CCTransitionFadeBL	无
向上百叶窗	CCTransitionFadeUp	无
向下百叶窗	CCTransitionFadeDown	无
随机小方块切换	CCTransitionTurnOffTiles	无
按行切换	CCTransitionSplitRows	无
按列切换	CCTransitionSplitCols	无
翻页	CCTransitionPageTurn	false：前翻 true：后翻

其中，CCTransitionPageTurn 类需要先设置摄像机，使用以下代码：

```
CCDirector::sharedDirector()->setDepthTest(true)
```

有三种特效需要检测 OpenGL 版本是否支持，如表 3-7 所示。使用如下代码检查，如果为真则不支持：

```
CCConfiguration::sharedConfiguration()->getGlesVersion() <= GLES_VER_1_0
```

表 3-7　需要检测 OpenGL 版本的切换场景动画

动 画 效 果	类　　名
淡出淡入交叉，同时进行	CCTransitionCrossFade
顺时针切入	CCTransitionRadialCCW
逆时针切入	CCTransitionRadialCW

tests 项目的 TransitionsTest 文件夹中的是场景切换的示例。切换场景动画的使用步骤如下。

1）新建场景。

2）根据需要的新建场景的切换动画选择 CCTransitionScene 子类，通过 create 将之前建的场景传入其中，并设置其他参数。

3）调用 CCDirector::sharedDirector()->replaceScene（第 2 步中定义的 CCTransitionScene 的子类）替换新场景。

下一节开始介绍布景层类 CCLayer。

3.4　布景层类

布景层类 CCLayer 是 CCNode 类的子类，并且在此基础上实现触屏事件代理（TouchEventsDelegate）协议，可以实现 CCNode 类的功能，并且可以处理输入，包括触屏和加速度传感器。

每个游戏场景中可以有很多层，每一层负责各自的任务，如专门负责显示地图的背景、

专门负责显示敌人、专门负责机关和专门负责主角等；每一层上可以放置不同的元素，包括文本、精灵图片和菜单等。通过层与层之间的组合关系，就可以构成游戏显示的界面 UI，游戏中等。当然为了看到每一层的东西，可把一些层设置为透明或半透明的，这样就可以看到不同布景层叠加到一起的效果了。CCLayer 类的继承关系如图 3-12 所示。

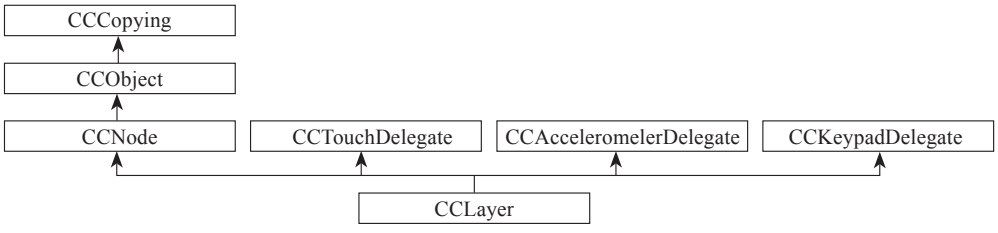

图 3-12　CCLayer 类的继承关系

由图 3-12 可以看出 CCLayer 类继承自 CCNode 类，并且 CCLayer 类还遵照触屏代理协议、加速度传感器代理协议、键盘时间代理协议等协议。除此之外，CCLayer 类还有子类，如图 3-13 所示。

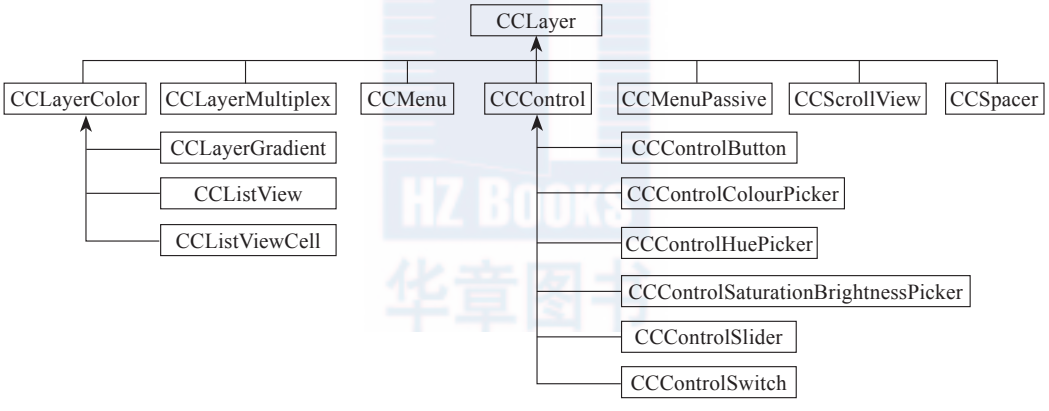

图 3-13　CCLayer 类的子类

这些子类的功能如表 3-8 所示。

表 3-8　CCLayer 子类的功能

类　名	功　能
CCLayerColor	实现 CCRGBAProtocol 协议，可设置层的颜色和不透明度
CCLayerMultiplex	可以将它和子布景层结合在一起
CCMenu	菜单布景层
CCControl	UI 控件
CCMenuPassive	这个布景层不发送时间，每个节点做自己的事情
CCScrollView	支持 Cocos2D-iPhone 的滚动视图
CCSpacer	获得水平或垂直的空间

首先来看 CCLayer 类的使用，然后再来看主要的子类使用。

3.4.1 CCLayer 类的函数

CCLayer 类的主要函数如表 3-9 所示。

表 3-9 CCLayer 类的主要函数

函 数 名	返回类型	描 述
init	布尔型	初始化 CCLayer
onEnter	空	当进入布景层时被调用
onExit	空	当离开布景层时被调用
onEnterTransitionDidFinish	空	过场动画结束时调用
ccTouchBegan	布尔型	触屏触摸屏幕
ccTouchMoved	空	触屏移动
ccTouchEnded	空	触屏结束
ccTouchCancelled	空	触屏取消
ccTouchesBegan	空	触屏触摸屏幕（多点触控）
ccTouchesMoved	空	触屏移动（多点触控）
ccTouchesEnded	空	触屏结束（多点触控）
ccTouchesCancelled	空	触屏取消（多点触控）
didAccelerate	空	加速度传感器
registerWithTouchDispatcher	空	如果触屏被允许，这个方法在 init 里调用
isTouchEnabled	布尔型	获得是否触屏
setTouchEnabled	空	设置触屏
isAccelerometerEnabled	布尔型	是否获得加速度传感器
setAccelerometerEnabled	空	设置获得加速度传感器
isKeypadEnabled	布尔型	是否获得键盘事件
setKeypadEnabled	空	设置获得键盘事件

来看 Cocos2D-x 的 HelloWorld 项目中的 HelloWorldScene.cpp 文件，scene 函数定义 CCLayer 类并把它加入场景中，如代码清单 3-16 所示。

代码清单 3-16 scene 函数定义 CCLayer 类并把它加入场景中

```
CCScene* HelloWorld::scene()
{
    // 新建场景类实例
    CCScene *scene = CCScene::create();

    // 定义布景层
    HelloWorld *layer = HelloWorld::create();

    // 将布景层加入场景
    scene->addChild(layer);
```

```
    //返回场景类
    return scene;
}
```

CCLayer 类的 init 函数在创建布景层时被调用，如代码清单 3-17 所示。

代码清单 3-17　CCLayer 类的 init 函数

```
bool HelloWorld::init()
{
    if ( !CCLayer::init() )
    {
        return false;
    }
    CCMenuItemImage *pCloseItem = CCMenuItemImage::create(
                                        "CloseNormal.png",
                                        "CloseSelected.png",
                                        this,
    menu_selector(HelloWorld::menuCloseCallback) );
    pCloseItem->setPosition( ccp(CCDirector::sharedDirector()->getWinSize().
    width - 20, 20) );
    CCMenu* pMenu = CCMenu::create(pCloseItem, NULL);
    pMenu->setPosition( CCPointZero );
    this->addChild(pMenu, 1);
    CCLabelTTF* pLabel = CCLabelTTF::create("Hello World", "Arial", 24);
    CCSize size = CCDirector::sharedDirector()->getWinSize();
    pLabel->setPosition( ccp(size.width / 2, size.height - 50) );
    this->addChild(pLabel, 1);
    CCSprite* pSprite = CCSprite::create("HelloWorld.png");
    pSprite->setPosition( ccp(size.width/2, size.height/2) );
    this->addChild(pSprite, 0);

    return true;
}
```

本书将在后面介绍具体显示在层次上的对象，目前只需要了解在 init 函数中定义要显示的对象并把它作为子类加入场景中。另外，关于触屏、键盘、加速度传感器等输入，将在后面的章节介绍。本节后面将介绍 CCLayer 类的子类。

3.4.2　颜色布景层类 CCLayerColor

颜色布景层类 CCLayerColor 是 CCLayer 类的子类，包含 CCLayer 类的特性，并且有两个拓展功能：可以为布景层增添颜色，以及设置不透明度。

首先看 CCLayerColor 类的定义初始化，如代码清单 3-18 所示。这段代码是 tests 项目下 LayerTest.cpp 文件中 LayerTest1 的 onEnter 函数。

代码清单 3-18 CCLayerColor 类的定义初始化

```
void LayerTest1::onEnter()
{
    LayerTest::onEnter();

    setTouchEnabled(true);

    CCSize s = CCDirector::sharedDirector()->getWinSize();
    CCLayerColor* layer = CCLayerColor::create( ccc4(0xFF, 0x00, 0x00, 0x80), 200, 200);

    layer->ignoreAnchorPointForPosition(false);
    layer->setPosition( CCPointMake(s.width/2, s.height/2) );
    addChild(layer, 1, kTagLayer);
}
```

create 函数的第一个参数是颜色的 ARGB 值，使用 ccc4 定义，其中第一个参数是颜色 a 值，第二个参数是 R 值，第三个参数是 G 值，最后一个参数是 B 值。除此之外，create 函数的后两个参数是布景层的宽和高。

另外，使用 ignoreAnchorPointForPosition 将忽略锚点置为 false。由于默认设置是忽略锚点，也就是以左下角为锚点，可以让布景层考虑锚点的影响（关于锚点在之前已经介绍过），这时默认的锚点在中心。运行效果如图 3-14 所示。

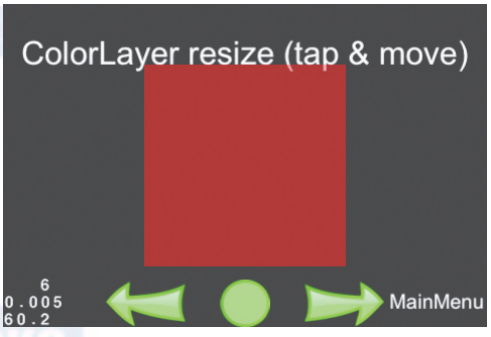

图 3-14 定义颜色布景层的显示效果

另外，LayerTest.cpp 文件中的 LayerTest1 的 updateSize 函数中，可以修改颜色布景层的大小，如代码清单 3-19 所示。

代码清单 3-19 修改颜色布景层的大小

```
void LayerTest1::updateSize(CCPoint &touchLocation)
{
    CCSize s = CCDirector::sharedDirector()->getWinSize();

    CCSize newSize = CCSizeMake( fabs(touchLocation.x - s.width/2)*2, fabs(touchLocation.
    y - s.height/2)*2);

    CCLayerColor* l = (CCLayerColor*) getChildByTag(kTagLayer);

    l->setContentSize( newSize );
}
```

通过 setContentSize 可以设置颜色布景层的大小。对于 CCLayerColor 类来说，还有一个比较常用的函数 setBlendFunc，可以让布景层的颜色产生渐变效果。比如，需要在整屏添加全屏的一些覆盖效果、全屏变黑或者全屏变暗时，都可以使用这个方法。代码清单 3-20 所示

是 tests 项目 LayerTestBlend 类中的 newBlend 函数，也就是使用 setBlendFunc 的示例。

代码清单 3-20 使用 setBlendFunc 的示例

```
void LayerTestBlend::newBlend(float dt)
{
     CCLayerColor *layer = (CCLayerColor*)getChildByTag(kTagLayer);

    GLenum src;
    GLenum dst;

    if( layer->getBlendFunc().dst == GL_ZERO )
    {
        src = GL_SRC_ALPHA;
        dst = GL_ONE_MINUS_SRC_ALPHA;
    }
    else
    {
        src = GL_ONE_MINUS_DST_COLOR;
        dst = GL_ZERO;
    }

    ccBlendFunc bf = {src, dst};
    layer->setBlendFunc( bf );
}
```

传入的参数是一个有起始效果和结束效果的参数，运行之后的效果如图 3-15 和图 3-16 所示。

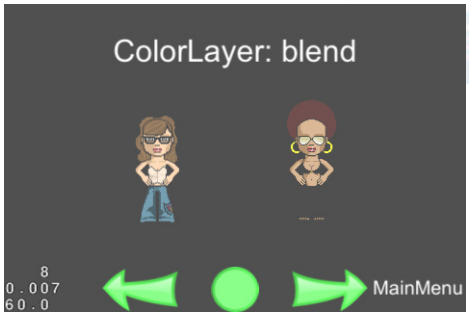

图 3-15 传入参数 GL_SRC_ALPHA 的效果

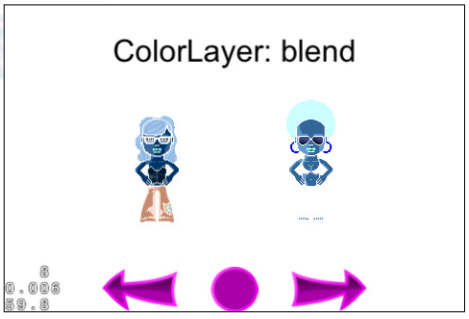

图 3-16 传入参数 GL_ONE_MINUS_DST_COLOR 的效果

CCLayerColor 类作为一个带颜色的布景层，在开发中可以给我们做特效带来方便。下一节介绍多层布景层类。

3.4.3 多层布景层类 CCLayerMultiplex

在游戏开发中，一般会把游戏分为两部分：一部分是游戏界面部分，也就是常说

的 UI 部分（User Interface，用户界面）；另一部分就是游戏本身部分。有时 UI 有很多页面，在页面中用的图也并不是很多，不需要使用切换场景，只需把不同页面做成不同的布景层，然后切换布景层。那么这就需要一个“管理者”来管理这些界面，这时候就要使用 CCLayerMultiplex 类。

在很多游戏中都需要在不同的界面中使用相同的几个变量，如果不这样做，就需要做大量的保存工作。

tests 项目中 MenuTest.cpp 的 MenuTestScene 类的 runThisTest 函数中有 CCLayerMultiplex 类的定义初始化方法，如代码清单 3-21 所示。

代码清单 3-21 CCLayerMultiplex 类的定义初始化

```
void MenuTestScene::runThisTest()
{
    CCLayer* pLayer1 = new MenuLayerMainMenu();
    CCLayer* pLayer2 = new MenuLayer2();
    CCLayer* pLayer3 = new MenuLayer3();
    CCLayer* pLayer4 = new MenuLayer4();
    CCLayer* pLayer5 = new MenuLayerPriorityTest();

    CCLayerMultiplex* layer = CCLayerMultiplex::create(pLayer1, pLayer2, pLayer3,
        pLayer4, pLayer5, NULL);
    addChild(layer, 0);

    pLayer1->release();
    pLayer2->release();
    pLayer3->release();
    pLayer4->release();
    pLayer5->release();

    CCDirector::sharedDirector()->replaceScene(this);
}
```

首先定义并初始化每个布景层类，然后将这些布景层实例以参数形式传给 CCLayerMultiplex 的 create 函数，最后以 NULL（空）结束。

这里在传入参数之后将这些布景层实例的指针释放，是为了防止内存泄露。至于 Cocos2D-x 的内存管理，本书将会在后面的章节介绍。

然后把 CCLayerMultiplex 实例作为子节点传入场景中，最后运行场景。代码清单 3-22 所示是切换布景层的 switchTo 函数使用方法。

代码清单 3-22 switchTo 函数使用方法

```
void MenuLayerPriorityTest::menuCallback(CCObject* pSender)
{
    ((CCLayerMultiplex*)m_pParent)->switchTo(0);
}
```

由于这个函数被 CCLayerMultiplex 实例的子布景，即初始化 CCLayerMultiplex 传入的布景类实例调用，所以它的 m_pParent 父节点就是 CCLayerMultiplex 实例本身。获得 CCLayerMultiplex 实例指针后，调用 switchTo 函数就可以转换到相应的子布景中。关于子布景中显示的菜单，下一节将会介绍。

3.4.4 菜单类 CCMenu

游戏中常用的菜单如图 3-17 所示，其中菜单项可以是图片、系统字，或者自定义的字体。

图 3-17 游戏中常用的菜单示例

CCMenu 是一个菜单项的容器，用来装载各种菜单项。代码清单 3-23 就是一个定义 CCMenu 类实例的过程，是 tests 项目中 MenuTest.cpp 的 MenuLayer2 的构造函数。

代码清单 3-23 定义 CCMenu 类实例

```
MenuLayer2::MenuLayer2()
{
    for( int i=0;i < 2;i++ )
    {
        CCMenuItemImage* item1 = CCMenuItemImage::create(s_PlayNormal, s_
    PlaySelect, this, menu_selector(MenuLayer2::menuCallback));
        CCMenuItemImage* item2 = CCMenuItemImage::create(s_HighNormal, s_
    HighSelect, this, menu_selector(MenuLayer2::menuCallbackOpacity) );
        CCMenuItemImage* item3 = CCMenuItemImage::create(s_AboutNormal, s_
    AboutSelect, this, menu_selector(MenuLayer2::menuCallbackAlign) );

        item1->setScaleX( 1.5f );
        item2->setScaleX( 0.5f );
        item3->setScaleX( 0.5f );

        CCMenu* menu = CCMenu::create(item1, item2, item3, NULL);

        CCSize s = CCDirector::sharedDirector()->getWinSize();
        menu->setPosition(ccp(s.width/2, s.height/2));
```

```
        menu->setTag( kTagMenu );

        addChild(menu, 0, 100+i);

        m_centeredMenu = menu->getPosition();
    }

    m_alignedH = true;
    alignMenusH();
}
```

首先定义菜单项（关于菜单项，本书会在后面的章节中做讲解），然后用它们定义初始化菜单 CCMenu 实例，最后将 CCMenu 实例加入 CCLayer 中显示出来，效果如图 3-18 所示。

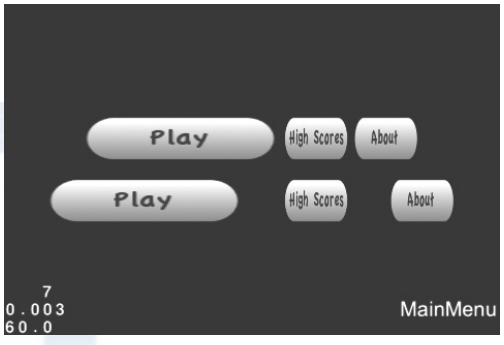

图 3-18 定义的菜单对象显示效果

菜单类还提供了 alignItemsVertically 和 align-ItemsHorizontally 等函数。如代码清单 3-24 所示，tests 项目中 MenuTest.cpp 的 MenuLayer2 的构造函数 alignMenusH 就是 alignItems-Horizontally 水平对齐两种方法对比，一种是 alignItems-Horizontally 水平对齐，底下是 alignItemsHorizontallyWithPadding 留空间水平对齐，效果对比请见之前的图 3-18。

代码清单 3-24 对齐方法对比函数

```
void MenuLayer2::alignMenusH()
{
    for(int i=0;i<2;i++)
    {
        CCMenu *menu = (CCMenu*)getChildByTag(100+i);
        menu->setPosition( m_centeredMenu );
        if(i==0)
        {
            // TIP: if no padding, padding = 5
            menu->alignItemsHorizontally();
            CCPoint p = menu->getPosition();
            menu->setPosition( ccpAdd(p, CCPointMake(0,30)) );

        }
        else
        {
            // TIP: but padding is configurable
            menu->alignItemsHorizontallyWithPadding(40);
            CCPoint p = menu->getPosition();
            menu->setPosition( ccpSub(p, CCPointMake(0,30)) );
        }
    }
}
```

使用方法比较简单，直接调用就可以。下一节介绍 Cocos2D-x 中的 UI 控件。

3.4.5 控件类及其子类

在应用的开发中，无论是 Android 操作系统还是 iOS 操作系统，其开发框架都提供了控件，包括按键、拖动滑块等，这样提高了开发效率。对于游戏的开发，UI 的开发同样需要控件来提高开发效率。对 Cocos2D-x 来说，从 2.0 版本开始提供了很多控件类来帮助我们更好地开发 UI。

1. 拖动滑块的控件类 CCControlSlider

首先来看拖动滑块的控件类 CCControlSlider。tests 项目中 ControlExtensionTest\CCControlSliderTest 目录下 CCControlSliderTest.cpp 中的代码如代码清单 3-25 所示。

代码清单 3-25 定义并初始化 CCControlSliderTest 类实例

```
bool CCControlSliderTest::init()
{
    if (CCControlScene::init())
    {
        CCSize screenSize = CCDirector::sharedDirector()->getWinSize();
        // 定义标签的代码，考虑到篇幅而被省略
       ...
        // 定义 CCControlSlider
        CCControlSlider *slider = CCControlSlider::create("extensions/
            sliderTrack.png","extensions/sliderProgress.png" ,"extensions/
            sliderThumb.png");
        slider->setAnchorPoint(ccp(0.5f, 1.0f));
        slider->setMinimumValue(0.0f); // 设置范围最小值
        slider->setMaximumValue(5.0f); // 设置范围最大值
        slider->setPosition(ccp(screenSize.width / 2.0f, screenSize.height /
            2.0f));
        // 添加回调函数，当滑块被拖动时被调用
        slider->addTargetWithActionForControlEvents(this, cccontrol_selector(CCC
            ontrolSliderTest::valueChanged), CCControlEventValueChanged);
        addChild(slider);
        return true;
    }
    return false;
}
void CCControlSliderTest::valueChanged(CCObject *sender, CCControlEvent
    controlEvent)
{
    CCControlSlider* pSlider = (CCControlSlider*)sender;
m_pDisplayValueLabel->setString(CCString::createWithFormat("Slider value =
    %.02f", pSlider->getValue())->getCString());
}
```

要定义拖动滑块对象，首先调用 create 函数，参数为图片路径，分别是滑块滑道图片

路径、滑块滑动后覆盖滑道图片路径和滑块图片路径；之后设置锚点，并设置范围最小值和设置范围最大值，设置位置后给拖动注册拖动事件接受函数。在拖动事件中，可以通过pSlider->getValue())->getCString() 来获取目前所在位置的值，运行效果如图 3-19 所示。

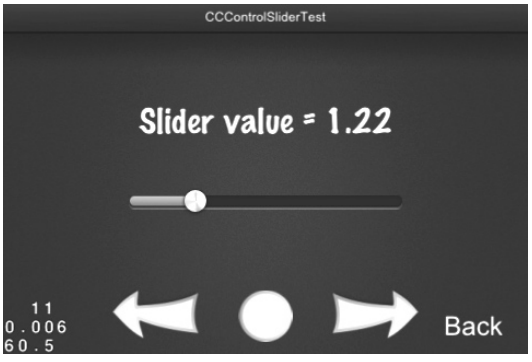

图 3-19 CCControlSlider 运行效果

2. 颜色选择盘类 CCControlColourPicker

颜色选择盘类 CCControlColourPicker 的定义和初始化如代码清单 3-26 所示。代码在 tests 项 目 中 ControlExtensionTest\ CCControlColourPickerTest 目录下的 CCControlColourPickerTest.cpp 文件中。

代码清单 3-26 CCControlColourPicker 类的定义和初始化

```
bool CCControlColourPickerTest::init()
{
    if (CCControlScene::init())
    {
        CCSize screenSize = CCDirector::sharedDirector()->getWinSize();

        CCNode *layer  = CCNode::create();
        layer->setPosition(ccp (screenSize.width / 2, screenSize.height / 2));
        addChild(layer, 1);

        double layer_width = 0;

        // 定义并初始化颜色选择盘
        CCControlColourPicker *colourPicker = CCControlColourPicker::create();
        colourPicker->setColor(ccc3(37, 46, 252));
        colourPicker->setPosition(ccp (colourPicker->getContentSize().width / 2, 0));
        // 添加到层次中
        layer->addChild(colourPicker);
        // 注册事件
        colourPicker->addTargetWithActionForControlEvents(this, cccontrol_selector(
    CCControlColourPickerTest::colourValueChanged), CCControlEventValueChanged);
        // 以下定义其他控件的代码省略
       ...
        return true;
    }
    return false;

}
void CCControlColourPickerTest::colourValueChanged(CCObject *sender,
CCControlEvent controlEvent)
{
   CCControlColourPicker* pPicker = (CCControlColourPicker*)sender;
```

```
    m_pColorLabel->setString(CCString::createWithFormat("#%02X%02X%02X",pPicker-
        >getColorValue().r, pPicker->getColorValue().g, pPicker->getColorValue().
        b)->getCString());
}
```

首先定义 CCControlColourPicker，直接调用 create 函数就可以。这里需要把 tests\Resources\extensions 目录下和 CCControlColourPicker 相关的文件复制到 Resources\extensions 目录，然后定义初始颜色，加入父节点中，并注册回调函数。回调函数通过 pPicker->getColorValue().g, pPicker->getColorValue().b)->getCString() 获得相应的颜色值字符串。运行效果如图 3-20 所示。

图 3-20　CCControlColourPicker 运行效果

3. 开关按钮类 CCControlSwitch

开关按钮类 CCControlSwitch 的定义和初始化如代码清单 3-27 所示。代码在 tests 项目中的 ControlExtensionTest\ CCControlSwitchTest 目录下的 CCControlSwitchTest.cpp 中。

代码清单 3-27　CCControlSwitch 的定义和初始化

```
bool CCControlSwitchTest::init()
{
    if (CCControlScene::init())
    {
        // 定义其他控件，代码省略
        ...
        // 定义开关控件
        CCControlSwitch *switchControl = CCControlSwitch::create
            (
                CCSprite::create("extensions/switch-mask.png"),
                CCSprite::create("extensions/switch-on.png"),
                CCSprite::create("extensions/switch-off.png"),
                CCSprite::create("extensions/switch-thumb.png"),
                CCLabelTTF::create("On", "Arial-BoldMT", 16),
                CCLabelTTF::create("Off", "Arial-BoldMT", 16)
            );
        switchControl->setPosition(ccp (layer_width + 10 + switchControl-
            >getContentSize().width / 2, 0));
        layer->addChild(switchControl);
        switchControl->addTargetWithActionForControlEvents(this, cccontrol_selec
            tor(CCControlSwitchTest::valueChanged), CCControlEventValueChanged);
        // 定义其他控件，代码省略
        ...
        return true;
```

```
        }
        return false;
    }
```

要定义开关对象，首先调用 create 函数，参数为图片路径和上面的文字标签。图片路径分别是背景图片路径、开状态背景图片路径、关状态背景图片路径和开关背景图片路径，文字标签是开文字标签、关文字标签。设置位置加入布景层后定义回调函数。运行效果如图 3-21 所示。

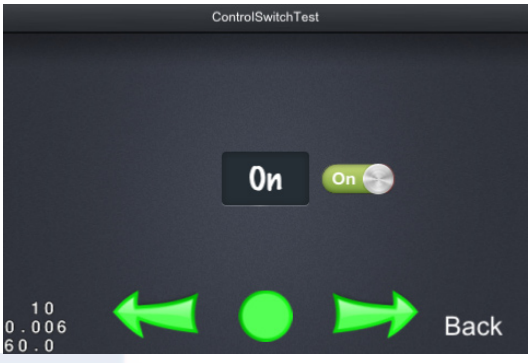

图 3-21 CCControlSwitch 运行效果

4. 按钮类 CCControlButton

按钮类 CCControlButton 的定义和初始化如代码清单 3-28 所示。代码是 tests 项目中 ControlExtensionTest \ CCControlButtonTest 目录下 CCControlButtonTest.cpp 中的 CCControlButtonTest_Event 的 init 函数。

代码清单 3-28 CCControlButton 的定义和初始化

```
bool CCControlButtonTest_Event::init()
{
    if (CCControlScene::init())
    {
        // 定义其他控件，代码省略
        ...
        // 定义并初始化按钮
        CCControlButton *controlButton = CCControlButton::create(titleButton,
            backgroundButton);
        controlButton->setBackgroundSpriteForState(backgroundHighlightedButton,
            CCControlStateHighlighted);
        controlButton->setTitleColorForState(ccWHITE, CCControlStateHighlighted);

        controlButton->setAnchorPoint(ccp(0.5f, 1));
        controlButton->setPosition(ccp(screenSize.width / 2.0f, screenSize.
            height / 2.0f));
        addChild(controlButton, 1);
        // 定义其他控件，代码省略
        ...
        // 加入回调函数
        controlButton->addTargetWithActionForControlEvent(this, cccontrol_
            selector(CCControlButtonTest_Event::touchDownAction),
            CCControlEventTouchDown);
        controlButton->addTargetWithActionForControlEvent(this, cccontrol_
            selector(CCControlButtonTest_Event::touchDragInsideAction),
            CCControlEventTouchDragInside);
```

```
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchDragOutsideAction), CCContr
                olEventTouchDragOutside);
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchDragEnterAction),
                CCControlEventTouchDragEnter);
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchDragExitAction),
                CCControlEventTouchDragExit);
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchUpInsideAction),
                CCControlEventTouchUpInside);
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchUpOutsideAction),
                CCControlEventTouchUpOutside);
            controlButton->addTargetWithActionForControlEvent(this, cccontrol_
                selector(CCControlButtonTest_Event::touchCancelAction),
                CCControlEventTouchCancel);
            return true;
        }
        return false;
    }
```

要定义按钮对象，首先调用 create 函数。该函数传入的参数可以是图片的路径，也可以是精灵对象。两个参数代表的分别是按钮标题字和背景。之后可以设置按钮的参数，包括设置按钮位置和加入定义的回调函数等。这里的回调函数可以有多个，根据需要的操作定义，包括按下、拖动和抬起等。运行效果如图 3-22 所示。

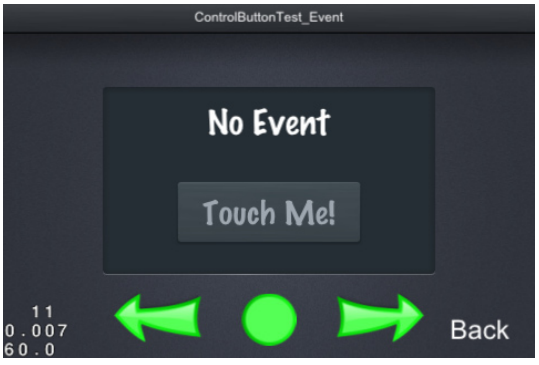

图 3-22　CCControlButton 运行效果

本节介绍了 Cocos2D-x 的布景层类和它的子类，下节开始介绍精灵类及其子类。

3.5　精灵类

精灵类 CCSprite 是一张二维的图片对象，它可以用一张图片或者一张图片的一块矩形

部分来定义。CCSprite 和它的子类可以作为精灵批处理类的子项。它的继承关系如图 3-23 所示。

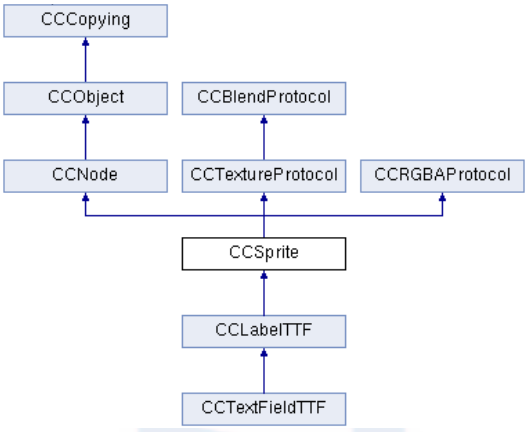

图 3-23 CCSprite 类的继承关系

3.5.1 CCSprite 类的成员数据及函数

CCSprite 类主要保护的成员数据如表 3-10 所示。

表 3-10 CCSprite 类主要保护的成员数据

名 称	类 型	描 述
m_bHasChildren	布尔型	是否有子节点
m_bShouldBeHidden	布尔型	是否被隐藏
m_obRect	矩形	长宽构成的矩形
m_bRectRotated	布尔型	矩形是否旋转
m_obOffsetPosition	点坐标	偏移位置
m_obUnflippedOffsetPositionFromCenter	点坐标	从中心位置的非平移偏移
m_sColorUnmodified	颜色	颜色 RGB 值
m_bOpacityModifyRGB	布尔型	是否不透明
m_bFlipX	布尔型	是否 x 轴镜像
m_bFlipY	布尔型	是否 y 轴镜像
m_bDirty	布尔型	是否需要更新
m_bRecursiveDirty	布尔型	是否需要递归的更新
m_sColor	颜色	颜色 RGB 值
m_nOpacity	不透明度	不透明度

CCSprite 类的主要公共函数如表 3-11 所示。

表 3-11 CCSprite 类的主要公共函数

函 数 名	返回类型	描 述
initWithTexture	布尔型	通过贴图定义精灵
initWithSpriteFrame	布尔型	通过 CCSpriteFrame 定义精灵
initWithSpriteFrameName	布尔型	通过 CCSpriteFrame 名称定义精灵
initWithFile	布尔型	通过文件路径定义
setTextureRect	空	设置贴图矩形
displayFrame	精灵帧	获得当前精灵帧
setDisplayFrame	空	设置当前显示帧
isFrameDisplayed	布尔型	当前是否显示此显示帧
getBatchNode	精灵批处理	获得精灵批处理节点
setBatchNode	空	设置精灵批处理节点
setDisplayFrameWithAnimationName	空	通过动画名称和索引设置显示帧
getTextureAtlas	贴图集	获得贴图集
setTextureAtlas	空	设置贴图集
setFlipX	空	设置 x 轴的镜像
setFlipY	空	设置 y 轴的镜像

这里需要说明的是，纹理贴图集是将我们需要的部分图片放在一张大小固定的图片，可以节约内存。因为 OpenGL 机制会把单张图处理成相应大小的长宽都是 2 的 n 次方的图片，所以把图片放在一起可以节约空间。关于如何制作纹理贴图集，你将会在后面的章节里看到。

3.5.2 贴图类 CCTexture2D

贴图类 CCTexture2D 是关于 OpenGL 的概念。在 OpenGL 中称图片为贴图，在 Cocos2D-x 中 CCTexture2D 就是图片对象的意思，可以通过它创建精灵等对象。CCTexture2D 类的继承关系如图 3-24 所示。

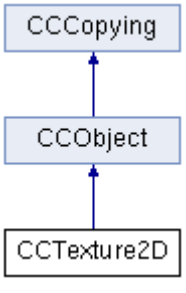

图 3-24 CCTexture2D 类的继承关系

CCTexture2D 类的主要函数如表 3-12 所示。

表 3-12 CCTexture2D 类的主要函数

函 数 名	返回类型	描 述
initWithString	布尔型	通过字符串定义 CCTexture2D 类
initWithImage	布尔型	通过图片路径定义 CCTexture2D 类
initWithPVRFile	布尔型	通过 PVR 图片路径定义 CCTexture2D 类

CCTexture2D 类是精灵类和其相关类的基础。以下会看到很多类都可以用 CCTexture2D 类定义。

3.5.3 精灵批处理类 CCSpriteBatchNode

当你需要显示两个或两个以上相同的精灵时，如果逐个渲染精灵，每一次渲染都会调用 OpenGL 的函数，因为当系统在屏幕上渲染一张贴图的时候，图形处理硬件必须首先准备渲染，然后渲染图形，最后完成渲染以后的清理工作。以上是每次渲染固定的开销，这样帧率就会下降 15% 左右或者更多。

如果将所有需要渲染的同一张贴图只进行一次准备，一次渲染，一次清理就可以解决这个问题了。这时可以使用 CCSpriteBatchNode 类来批处理这些精灵，比如游戏屏幕中的子弹等就可以这样做。用它作为父层来创建子精灵，并且使用它来管理精灵类，这样可以提高程序的效率。CCSpriteBatchNode 类的继承关系如图 3-25 所示。

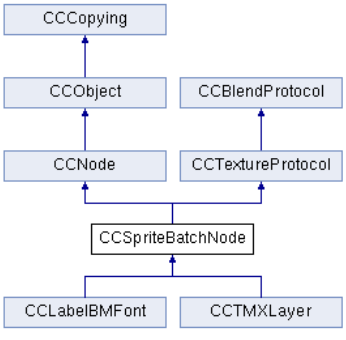

图 3-25 CCSpriteBatchNode 类的继承关系

可以看到，CCSpriteBatchNode 类继承于节点类和贴图协议。

这里需要说明的是，加入 CCSpriteBatchNode 类的精灵类越多，提高效率的效果就越明显。不过也有一些限制，所有属于同一个 CCSpriteBatchNode 类的精灵类都有相同的深度值，也就是说，如果需要呈现一个子弹在人物前面、另外一个子弹在人物后面的不同遮挡关系，获得每个子精灵并单独设置和重排序它们，尽管使用的是同一张贴图，但可以把它们理解为不在同一“层”（并不是布景层）。

此外，所有属于同一个 CCSpriteBatchNode 类控制的精灵类必须使用同一张贴图，但是这并不是一个限制，如果想使用不同的图片，可以把它们放在同一张贴图集当中。

另外还有一些限制，就是 CCSpriteBatchNode 类设置锯齿 / 抗锯齿效果时，所有子精灵也同时设置了锯齿 / 抗锯齿效果，不可以单独设置。同样不能单独设置的还有混合函数（blendfunc）。可以把 CCSpriteBatchNode 类理解为 CCLayer 类，只不过 CCSpriteBatchNode 类只接受 CCSprite 类和它的子类。下面介绍 CCSpriteBatchNode 精灵批处理类。

CCSpriteBatchNode 类的主要函数见表 3-13。

表 3-13 CCSpriteBatchNode 类的主要函数

函 数 名	返回类型	描 述
initWithTexture	布尔型	通过二维贴图来初始化 CCSpriteBatchNode 精灵批处理类，第二个参数是估计的 CCSprite 精灵个数，但是并不会约束你的使用个数
initWithFile	布尔型	通过图片路径（格式可以是 PNG、JPEG、PVR 等）来初始化 CCSpriteBatchNode 精灵批处理类，第二个参数是估计的 CCSprite 精灵个数，但是并不会约束你的使用个数
increaseAtlasCapacity	空	增加贴图集容量
removeSpriteFromAtlas	空	将精灵从贴图集中删除
init	布尔型	初始化

创建方法的第一个参数可以是贴图对象，也可以是图片路径。这里主要说明两个创建方法的第二个参数。这个参数是子节点的数量。当然，如果使用第一种方法不显示的定义子节点的数量，系统会使用默认值 29，在运行时如果超过空间了，会增加 33% 的容量。

下面介绍精灵帧类 CCSpriteFrame。

3.5.4　精灵帧类 CCSpriteFrame

精灵帧的概念是相对于动画而产生的。一个精灵是固定的节点，它可以拥有许多精灵帧（CCSpriteFrame），在它们之间切换就形成了动画。CCSpriteFrame 类的继承关系如图 3-26 所示。

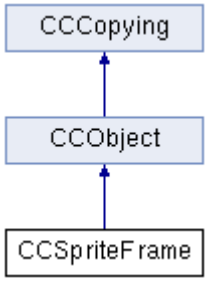

图 3-26　CCSpriteFrame 类的继承关系

CCSpriteFrame 类通过贴图定义，也可以是贴图的一部分，可以通过精灵的 setDisplayFrame 函数来设置当前显示的精灵帧。它的主要函数见表 3-14。

表 3-14　CCSpriteFrame 类的主要函数

函 数 名	返回类型	描　　述
initWithTextureFilename	布尔型	通过图片路径（格式可以是 PNG、JPEG、PVR 等），第二个参数是矩形范围，也就是精灵帧的大小。还有一种是可选的，第三个参数设置是否旋转，第四个参数设置起始点的偏移位置和被裁减之前的原始大小
initWithTexture	布尔型	通过贴图定义，第二个参数是矩形范围，也就是精灵帧的大小。还有一种是可选的，第三个参数设置是否旋转，第四个参数设置起始点的偏移位置和被裁减之前的原始大小
getOriginalSize	尺寸	获得被裁减前的原始大小
setOriginalSize	空	设置被裁减前的原始大小

3.5.5　精灵帧缓存类 CCSpriteFrameCache

精灵帧缓存类 CCSpriteFrameCache 用来存储精灵帧，提前缓存起来有助于提高程序的效率。 CCSpriteFrameCache 是一个单例模式，不属于某个精灵，是所有精灵共享使用的。CCSpriteFrameCache 类的继承关系如图 3-27 所示。

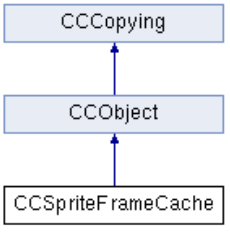

图 3-27　CCSpriteFrameCache 类的继承关系

CCSpriteFrameCache 类的主要函数见表 3-15。

表 3-15　CCSpriteFrameCache 类的主要函数

函 数 名	返回类型	描　　述
addSpriteFramesWithFile	空	第一个参数是贴图集 plist 文件的路径名。可以是默认贴图，也可以通过第二个参数定义贴图使用贴图对象或路径
addSpriteFrame	空	通过 CCSpriteFrame 精灵帧定义，第二个参数是自定义的名称
removeSpriteFrames	空	清空精灵帧
removeUnusedSpriteFrames	空	删除不用的精灵帧

（续）

函 数 名	返回类型	描 述
spriteFrameByName	精灵帧	根据定义的名称找到精灵帧，如果没有对应的，返回空
removeSpriteFrameByName	空	通过名称删除精灵帧

学了这么多基础知识，你可能会有些厌倦了，下一节就看看使用的例子。

3.5.6 实例：精灵类及其相关类的使用

tests 项目中的 SpriteTest 目录下是关于精灵类的使用示例。首先是定义 CCSprite 类的使用方法，SpriteTest.cpp 文件中 Sprite1 类中的 addNewSpriteWithCoords 函数，如代码清单 3-29 所示。

代码清单 3-29 CCSprite 类的使用方法

```
void Sprite1::addNewSpriteWithCoords(CCPoint p)
{
    int idx = (int)(CCRANDOM_0_1() * 1400.0f / 100.0f);
    int x = (idx%5) * 85;
    int y = (idx/5) * 121;

    CCSprite* sprite = CCSprite::create("Images/grossini_dance_atlas.png",
        CCRectMake(x,y,85,121) );
    addChild( sprite );

    sprite->setPosition( ccp( p.x, p.y) );
    //后面是动作的定义，这里暂时不讨论，代码省略
    ...
}
```

这段代码很清晰，首先确定在贴图中取图片的位置，然后根据这个位置选取图片并定义。第一个参数是图片地址，第二个参数是矩形，前两个参数起点为取图的起点横纵坐标，后两个参数为矩形宽高。运行效果如图 3-28 所示。

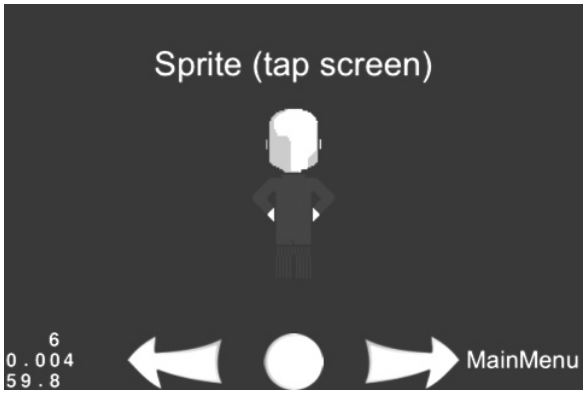

图 3-28 添加精灵类运行效果

下面来看新建精灵批处理类 CCSpriteBatchNode 的示例，出自 tests 项目，SpriteTest 目录下 SpriteTest.cpp 文件中 SpriteBatchNode1 类中的构造函数和 addNewSpriteWithCoords 方法如代码清单 3-30 所示。

代码清单 3-30　新建 CCSpriteBatchNode 类的示例

```
SpriteBatchNode1::SpriteBatchNode1()
{
    setTouchEnabled( true );

    CCSpriteBatchNode* BatchNode = CCSpriteBatchNode::create("Images/grossini_
        dance_atlas.png", 50);
    addChild(BatchNode, 0, kTagSpriteBatchNode);

    CCSize s = CCDirector::sharedDirector()->getWinSize();
    addNewSpriteWithCoords( ccp(s.width/2, s.height/2) );
}
void SpriteBatchNode1::addNewSpriteWithCoords(CCPoint p)
{
    CCSpriteBatchNode* BatchNode = (CCSpriteBatchNode*) getChildByTag(
        kTagSpriteBatchNode );

    int idx = CCRANDOM_0_1() * 1400 / 100;
    int x = (idx%5) * 85;
    int y = (idx/5) * 121;

    CCSprite* sprite = CCSprite::create(BatchNode->getTexture(),
        CCRectMake(x,y,85,121));
    BatchNode->addChild(sprite);

    sprite->setPosition( ccp( p.x, p.y) );
    //后面是动作的定义，这里暂时不讨论，代码省略
    ...
}
```

首先看 SpriteBatchNode1 的构造函数，定义 CCSpriteBatchNode 类，第一个参数是贴图路径，第二个参数是预估的 CCSprite 类个数。在 addNewSpriteWithCoords 方法中把新建的方法作为子节点加入 CCSpriteBatchNode 类中，运行效果如图 3-29 所示。

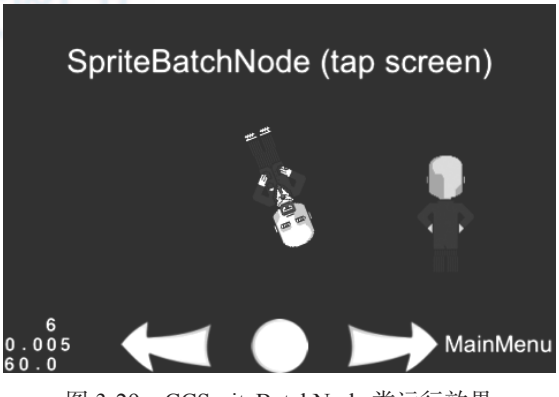

图 3-29　CCSpriteBatchNode 类运行效果

下面分别看一下精灵类和精灵批处理类改变 z 轴顺序并改变遮挡关系的方法。首先看精灵类改变 z 轴顺序并改变遮挡关系的方法，如代码清单 3-31 所示。

代码清单 3-31　精灵类改变 z 轴顺序并改变遮挡关系的方法

```
void SpriteZOrder::reorderSprite(float dt)
{
```

```
    CCSprite* sprite = (CCSprite*)(getChildByTag(kTagSprite1));
    int z = sprite->getZOrder();
    if( z < -1 )
        m_dir = 1;
    if( z > 10 )
        m_dir = -1;
    z += m_dir * 3;
    reorderChild(sprite, z);
}
```

首先通过 getZOrder 方法获得目前的 z 轴值，然后改变后父节点调用 reorderChild 函数，第一个参数是精灵对象，第二个参数是设置的 z 轴值。

下面来看精灵批处理类改变 z 轴顺序并改变遮挡关系的方法，如代码清单 3-32 所示。

代码清单 3-32 精灵批处理类改变 z 轴顺序并改变遮挡关系的方法

```
void SpriteBatchNodeZOrder::reorderSprite(float dt)
{
    CCSpriteBatchNode* batch= (CCSpriteBatchNode*)(getChildByTag(
        kTagSpriteBatchNode ));
    CCSprite* sprite = (CCSprite*)(batch->getChildByTag(kTagSprite1));
    int z = sprite->getZOrder();
    if( z < -1 )
        m_dir = 1;
    if( z > 10 )
        m_dir = -1;
    z += m_dir * 3;
    batch->reorderChild(sprite, z);
}
```

两者的区别就是，这次父节点就是精灵批处理类调用 reorderChild 函数。运行效果如图 3-30 所示。

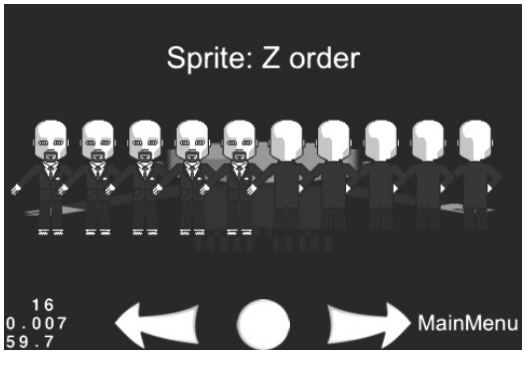

图 3-30 改变 z 轴遮挡关系的运行效果

精灵类设置锯齿和抗锯齿的方法，如代码清单 3-33 所示，出自 tests 项目中的 SpriteTest 目录下 SpriteTest.cpp 文件中的 SpriteAliased 类中的 onEnter 和 onExit 函数。

代码清单 3-33　精灵类设置锯齿和抗锯齿的方法

```
void SpriteAliased::onEnter()
{
    SpriteTestDemo::onEnter();
    CCSprite* sprite = (CCSprite*)getChildByTag(kTagSprite1);
    sprite->getTexture()->setAliasTexParameters();
}
void SpriteAliased::onExit()
{
    CCSprite* sprite = (CCSprite*)getChildByTag(kTagSprite1);
    sprite->getTexture()->setAntiAliasTexParameters();
    SpriteTestDemo::onExit();
}
```

首先精灵类通过 getTexture 获得贴图，分别调用 setAliasTexParameters 设置锯齿，调用 setAntiAliasTexParameters 设置抗锯齿。

提示　为什么要设置抗锯齿呢？因为受分辨率的制约，在渲染物体时，被绘制的物体边缘总会或多或少地呈现三角形的锯齿，抗锯齿对图像边缘进行柔化处理，使图像边缘看起来更平滑，更接近实物的物体。因为默认是抗锯齿，所以抗锯齿不用设置，锯齿需要在进入和出场景时设置。

运行效果如图 3-31 所示，左边的是锯齿，右边的是抗锯齿。二者的区别可以在相对较低分辨率的设备上明显看到。

图 3-31　精灵类设置锯齿和抗锯齿的运行效果

代码清单 3-34 是定义和使用 CCSpriteFrameCache 类的函数，出自 tests 项目，是 SpriteTest 目录下 SpriteTest.cpp 文件中的 SpriteFrameTest 类中的 onEnter 函数。

代码清单 3-34　定义和使用 CCSpriteFrameCache 类的方法

```
CCSpriteFrameCache* cache = CCSpriteFrameCache::sharedSpriteFrameCache();
    cache->addSpriteFramesWithFile("animations/grossini.plist");
```

```
    cache->addSpriteFramesWithFile("animations/grossini_gray.plist", "animations/
        grossini_gray.png");
    cache->addSpriteFramesWithFile("animations/grossini_blue.plist", "animations/
        grossini_blue.png");
    m_pSprite1 = CCSprite::createWithSpriteFrameName("grossini_dance_01.png");
    m_pSprite1->setPosition( ccp( s.width/2-80, s.height/2) );
    CCSpriteBatchNode* spritebatch = CCSpriteBatchNode::create("animations/
        grossini.png");
    spritebatch->addChild(m_pSprite1);
    addChild(spritebatch);

    CCArray* animFrames = CCArray::create(15);

    char str[100] = {0};
    for(int i = 1; i < 15; i++)
    {
        sprintf(str, "grossini_dance_%02d.png", i);
        CCSpriteFrame* frame = cache->spriteFrameByName( str );
        animFrames->addObject(frame);
    }
```

首先是通过贴图集的 plist 文件和图的路径传入 addSpriteFramesWithFile 函数定义，然后通过 spriteFrameByName 传入图片名称获得精灵帧缓存类 CCSpriteFrameCache 对象。要在 onExit 函数中删除这些精灵帧缓存，如代码清单 3-35 所示。

代码清单 3-35　删除精灵帧缓存

```
void SpriteFrameTest::onExit()
{
    SpriteTestDemo::onExit();
    CCSpriteFrameCache *cache = CCSpriteFrameCache::sharedSpriteFrameCache();
    cache->removeSpriteFramesFromFile("animations/grossini.plist");
    cache->removeSpriteFramesFromFile("animations/grossini_gray.plist");
    cache->removeSpriteFramesFromFile("animations/grossini_blue.plist");
}
```

调用 removeSpriteFramesFromFile 删除即可。

下面来看精灵帧类的使用方法。首先是创建精灵帧类，如代码清单 3-36 所示，出自 tests 项目中 SpriteTest 目录下 SpriteTest.cpp 文件中 SpriteAnimationSplit 类中的构造函数。

代码清单 3-36　创建精灵帧类

```
CCSpriteFrame *frame0 = CCSpriteFrame::create(texture, CCRectMake(132*0, 132*0,
    132, 132));
    CCSpriteFrame *frame1 = CCSpriteFrame::create(texture, CCRectMake(132*1,
        132*0, 132, 132));
    CCSpriteFrame *frame2 = CCSpriteFrame::create(texture, CCRectMake(132*2,
        132*0, 132, 132));
    CCSpriteFrame *frame3 = CCSpriteFrame::create(texture, CCRectMake(132*3,
```

```
        132*0, 132, 132));
    CCSpriteFrame *frame4 = CCSpriteFrame::create(texture, CCRectMake(132*0,
        132*1, 132, 132));
    CCSpriteFrame *frame5 = CCSpriteFrame::create(texture, CCRectMake(132*1,
        132*1, 132, 132));
    CCSprite* sprite = CCSprite::create(frame0);
```

函数通过 create 函数创建，并用第 0 帧创建精灵类。删除无用的精灵帧如代码清单 3-37 所示，出自 SpriteAnimationSplit 类的 onExit 函数。

代码清单 3-37 删除无用的精灵帧

```
void SpriteAnimationSplit::onExit()
{
    SpriteTestDemo::onExit();
    CCSpriteFrameCache::sharedSpriteFrameCache()->removeUnusedSpriteFrames();
}
```

获得精灵帧缓存单例，并调用 removeUnusedSpriteFrames 就可以删除无用的精灵帧。下一节介绍 Cocos2D-x 中的摄像机类。

3.6 摄像机类

所有节点都拥有一个摄像机类 CCCamera。只有通过摄像机类，节点才会被渲染出来。当节点发生缩放旋转和位置变化的时候，都需要覆盖 CCCamera 类，让这个节点通过 CCCamera 类重新渲染。

注意 CCNode 类里有些方法可以实现缩放、旋转和位置变化，当使用摄像机类实现这些的时候，那些方法就不能同时使用了。使用摄像机类也不可以同时使用世界坐标了。

CCCamera 类的继承关系如图 3-32 所示。

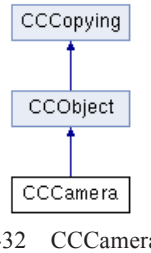

图 3-32 CCCamera 类的继承关系

Cocos2D-x 中的 CCCamera 类使用 OpenGL 的 gluLookAt 函数来设置位置。gluLookAt 函数有三组关于坐标的参数，其中“Eye”系列的 x、y、z 坐标参数是视角的位置，而“Center”系列的 x、y、z 坐标参数是所视目标的坐标位置，“Up”系列的 x、y、z 坐标参数是摄像机方向的向量坐标。关于这三个参数，你可以理解为以“Eye”为起点，沿着“Up”方向，朝“Center”看。以下分别是 CCCamera 类的成员数据和函数。

注意 这里需要说明的是，在三维效果中，使用 CCCamera 类是可以的，但是如果你只需要一些二维特效的话，那么更推荐跟随类 CCFollow。跟随类 CCFollow 的相关知识，将在第 4 章介绍。

3.6.1 CCCamera 类的成员数据

CCCamera 类的主要保护成员数据，如表 3-16 所示。

表 3-16 CCCamera 类的主要保护成员数据

名 称	类 型	描 述
m_fEyeX	浮点数	视角位置坐标 x
m_fEyeY	浮点数	视角位置坐标 y
m_fEyeZ	浮点数	视角位置坐标 z
m_fCenterX	浮点数	目标位置坐标 x
m_fCenterY	浮点数	目标位置坐标 y
m_fCenterZ	浮点数	目标位置坐标 z
m_fUpX	浮点数	摄像机方向坐标 x
m_fUpY	浮点数	摄像机方向坐标 y
m_fUpZ	浮点数	摄像机方向坐标 z

3.6.2 CCCamera 类的函数

CCCamera 类的主要公共函数如表 3-17 所示。

表 3-17 CCCamera 类的主要公共函数

函 数 名	返回类型	描 述
init	空	初始化
restore	空	重置为初始位置
locate	空	使用 OpenGL 的 gluLookAt 函数来设置
setEyeXYZ	空	设置视角位置坐标
getEyeXYZ	空	获得视角位置坐标
setCenterXYZ	空	设置目标位置坐标
getCenterXYZ	空	获得目标位置坐标
setUpXYZ	空	设置摄像机方向坐标
getUpXYZ	空	获得摄像机方向坐标

3.6.3 实例：CCCamera 类的使用

CCCamera 类可以实现节点对象的缩放旋转等，在 tests 项目的 TestNode 文件夹下，TestNode.cpp 文件中的 CameraZoomTest 类中就有使用摄像机类的实例，如代码清单 3-38 所示。

代码清单 3-38　使用摄像机类实现缩放的实例

```
void CameraZoomTest::update(float dt)
{
    CCNode *sprite;
    CCCamera *cam;

    m_z += dt * 100;

    sprite = getChildByTag(20);
    // 获得摄像机
    cam = sprite->getCamera();
    // 设置 z 轴位置
    cam->setEyeXYZ(0, 0, m_z);

    sprite = getChildByTag(40);
    cam = sprite->getCamera();
    cam->setEyeXYZ(0, 0, -m_z);
}
```

可以通过 getCamera 函数获得摄像机实例，并通过设置视角的 z 轴位置来实现缩放效果，运行效果如图 3-33 所示。

需要再次强调的是，不推荐使用摄像机实现二维特效，本实例只是一个使用的范例。下一节介绍 Cocos2D-x 中的容器类。

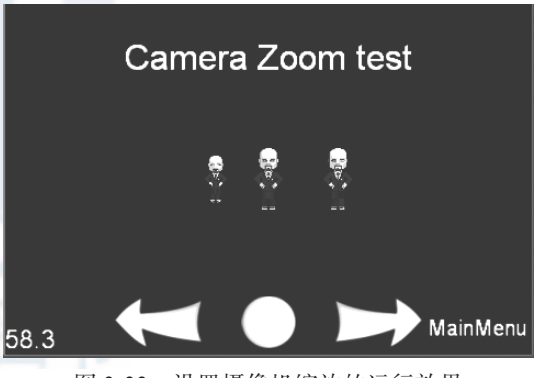

图 3-33　设置摄像机缩放的运行效果

3.7　容器类

之前已经说过了，为了方便从 Cocos2D-iPhone 移植到 Cocos2D-x，Cocos2D-x 引擎底层实现了一些 Objective-C 语言框架中的容器类，包括 CCMutableArray、CCArray、CCMutableDictionary 和 CCDictionary。在 Cocos2D-x 2.0 版本以后，CCMutableArray 和 CCMutableDictionary 继承 STL（Standard Template Library，标准模板库）。相比之下，继承 UTHash（哈希表的宏实现）的 CCArray 和 CCDictionary 效率更高，并且功能也有所增加，更方便绑定 JavaScript 脚本。

3.7.1　CCMutableArray 和 CCArray

之前已经说过了，Cocos2D-x 2.0 以后的版本已经不支持 CCMutableArray 了，本书讨论 CCMutableArray 只是为了使读者看到之前的游戏代码时不会疑惑。

可以把 CCMutableArray 理解成是一个数组的容器，装载的对象只要是 CCObject 的子类就可以。CCMutableArray 的常用函数如表 3-18 所示。

表 3-18 CCMutableArray 的常用函数

函 数 名	返回类型	描 述
CCMutableArray	构造函数	构造函数参数是长度
count	整数	数组长度
getIndexOfObject	整数	获得相应对象的索引
containsObject	布尔型	是否含有相应对象
getLastObject	对象	获得最后一个对象
getObjectAtIndex	对象	根据索引获得对象
addObject	空	增加对象
addObjectsFromArray	空	将一个数组添加进来
insertObjectAtIndex	空	插到某个索引上
removeLastObject	空	删除最后一个对象
removeObject	空	删除对象
removeObjectsInArray	空	删除整个数组的对象
removeObjectAtIndex	空	删除相应索引的对象
removeAllObjects	空	删除所有对象
replaceObjectAtIndex	空	替换相应索引位置的对象
begin	迭代器	获得开始项迭代器
end	迭代器	获得结束项迭代器

CCMutable Array 定义函数的使用如代码清单 3-39 所示。

代码清单 3-39 CCMutableArray 定义函数

```
CCMutableArray<AstarItem*> open = new CCMutableArray<AstarItem*>();
```

其中 AstarItem 为自己定义的类，继承自 CCObject，使用时调用相应函数就可以了。

Cocos2D-x 2.0 以后的版本，CCArray 取代了 CCMutableArray。CCArray 的常用函数如表 3-19 所示。

表 3-19 CCArray 的常用函数

函 数 名	返回类型	描 述
init	布尔型	初始化
initWithObject	布尔型	初始化，参数为对象
initWithObjects	布尔型	初始化，参数为多个对象
initWithCapacity	布尔型	初始化，参数为个数
initWithArray	布尔型	初始化，参数为数组
count	整数	目前对象个数
capacity	整数	数组对象
indexOfObject	整数	获得相应对象的索引

（续）

函 数 名	返回类型	描 述
objectAtIndex	对象	根据索引获得对象
lastObject	对象	返回最后一个对象
randomObject	对象	随机返回一个对象
addObject	空	增加对象
addObjectsFromArray	空	将一个数组添加进来
insertObjectAtIndex	空	插到某个索引上
removeLastObject	空	删除最后一个对象
removeObject	空	删除对象
removeObjectsInArray	空	删除整个数组的对象
reverseObjects	空	颠倒对象
exchangeObject	空	交换两个对象，参数为对象
exchangeObjectAtIndex	空	交换两个对象，参数为索引
fastRemoveObject	空	快速删除一个确定的对象
fastRemoveObjectAtIndex	空	根据索引快速删除一个对象

CCArray 定义函数的使用如代码清单 3-40 所示。

代码清单 3-40 CCArray 定义函数

```
CCArray *newArray = new CCArray(m_pControlPoints->count());
```

可以发现，它不用确定存储对象的类型，因此每个对象的类型可以不相同，这也是 CCArray 的一个特点。

3.7.2 CCMutableDictionary 和 CCDictionary

CCMutableDictionary 和 CCDictionary 类似于哈希表的键值对应的容器，同样，在 Cocos2D-x 2.0 之前的版本有 CCMutableDictionary，2.0 以后的版本使用 CCDictionary。CCMutableDictionary 的常用函数如表 3-20 所示。

表 3-20 CCMutableDictionary 的常用函数

函 数 名	返回类型	描 述
CCMutableDictionary	构造函数	构造函数
count	整数	项目个数
allKeys	Vector 数组	所有的键值
allKeysForObject	Vector 数组	所有键值对应的对象
objectForKey	对象	键值对应的对象
setObject	布尔型	添加项，第一个参数是键，第二个是对应的值
removeObjectForKey	空	根据键删除项

CCMutableDictionary 定义函数的使用如代码清单 3-41 所示。

代码清单 3-41 CCMutableDictionary 的定义函数

```
CCMutableDictionary<string,CCString *> *tiledic = map->propertiesForGID(tilegid);
```

CCDictionary 的常用函数如表 3-21 所示。

表 3-21 CCDictionary 的常用函数

函 数 名	返回类型	描 述
CCDictionary	构造函数	构造函数
count	整数	项目个数
allKeys	CCArray 数组	所有的键值
allKeysForObject	CCArray 数组	所有键值对应的对象
objectForKey	对象	键值对应的对象
setObject	布尔型	添加项，第一个参数是对应的对象，第二个是键，键可以是字符串，也可以是整数
removeObjectForKey	空	根据键删除项
removeObjectsForKeys	空	根据键值数组删除项
removeObjectForElememt	空	根据元素删除相应对象
removeAllObjects	空	删除所有项

CCDictionary 定义函数的使用如代码清单 3-42 所示。

代码清单 3-42 CCDictionary 的定义函数

```
CCDictionary* pRet = new CCDictionary();
```

3.8 拖动渐隐效果类 CCMotionStreak

在游戏的实现过程中，有时会需要在某个游戏对象上的运动轨迹上实现渐隐效果。这种感觉就好像是类似飞机拉线的拖尾巴，在视觉上感觉很好，比如子弹的运动轨迹等，如果不借助引擎的帮助，这种效果往往需要通过大量的图片来实现。而 Cocos2D-x 提供了一种内置的拖动渐隐效果类 CCMotionStreak 来帮助我们实现这个效果。它是 CCNode 类的子类，继承关系如图 3-34 所示。

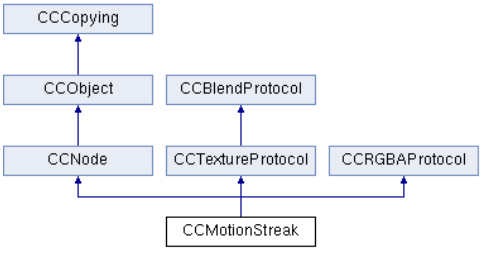

图 3-34 CCMotionStreak 类的继承关系

CCMotionStreak 类的常用函数如表 3-22 所示。

表 3-22　CCMotionStreak 类的常用函数

函 数 名	返回类型	描　　述
initWithFade	布尔型	第一个参数是间隐的时间，第二个参数是间隐片断的大小，第三个参数是贴图的宽高，第四个参数是颜色值 RGB，第五个参数是贴图的路径或者贴图对象
tintWithColor	空	定义闪烁颜色
setFastMode	空	设置为快速模式，快速模式中，新的点的增加会更快，但是精度会降低
isFastMode	布尔型	返回是否是快速模式

以下示例出自 tests 项目中的 MotionStreakTest 文件夹下的 MotionStreakTest.cpp 文件，其中的 MotionStreakTest2 类如代码清单 3-43 所示。

代码清单 3-43　定义 CCMotionStreak 对象

```
void MotionStreakTest2::onEnter()
{
    MotionStreakTest::onEnter();

    setTouchEnabled(true);

    CCSize s = CCDirector::sharedDirector()->getWinSize();

    streak = CCMotionStreak::create(3, 3, 64, ccWHITE, s_streak );
    addChild(streak);

    streak->setPosition( CCPointMake(s.width/2, s.height/2) );
}



void MotionStreakTest2::ccTouchesMoved(CCSet* touches, CCEvent* event)
{
    CCSetIterator it = touches->begin();
    CCTouch* touch = (CCTouch*)(*it);

    CCPoint touchLocation = touch->locationInView();
    touchLocation = CCDirector::sharedDirector()->convertToGL( touchLocation );

    streak->setPosition( touchLocation );
}
```

以上代码使用 create 函数创建 CCMotionStreak 对象，每次调用 setPosition 函数重新设置对象位置时，“影子”将被创建并且慢慢渐隐，运行效果如图 3-35 所示。

图 3-35 拖动渐隐效果运行效果

3.9 绘制图形

在节点类CCNode中，可以重写draw函数并在其中绘制图形，如tests项目中DrawPrimitivesTest文件夹下DrawPrimitivesTest.cpp文件中的DrawPrimitivesTest类中的draw函数。

1）绘制直线。

参数分别为直线的起点和终点。如代码清单3-44所示。

代码清单 3-44 绘制直线

```
ccDrawLine( ccp(0, 0), ccp(s.width, s.height) );
```

默认绘制的直线是白色的、不透明的、线宽是1并且是抗锯齿的。设置这些绘制参数后，绘制直线如代码清单3-45所示。

代码清单 3-45 设置绘制参数后绘制直线

```
glLineWidth( 5.0f );
ccDrawColor4B(255,0,0,255);
ccDrawLine( ccp(0, s.height), ccp(s.width, 0) );
```

这里需要说明的是，OpenGL中是状态机，除非重新设置，否则这些参数都会保持到下一个状态。

2）绘制点。

如代码清单3-46所示，注意OpenGL中的点是正方形的。

代码清单 3-46　绘制点

```
CCPoint points[] = { ccp(60,60), ccp(70,70), ccp(60,70), ccp(70,60) };
ccPointSize(4);
ccDrawColor4B(0,255,255,255);
ccDrawPoints( points, 4);
```

3）画圆。

函数的参数为圆心、半径、角度、分段数（将圆微分为直线）和是否与中心连线，如代码清单 3-47 所示。

代码清单 3-47　画圆

```
glLineWidth(2);
ccDrawColor4B(0, 255, 255, 255);
ccDrawCircle( ccp(s.width/2, s.height/2), 50,     CC_DEGREES_TO_RADIANS(90), 50,
true);
```

4）画多边形。

函数的参数为点数组、点数量和图形是否封闭，如代码清单 3-48 所示。

代码清单 3-48　画多边形

```
ccDrawColor4B(255, 0, 255, 255);
glLineWidth(2);
CCPoint vertices2[] = { ccp(30,130), ccp(30,230), ccp(50,200) };
ccDrawPoly( vertices2, 3, true);
```

5）画贝塞尔曲线。

函数的第一个参数为一个控制点，第二个和第三个参数为两个控制点，最后一个参数为分段数，如代码清单 3-49 所示。

代码清单 3-49　画贝塞尔曲线

```
CHECK_GL_ERROR_DEBUG();

// draw quad bezier path
ccDrawQuadBezier(ccp(0,s.height), ccp(s.width/2,s.height/2), ccp(s.width,s.
    height), 50);

CHECK_GL_ERROR_DEBUG();

// draw cubic bezier path
ccDrawCubicBezier(ccp(s.width/2, s.height/2), ccp(s.width/2+30,s.height/2+50),
    ccp(s.width/2+60,s.height/2-50),ccp(s.width, s.height/2),100);
```

示例的运行效果如图 3-36 所示。

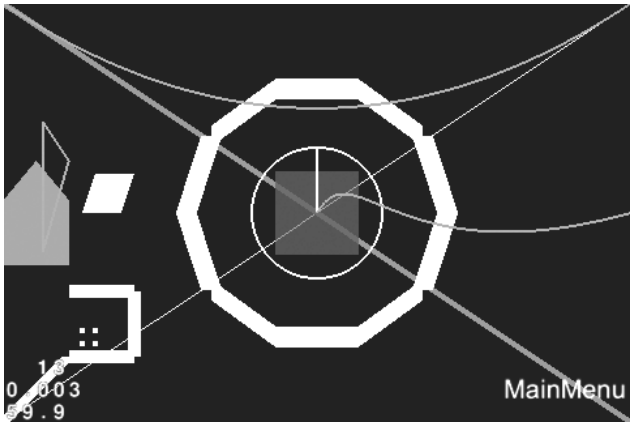

图 3-36 绘制示例运行效果

注意 Cocos2D-x 提供的绘制图形函数里没有绘制实心圆形的函数。需要绘制实心圆形时，请将 ccDrawCircle 函数（在 CCDrawingPrimitives.cpp 文件中）调用 glDrawArrays 函数时的第一个参数由 GL_LINE_STRP 改为 GL_TRIANGLE_FAN 即可。

3.10 时间调度

在游戏中，时常需要隔一段时间更新一些数据或者是人物位置，Cocos2D-x 中提供了这些时间调度的函数，所有 CCNode 类的子类都有这样的函数，定义方法如代码清单 3-50 所示。

代码清单 3-50 schedule 的使用

```
schedule(schedule_selector(SchedulerAutoremove::autoremove), 0.5f);
```

这是一个按时调用一个函数的方法。第一个参数使用 schedule_selector 选择器将 autoremove 函数名称传进来。第二个参数是时间间隔。定义这个参数以后就会隔一段时间调用一次该函数，直到 unschedule 被调用。

unschedule 的使用如代码清单 3-51 所示。

代码清单 3-51 unschedule 的使用

```
unschedule(schedule_selector(SchedulerAutoremove::autoremove));
```

这句被调用，之前 schedule 的时间调度将结束。使用如代码清单 3-52 所示的代码分别暂停并重新启动 schedule。

代码清单 3-52 暂停并重新启动 schedule

```
// 暂停 schedule
m_pPausedTargets = pDirector->getScheduler()->pauseAllTargets();
CC_SAFE_RETAIN(m_pPausedTargets);
// 重新启动 schedule
pDirector->getScheduler()->resumeTargets(m_pPausedTargets);
CC_SAFE_RELEASE_NULL(m_pPausedTargets);
```

使用 unscheduleAllSelectors() 或者如代码清单 3-53 所示的代码，可以使所有 schedule 停止。

代码清单 3-53 使所有 schedule 停止

```
CCDirector::sharedDirector()->getScheduler()->unscheduleAllSelectors();
```

有一种固定的调用方式，就是使用 scheduleUpdate 会在 0.01s 左右调用一次 update 方法，只要重载 update 方法即可。

3.11 本章小结

本章介绍了 Cocos2D-x 的一些基本的类，包括 CCNode 类，它是一个没有贴图的类；后面介绍了构成游戏的 CCScene 类和 CCLayer 类，并且介绍了布景层类的子类；然后介绍了 CCSprite 类和它的相关类，并介绍了渲染节点的关键类 CCCamera 类；最后介绍了容器类和时间调度的内容。

学完本章内容，可以根据示例和知识，自己写一些小程序。第 4 章将介绍动作类和动画方面的知识，你的“精灵”可以动起来了。