### 4.1 UI控件的使用

本节将学习在Cocos2d-x中一些控件的高级用法，如可以拉伸的控件及列表等。前面已经学习了在界面场景中添加Label（标签）、Sprite（精灵）、Menu（菜单）等控件，要实现一些比较复杂的效果时，要么引用额外的库，要么就得靠开发者自己组合。本节学习一些额外的控件，然后再学习几种通过基础控件组合实现的特殊效果。

【本节知识点】

* 九宫格控件的使用方法
* 如何使用九宫格控件实现按钮
* Cocos2d-x中拖动条的实现方法

**（1）九宫格简介**

在游戏界面设计中，通常为如下图所示的结构。通常的界面设计都是对下图9个分栏的简单拉伸和组合。在界面设计师，通常先对下图1、2、7、9的分栏进行定位，然后根据需要对下图中标号为2、4、5、6、8的分栏进行相应的拉伸，最终经过拼接形成一个完整的面板样式。如果此时再加入一些类似物品栏或文字之类的内容，就很自然地成为一个完成的面板了。



这种技术在移动开发中非常常见，又被称为9Patch或者“九妹”，除了能够适应移动设备中的各种尺寸的屏幕以外，很重要的一个优势就是能够节约大量的系统资源。毕竟通过这种方式，仅需要一张图片素材，就能够取代过去需要许多图片素材才能够完成的任务。

尤其是当游戏举办一些活动，比如新年庆典活动时，往往会推出喜庆风格的UI，那么仅需要替换其中的一张图片就可以了，这与传统方式相比，优势是不言而喻的。

**（2）实现九宫格的效果**

通过自定义类NinePatch，来实现一个九宫格效果的控件，它实现九宫格显示栏目背景面板的功能。运行结果如下图所示。



【代码解释】

NinePatch类

|  |  |  |
| --- | --- | --- |
| **序号** | **方法名** | **主要功能** |
| **1** | static NinePatch\* createPatch(int width, int height, int borderSize, Point position); |  |
| **2** | void initPatch(int width, int height, int borderSize, Point position); |  |

代码中的patch1~9分别对应将背景图片拆分开来的9个栏目块。其中范例中create方法中的参数Rect就是制定对应的精灵对象，取目标素材图片的某一部分，其包括4个参数：要取部分的左上角的横、纵坐标、宽度、高度。注意在Rect中，图片的坐标是以左上角为原点的。比如要取某个图片的最左上角宽为50像素、高为100像素的一部分，就应当使用Rect（0,0,50,100）。

然后可以通过这样的方式来为面板固定4个角的位置。

最后填充面板的空白区域，如果发现长度不够，就需要对其进行缩放，下面代码中红色代码就实现的这个功能。最后逐个将面板中的其他栏目通过缩放的方式补齐。

由于是通过缩放而不是简单的拼接来实现的，因此如果素材图片上存在纹理，基本上时能够在面板上再现出来的，而不会出现由于拼接而产生的“断痕”。这也是九宫格方法的一个优点。

【代码】

NinePatch\* NinePatch::createPatch(int width, int height, int borderSize, Point position)

{

NinePatch\* patch = NinePatch::create();

if (patch)

{

patch->initPatch(width, height, borderSize, position);

return patch;

}

CC\_SAFE\_DELETE(patch);

return NULL;

}

void NinePatch::initPatch(int width, int height, int borderSize, Point position)

{

// 左上角

auto\* patch1 = Sprite::create("patch.png", Rect(0, 0, borderSize, borderSize));

patch1->setAnchorPoint(Vec2(0, 0));

patch1->setPosition(position.x, position.y + height - borderSize);

addChild(patch1);

// 右上角

auto\* patch3 = Sprite::create("patch.png", Rect(PATCH\_WIDTH - borderSize, 0, borderSize, borderSize));

patch3->setAnchorPoint(Vec2(0, 0));

patch3->setPosition(position.x + width - borderSize, position.y + height - borderSize);

addChild(patch3);

// 左下角

auto\* patch7 = Sprite::create("patch.png", Rect(0, PATCH\_HEIGHT - borderSize, borderSize, borderSize));

patch7->setAnchorPoint(Vec2(0, 0));

patch7->setPosition(position.x, position.y);

addChild(patch7);

// 右下角

auto\* patch9 = Sprite::create("patch.png", Rect(PATCH\_WIDTH - borderSize, PATCH\_HEIGHT - borderSize, borderSize, borderSize));

patch9->setAnchorPoint(Vec2(0, 0));

patch9->setPosition(position.x + width - borderSize, position.y);

addChild(patch9);

// 顶部分栏

auto\* patch2 = Sprite::create("patch.png", Rect(borderSize, 0, PATCH\_WIDTH - borderSize \* 2, borderSize));

patch2->setAnchorPoint(Vec2(0, 0));

patch2->setPosition(position.x + borderSize, position.y + height - borderSize);

**patch2->setScaleX((float)(width - borderSize \* 2) / (PATCH\_WIDTH - borderSize \* 2));**

addChild(patch2);

// 左侧分栏

auto\* patch4 = Sprite::create("patch.png", Rect(0, borderSize, borderSize, PATCH\_HEIGHT - borderSize \* 2));

patch4->setAnchorPoint(Vec2(0, 0));

patch4->setPosition(position.x, position.y + borderSize);

patch4->setScaleY((float)(height - borderSize \* 2) / (PATCH\_HEIGHT - borderSize \* 2));

addChild(patch4);

// 右侧分栏

auto\* patch6 = Sprite::create("patch.png", Rect(PATCH\_HEIGHT - borderSize, borderSize, borderSize, PATCH\_HEIGHT - borderSize \* 2));

patch6->setAnchorPoint(Vec2(0, 0));

patch6->setPosition(position.x + width - borderSize, position.y + borderSize);

patch6->setScaleY((float)(height - borderSize \* 2) / (PATCH\_HEIGHT - borderSize \* 2));

addChild(patch6);

// 底部

auto\* patch8 = Sprite::create("patch.png", Rect(borderSize, PATCH\_HEIGHT - borderSize, PATCH\_WIDTH - borderSize \* 2, borderSize));

patch8->setAnchorPoint(Vec2(0, 0));

patch8->setPosition(position.x + borderSize, position.y);

patch8->setScaleX((float)(width - borderSize \* 2) / (PATCH\_WIDTH - borderSize \* 2));

addChild(patch8);

// 中间填充部分

auto\* patch5 = Sprite::create("patch.png", Rect(borderSize, borderSize, PATCH\_WIDTH - borderSize\*2, PATCH\_HEIGHT - borderSize\*2));

patch5->setAnchorPoint(Vec2(0, 0));

patch5->setPosition(position.x + borderSize, position.y + borderSize);

patch5->setScale((float)(width - borderSize \* 2) / (PATCH\_WIDTH - borderSize \* 2), (float)(height - borderSize \* 2) / (PATCH\_HEIGHT - borderSize \* 2));

addChild(patch5);

}

**（3）神奇的Scale9Sprite**

前面通过一种用自定义类的方式实现九宫格自动缩放显示面板的方法，但实际上在Cocos2d-x中有一个专门的库：cocos-ext.h，在引用了它之后，可以使用一些在Cocos2d-x中原本没有提供的控件，比如下面要使用的Scale9Sprite，使用方法示例如下。

【代码解释】

Scale9Sprite\* bar = Scale9Sprite::create("patch.png",Rect(0,0,200,200),Rect(50,50,100,100));

创建一个Scale9Sprite类型的对象bar，其中包含3个参数，第一个参数是使用的素材图片的名称；第二个参数用来指定使用素材的哪一部分，该图片的尺寸为200\*200像素，这里是使用整个图片；第三个参数图像的中心部分，即使用Rect类指向9宫格图的数字为5的部分，系统会根据这个尺寸来进行拉伸部分和4个角部分坐标的识别和计算。

bar->setPreferredSize(Size(500, 300));

设置面板的尺寸。

【代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

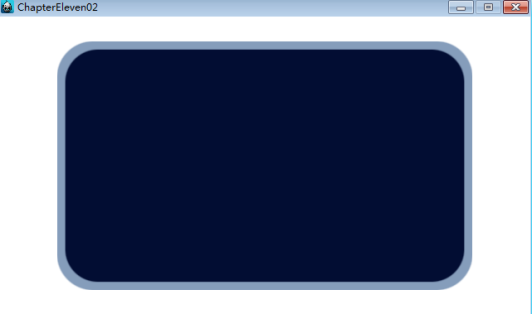
Scale9Sprite\* bar = Scale9Sprite::create("patch.png",Rect(0,0,200,200),Rect(50,50,100,100));

bar->setPreferredSize(Size(500, 300));

bar->setPosition(320, 180);

addChild(bar);

运行程序后的结果如下图：



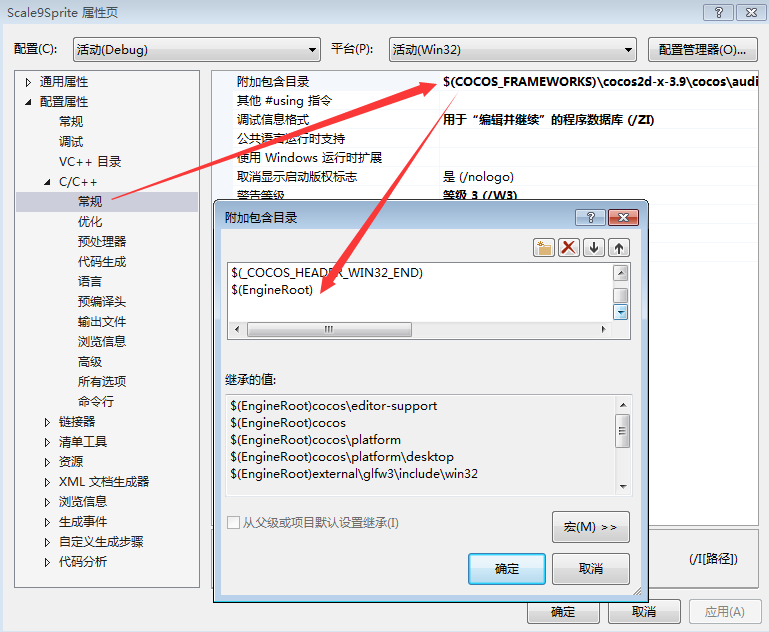
在代码运行时，因为cocos-ext是一个额外库，要使用它需要在项目中将它与相关的方法和类包含进来。

首先需引用cocos-ext.h，并声明其对应的命名空间：

#include "extensions\cocos-ext.h"

USING\_NS\_CC\_EXT;

然后在项目属性下图界面，加入新包含的目录$(EngineRoot)



**（4）Scale9Sprite在按钮中的应用**

除了面板外，实际应用中个，按钮九宫格的使用方式也非常常见，因为游戏中需要用到不同的按钮。为了保证界面风格的移植，虽然按钮的尺寸是不同的，但也使用了这种九宫格的方式来实现。下面的示例我们来学习在按钮中使用Scale9Sprite的方法。运行效果如下：



当按钮被单击时，按钮将放大。

【代码解释】

【代码】

Scale9Sprite\* btnNormal = Scale9Sprite::create("button0.png", Rect(0, 0, 200, 200), Rect(50, 50, 100, 100));

btnNormal->setPreferredSize(Size(500, 300));

Scale9Sprite\* btnPressed = Scale9Sprite::create("button1.png", Rect(0, 0, 200, 200), Rect(50, 50, 100, 100));

btnPressed->setPreferredSize(Size(500, 300));

auto\* label = Label::create("button", "", 40);

auto\* button = ControlButton::create(label, btnNormal);

button->setBackgroundSpriteForState(btnNormal, Control::State::NORMAL);

button->setBackgroundSpriteForState(btnPressed, Control::State::SELECTED);

button->setPosition(320, 180);

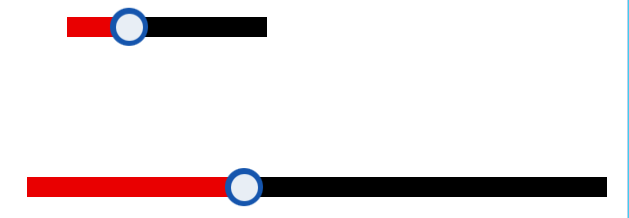
addChild(button);

（5）在Cocos2d-x中使用滑动条

在移动开发中，常用到一种控件，用来控制应用中某些进度，比如视频、音乐的播放进度等。在游戏中则常常用它来设置音量，也常常让它变形来实现对血量等数据的显示。本节学习这种控件的使用。

* 自定义SlideBar实现滑动条

通过编写自定义类SlideBar来实现滑动条效果。



SliderBar类

|  |  |  |
| --- | --- | --- |
| **序号** | **属性或方法名** | **主要功能** |
| **1** | int size; | **滑动条的长度（宽度）** |
| **2** | bool isTouched; | **标记变量** |
| **3** | static SlideBar\* createBar(int size, Point position, int number); | **创建滑动条** |
| **4** | void barInit(int size, Point position, int number); | **初始化滑动条** |
| **5** | virtual bool onTouchBegan(Touch\* touch, Event\* event); | **触摸事件的响应** |
| **6** | virtual void onTouchMoved(Touch\* touch, Event\* event); |  |
| **7** | virtual void onTouchEnded(Touch\* touch, Event\* event); |  |

【创建和调用滑动条的代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

auto\* bar1 = SlideBar::createBar(580, Vec2(40, 110), 200);

addChild(bar1);

auto\* bar2 = SlideBar::createBar(200, Vec2(80, 270), 200);

addChild(bar2);

**【代码解释】**

**auto\* bg = LayerColor::create(Color4B(0, 0, 0, 255), size, 20);**

**创建滑动条的背景**

**auto\* numBar = LayerColor::create(Color4B(233, 0, 1, 255), number, 20);**

**创建滑动条的指示部分（红色）**

**auto\* bar = Sprite::create("bar.png");**

**创建滑动条的滑块**

**根据barInit方法的参数来确定这些对象的排放位置和尺寸。**

**然后定义一个监听器Listener，将其与触摸行为绑定，当用户的手指单击在滑块上时，滑块就会随着用户的手指改变其x坐标，并以此来计算numBar的宽度，从而使其尺寸也发生改变。**

**本例可以实现类似的方法来使滑块不显示，然后可以在游戏中当做怪物的血量条来使用。**

【代码】

SlideBar\* SlideBar::createBar(int size, Point position, int number)

{

SlideBar\* layer = SlideBar::create();

if (layer)

{

layer->barInit(size, position, number);

return layer;

}

CC\_SAFE\_DELETE(layer);

return NULL;

}

void SlideBar::barInit(int size, Point position, int number)

{

this->size = size;

isTouched = false;

auto\* bg = LayerColor::create(Color4B(0, 0, 0, 255), size, 20);

bg->setAnchorPoint(Vec2(0, 0));

bg->setPosition(position);

addChild(bg);

auto\* numBar = LayerColor::create(Color4B(233, 0, 1, 255), number, 20);

numBar->setAnchorPoint(Vec2(0, 0));

numBar->setPosition(position);

numBar->setTag(1);

addChild(numBar);

auto\* bar = Sprite::create("bar.png");

bar->setPosition(position.x + number, position.y + 10);

bar->setTag(2);

addChild(bar);

auto\* listener = EventListenerTouchOneByOne::create();

listener->onTouchBegan = CC\_CALLBACK\_2(SlideBar::onTouchBegan, this);

listener->onTouchMoved = CC\_CALLBACK\_2(SlideBar::onTouchMoved, this);

listener->onTouchEnded = CC\_CALLBACK\_2(SlideBar::onTouchEnded, this);

listener->setSwallowTouches(true);

Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);

}

bool SlideBar::onTouchBegan(Touch\* touch, Event\* event)

{

Sprite\* bar = (Sprite\*)getChildByTag(2);

Point point = touch->getLocation();

if (bar->boundingBox().containsPoint(point))

{

isTouched = true;

return true;

}

return false;

}

void SlideBar::onTouchMoved(Touch\* touch, Event\* event)

{

if (!isTouched)

{

return;

}

auto\* numBar = (Sprite\*)getChildByTag(1);

auto\* bar = (Sprite\*)getChildByTag(2);

auto point = touch->getLocation();

if (point.x > numBar->getPositionX() && point.x < numBar->getPositionX() + size)

{

bar->setPositionX(point.x);

numBar->setContentSize(Size(point.x - numBar->getPositionX(), numBar->getContentSize().height));

}

}

void SlideBar::onTouchEnded(Touch\* touch, Event\* event)

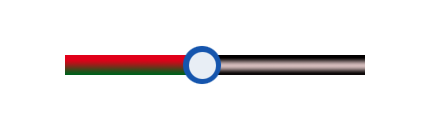
{

isTouched = false;

}

* **使用类ControlSlider来实现滑动条的效果**

在cocos-ext.h中获取相应的控件。这个控件就是ControlSlider。



【代码解释】

auto\* slider = ControlSlider::create("background.png", "progress.png", "bar.png");

首先创建一个ControlSlider类型的对象slider。create方法共有3个参数，分别指向滑动条背景、进度条和滑块所使用的图像资源名。

创建完成之后，继续设置它的位置并将它加入到场景中。此时如果运行程序，滑动条上的滑块是无法移动的。

此时需要对它进行进一步的设置，通过setMaximumValue和setMinimumValue设置滑动条的最大、最小值，还可以通过方法setValue来设置滑动条当前的滑块位置。

【其他说明】如果想要实现一些其他的效果，比如实现游戏中显示血量的功能，只需要给滑块施加一个透明的图像素材就可以了。

【代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

auto\* slider = ControlSlider::create("background.png", "progress.png", "bar.png");

slider->setPosition(320, 180);

addChild(slider);

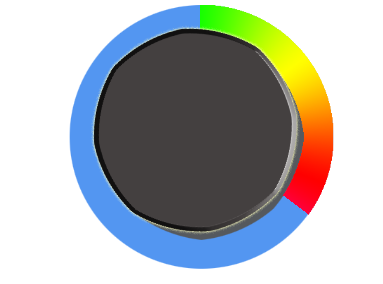
slider->setMaximumValue(100);

slider->setMinimumValue(0);

slider->setValue(30);

**（6）另一种形式的进度条**

本节我们学习一种圆形“进度条”，它也是cocos-ext.h文件提供的控件，运行示例如下图所示。这种进度条的形式为圆形，被应用于秒表或者音量控制等部分，在Cocos2d-x中被称为电位计（ControlPotentiometer）。通过拖动“电位计”中央的按钮部分，改变进度条部分的值。



**【代码解释】**

auto\* controler = ControlPotentiometer::create("background.png", "progress.png", "button.png");

**创建电位计对象时使用了3个参数，分别是电位计的背景、进度条和按钮（对应滑动挑中的滑块）**

**系统怎样响应用户对电位计的操作呢？**

**两种方法：**

**一种是参考之前实现虚拟摇杆所用的方法，使用一个定时器，不断地调用getValue方法获取电位计的值。非常简单但不推荐。**

**另一种方法是使用回调函数change的方式。然后让电位计controler对它进行调用。**

**【代码】**

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

auto\* controler = ControlPotentiometer::create("background.png", "progress.png", "button.png");

controler->setPosition(290, 130);

addChild(controler);

controler->setMinimumValue(0);

controler->setMaximumValue(100);

controler->setScale(0.7f);

controler->setValue(30);

controler->addTargetWithActionForControlEvents(this,

cccontrol\_selector(HelloWorld::change), Control::EventType::VALUE\_CHANGED);

void HelloWorld::change(CCObject \* pSender, Control::EventType event)

{

auto\* controler = (ControlPotentiometer\*)pSender;

float num = controler->getValue();

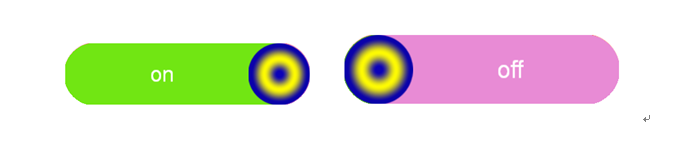
CCLOG("the value of controler is:%f", num);

}

**（7）在游戏中使用开关**

在游戏开发中还会用到一种像现实中的开关一样的控件，这类控件往往被应用于诸如是否静音或者是否使用震动反馈之类的设置界面中，可以由开发者自定义来实现这种开关效果，也可以使用cocos-ext提供的ControlSwitch来完成。

本节示例的运行结果如下图所示。



【代码解释】

auto\* controlSwitch = ControlSwitch::create(switchBG, switchOn, switchOff, switchBar, on, off);

ControlSwitch的create方法一共有6个参数，与之前的电位计和滑动条的create方法不同的是，这里并不能直接使用图像名称作为参数，而需要先将它们转换成对应的精灵对象。

最后两个参数用于显示开关打开和关闭时的文字。

【代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

auto\* switchBG = Sprite::create("background.png");

auto\* switchOn = Sprite::create("on.png");

auto\* switchOff = Sprite::create("off.png");

auto\* switchBar = Sprite::create("button.png");

auto\* on = Label::create("on", "Arial", 36);

auto\* off = Label::create("off", "Arial", 36);

auto\* controlSwitch = ControlSwitch::create(switchBG, switchOn, switchOff, switchBar, on, off);

controlSwitch->setPosition(320, 180);

addChild(controlSwitch);

controlSwitch->addTargetWithActionForControlEvents(this, cccontrol\_selector(HelloWorld::change), Control::EventType::VALUE\_CHANGED);

void HelloWorld::change(Object \* pSender, Control::EventType event)

{

if (event == Control::EventType::VALUE\_CHANGED)

{

auto\* s = (ControlSwitch\*)pSender;

if (s->isOn())

{

CCLOG("the switch is on!");

}

else

{

CCLOG("the switch is off!");

}

}