Cocos2dx 基本界面元素与动作动画调度器

by 何家豪

目 录 / contents

section1

基础界面元素UI

文字Label

菜单Menu

进度条ProgressBar

文字Label

在游戏中，文字挂件是非常常见的。回想一下你所玩过的游戏，在哪些地方使用过文字？

文字Label使用方法及例子

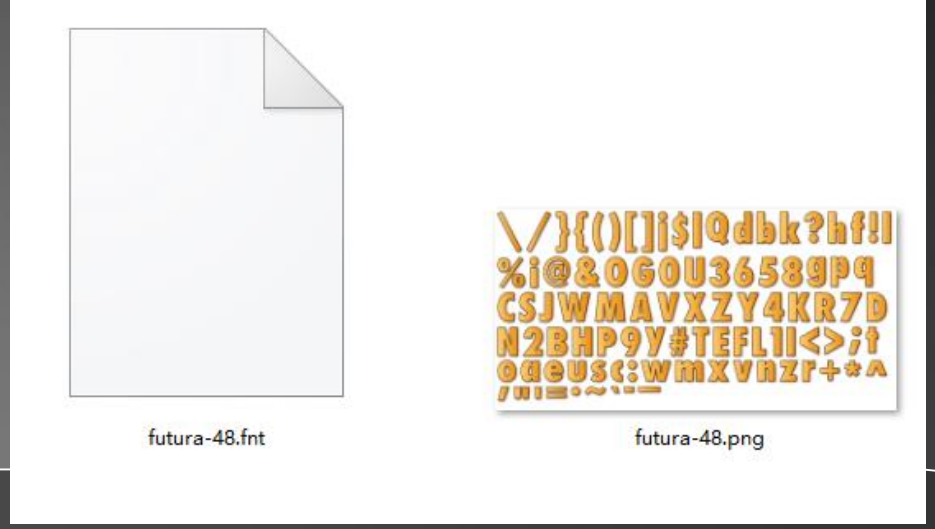
```cpp
auto ttfLabel = Label::createWithTTF("Hello World", "fonts/Marker Felt.ttf", 24);
auto sysLabel = Label::createWithSystemFont("你好 世界", "Microsoft Yahei", 24);
auto bmfLabel = Label::createWithBMFont("fonts/futura-48.fnt", "Hello World");

//use ttf to control the property of the label
TTFConfig ttfConfig;
ttfConfig.fontFilePath = "fonts/Marker Felt.ttf";
ttfConfig.fontSize = 24;
ttfConfig.outlineSize = 2;
auto conLabel = Label::createWithTTF(ttfConfig, "Hello World");

// position the label on the center of the screen
ttfLabel->setPosition(Vec2(origin.x + visibleSize.width/2,
                           origin.y + visibleSize.height - ttfLabel->getContentSize().height));
sysLabel->setPosition(ttfLabel->getPosition().x,
                           ttfLabel->getPosition().y - sysLabel->getContentSize().height);
bmfLabel->setPosition(ttfLabel->getPosition().x,
                           sysLabel->getPosition().y - bmfLabel->getContentSize().height);
conLabel->setPosition(ttfLabel->getPosition().x,
                           bmfLabel->getPosition().y - conLabel->getContentSize().height);

// add the label as a child to this layer
this->addChild(ttfLabel, 1);
this->addChild(sysLabel, 1);
this->addChild(bmfLabel, 1);
this->addChild(conLabel, 1);
```


Demo
Hello World
你好 世界
Hello World
Hello World


中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX

BMFont



创建bmfont需要2个文件，1个.fnt文件和一张拥有每个.png格式字符的图片。我们可以看到创建bmfont的时候并不需要指名字体大小，这是因为BMFont是一种使用bitmap的标签类型。bitmap字体的特点是点矩阵形成的。非常快和方便去使用，但不可扩展的，因为它需要对每个大小字需要单独的字体。每个字在标签里都是一个独立的精灵。

```
auto bmfLabel = Label::createWithBMFont("fonts/futura-48.fnt", "Hello World");
```

TTF

Ttf，true type fonts和bmfont不同，ttf渲染的是字体的轮廓，因此我们不需要像bmfont那样，为各个大小或者颜色准备一个独立的字体。Ttf可以渲染不同大小的字体而不需要独立的字体文件。而且，我们还可以使用ttfConfig来预先设置好我们所预想的字体格式，然后使用ttfConfig来创建ttflabel

```
TTFConfig ttfConfig;
ttfConfig.fontFilePath = "fonts/Marker Felt.ttf";
ttfConfig.fontSize = 24;
ttfConfig.outlineSize = 2;
auto conLabel = Label::createWithTTF(ttfConfig, "Hello World");
```

LabelEffect

不是所有的标签对象都支持应用效果。效果有：shadow（阴影），outline（轮廓），glow（发光）。你可以应用一个或者多个效果到你的标签上。

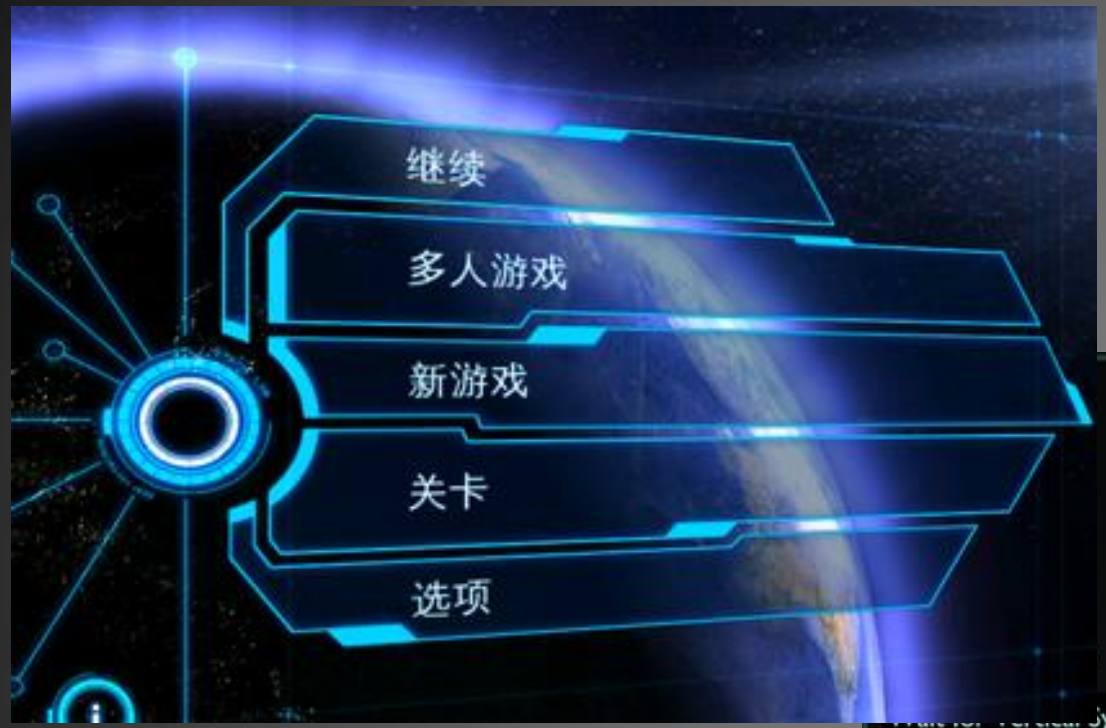
```
ttfLabel->enableGlow(Color4B::RED);
```


```
ttfLabel->enableShadow(Color4B::RED);
```


```
ttfLabel->enableOutline(Color4B::GREEN, 2);
```

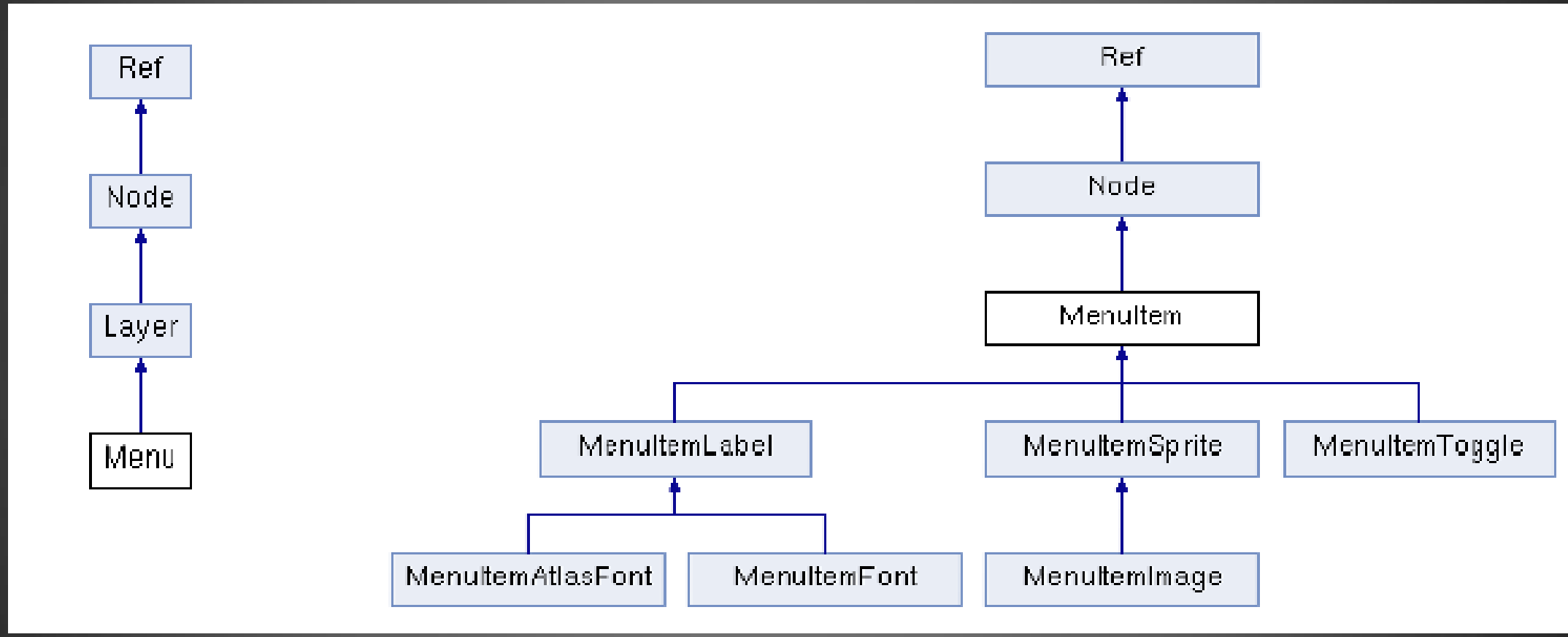
菜单Menu，MenuItem

菜单是每个游戏都不可或缺的元素。在游戏里，通过使用菜单对象在各个游戏选项中跳转。菜单经常包含按钮，例如：PLAY,QUIT，SETTING,ABOUT。菜单也能包含菜单对象来组成嵌套的菜单系统。一个菜单对象是一种特殊的节点对象。



Auto
1920x1080
Borderless
High
Effects Quality
High
Environment Quality
High
Shadows
Low
Frame Rate Cap
Stable
Advanced Options
Colorblind Mode
Enable HUD Animations
OK
Cancel

菜单和菜单项的继承图


Ref
Node
Layer
Menu
Ref
Node
MenuItem
MenuItemLabel
MenuItemSprite
MenuItemToggle
MenuItemAtlasFont
MenuItemFont
MenuItemImage

菜单Menu，MenuItem使用方法及例子

可以通过图片，label，精灵来创建菜单项目。除了使用CC_CALLBACK_X来调用回调函数，还可以使用lambda表达式。

```
auto menuLabel = Label::createWithSystemFont("菜单项1", "Microsoft Yahei", 24);
auto item1 = MenuItemLabel::create(menuLabel, CC_CALLBACK_0(HelloWorld::menuEvent,this));
auto closeItem = MenuItemImage::create(
    "CloseNormal.png",
    "CloseSelected.png",
    CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));
auto sprite = Sprite::create("item3.png");
auto item3 = MenuItemSprite::create(sprite,
                                    sprite,
                                    CC_CALLBACK_0(HelloWorld::menuEvent, this));
item1->setPosition(Vec2(origin.x + visibleSize.width / 2,
                        origin.y + visibleSize.height - item1->getContentSize().height));
closeItem->setPosition(Vec2(origin.x + visibleSize.width / 2,
                            item1->getPosition().y - closeItem->getContentSize().height));
item3->setPosition(Vec2(origin.x + visibleSize.width / 2,
    closeItem->getPosition().y - item3->getContentSize().height));
// create menu, it's an autorelease object
auto menu = Menu::create(item1,closeItem,item3, NULL);
menu->setPosition(Vec2::ZERO);
this->addChild(menu, 1);
```

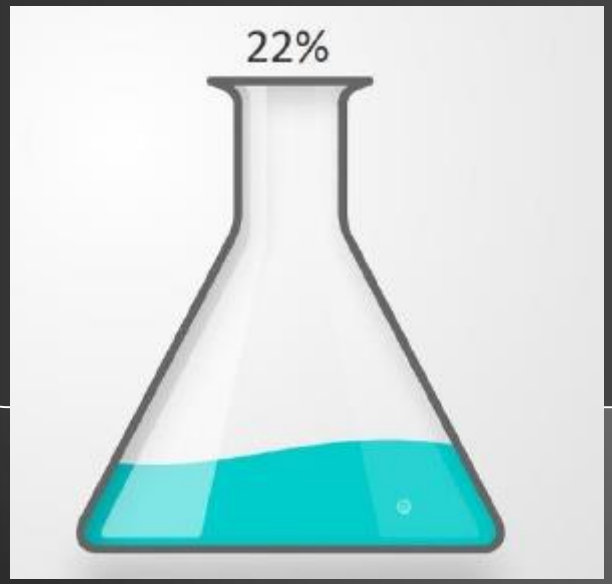
MenuItemSprite类

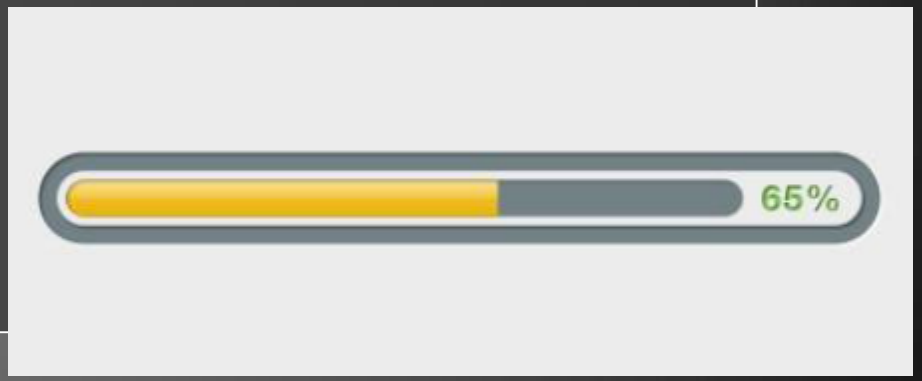
是一个由精灵对象组成的菜单按钮。此类的内部属性，提供了三个精灵对方，分别代表了按钮的三个状态。这就是正常、选中和无效。

```
//根据按钮状态不同的需求来创建MenuItem
static MenuItemSprite * create(Node* normalSprite, Node* selectedSprite, Node* disabledSprite = nullptr);
static MenuItemSprite * create(Node* normalSprite, Node* selectedSprite, const ccMenuCallback& callback);
static MenuItemSprite * create(Node* normalSprite, Node* selectedSprite, Node* disabledSprite, const
ccMenuCallback& callback);
```

进度条ProgressBar

在游戏中，我们经常能见到加载的时候提示我们加载进度的进度条。





进度条ProgressBar使用方法

主要接口

```
static ProgressTimer* create(Sprite* sp); // 从sprite创建
void setType(Type type); // 类型设置, RADIAL/BAR
void setPercentage(float percentage); // 设置进度
void setMidpoint(constPoint& point); // 设置旋转点
```

条形进度条

setBarChangeRate(); //进度条变化样式，参数为Point(x,y)

Point(1,0); //x轴变化

Point(0,1); //y轴变化

Point(1,1); //x,y轴都变化

当计时条样式为X轴变化时设置setMidpoint()：

Point(1,0); //从右到左

Point(0.5,0); //中间到两边

Point(0,0); //从左到右

当计时条样式为y轴变化时：

Point(0,1); //从上到下

Point(0,0.5); //中间到两边

Point(0,0); //从下到上

section2

动作Action

动作Action

让一个游戏更生动你必须让事物都动起来。动作对象是每个游戏中不可分割的一部分。让节点改变自身的属性达到动态的效果。动作对象允许节点在一段时间里改变自身的属性。任何对象只要是继承Node对象都可以使用Action来执行动作。

动作Aciton使用方法和例子

MoveTo,MoveBy

By是和节点现在的状态有关的。而To的动作是绝对的，这意味着To不需要获取节点现在的状态。

```
//参数: 动作时间长度，移动到的坐标
auto actionTo = MoveTo::create(2, Point(80,80));
//参数: 动作时间长度，移动的位移
auto actionBy = MoveBy::create(2, Point(80,80));
```

RotateTo,RotateBy

旋转动作和Move一样，By是和节点现在的状态有关的。而To的动作是绝对的，这意味着To不需要获取节点现在的状态。

```
//参数:动作时间长度，x和y轴上旋转到的角度
auto actionTo = RotateTo::create( 2, 45);
//参数:动作时间长度，x和y轴上旋转的角度
auto actionBy = RotateBy::create(2 , 360);
//创建一个由actionBy特效逆转的动作
auto actionByBack = actionBy->reverse();
```

COCOS2DX

ScaleTo,ScaleBy

缩放动作。

```
//参数:动作时间长度，缩放到的倍数
auto actionTo = ScaleTo::create(2.0f, 0.5f);
//参数:动作时间长度，缩放的倍数
auto actionBy= ScaleBy::create(2.0f, 1.0f, 10.0f);
```

JumpTo,JumpBy

跳跃动作。

```
//参数: 动作时间长度，跳跃的目的坐标，跳跃的高度，跳跃的次数
auto actionTo = JumpTo::create(2, Point(300,300), 50, 4);
//参数: 动作时间长度，跳跃的位移，跳跃的高度，跳跃的次数
auto actionBy = JumpBy::create(2, Point(300,0), 50, 4);
```

FadeIn,FadeOut

渐入淡出动作。

```
//在1秒内渐入
auto fadeIn = FadeIn::create(1.0f);//参数: 动作时间长度，跳跃的位移，跳跃的高度，跳跃的次数
//在2秒内淡出
auto fadeOut = FadeOut::create(2.0f);
```

COCOS2DX

TintTo,TintBy

渐入淡出动作。

```
//参数: 着色时间，红色、绿色、蓝色的着色目的值
auto action1 = TintTo::create(2, 255, 0, 255);
//参数: 着色时间，红色、绿色、蓝色的着色改变值
auto action2 = TintBy::create(2, -127, -255, -127);
```

Blink

闪烁动作。

```
//参数: 持续时间，闪烁次数
auto action1 = Blink::create(2, 10);
```

COCOS2DX

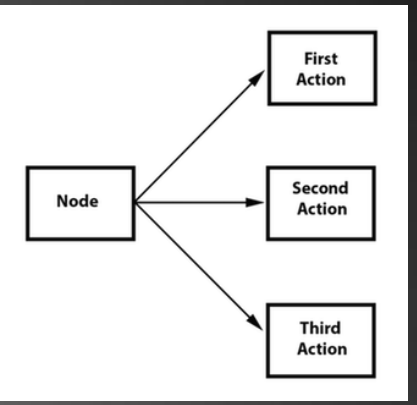
BezierTo,BezierBy

贝塞尔曲线动作动作。

```
//参数: 获得当前屏幕大小
auto s = Director::getInstance()->getWinSize();
//新建一个贝塞尔曲线，定义它的控制点和结束点
ccBezierConfig bezier;
bezier.controlPoint_1 = Point(0, s.height/2);
bezier.controlPoint_2 = Point(300, -s.height/2);
bezier.endPosition = Point(300,100);
//创建一个贝塞尔曲线特效，参数：持续的时间，贝塞尔曲线
auto bezierForward = BezierBy::create(3, bezier);
```

常用动作组合使用方法和例子

Spawn



同时运行所有的动作。你可以包含任意的动作对象，甚至是Spawn对象。

```
//创建一个组合特效，使这些特效同时进行，Spawn执行时间以最长的特效为准，但是里面的每个的动作的执行时间可以不同
//参数: FiniteTimeAction类的动作1， FiniteTimeAction类的动作2…… FiniteTimeAction类的动作N， NULL
auto action = Spawn::create(JumpBy::create(2, Point(300,0), 50, 4), RotateBy::create( 2, 720), NULL);
```

COCOS2DX

Sequence

顺序运行所有的动作。你可以包含任意的动作对象，甚至是Sequence对象。

```
//创建一个组合特效，使这些特效顺序进行
//参数: 动作特效1，动作特效2.....动作特效N，NULL
auto action = Sequence::create(MoveBy::create( 2,
Point(240,0)),RotateBy::create( 2, 540), NULL);
```

Repeat

重复和永久重复特效。

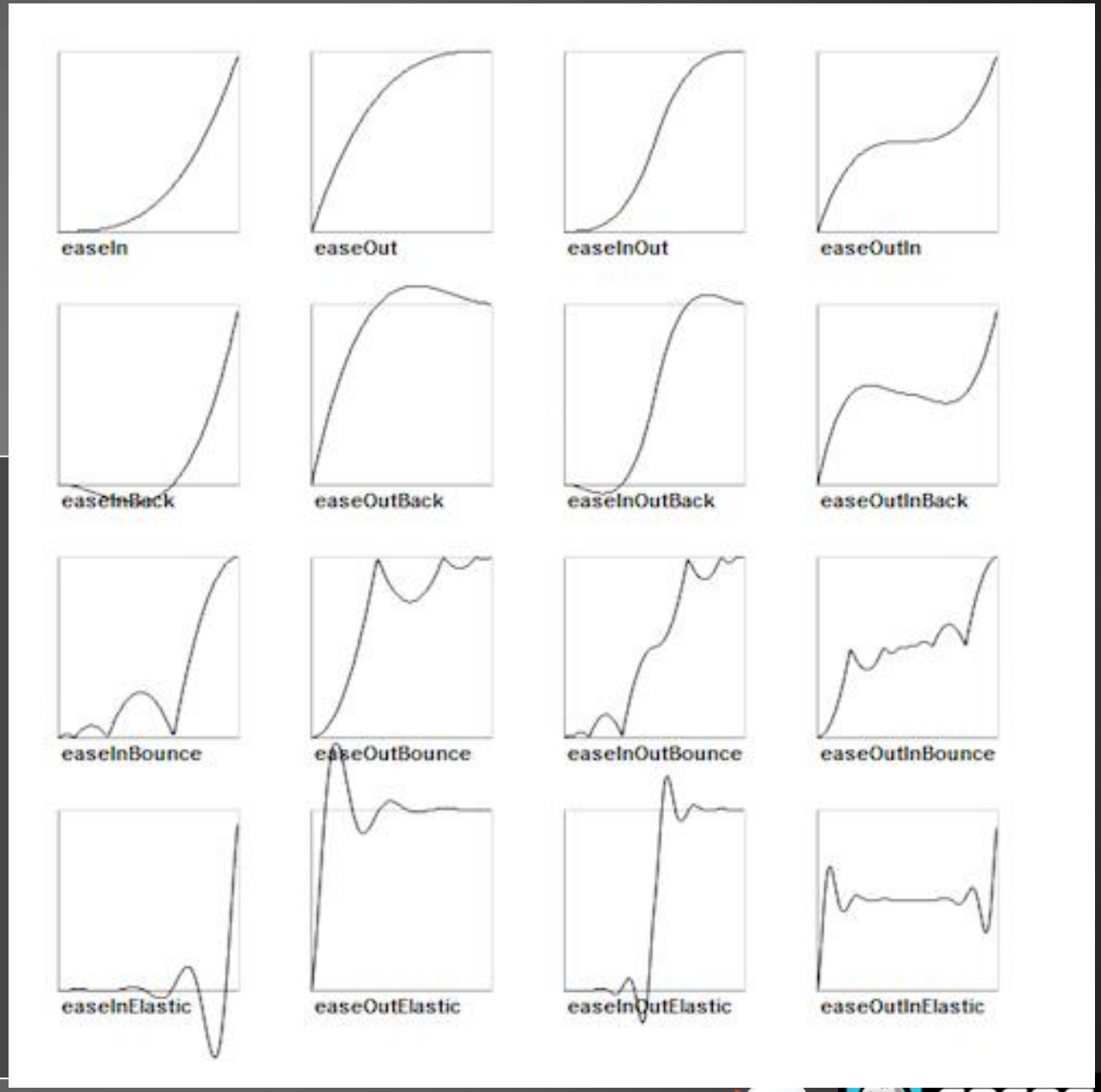
```
auto act1 = RotateTo::create(1, 90);
auto act2 = RotateTo::create(1, 0);
auto seq= Sequence::create(act1, act2, NULL);
//参数: 需要永久重复执行的ActionInterval动作特效
auto rep1 = RepeatForever::create(seq);
//参数: 需要重复执行的ActionInterval动作特效，重复执行的次数
auto rep2 = Repeat::create( seq->clone(), 10);
```

Speed

线性变速动作。继承自Action。用于线性的改变某个动作的速度。

```
//参数: 要改变速度的ActionInterval类的目标动作， 改变的速度倍率
    auto action = Speed::create(RepeatForever::create(spawn), 1.0f);
```

Easing


easeIn
easeOut
easeInOut
easeOutIn
easeInBack
easeOutBack
easeInOutBack
easeOutInBack
easeInBounce
easeOutBounce
easeInOutBounce
easeOutInBounce
easeInElastic
easeOutElastic
easeInOutElastic
easeOutInElastic

section3

帧动画SpriteFrame

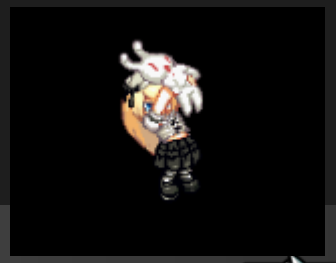
帧动画SpriteFrame

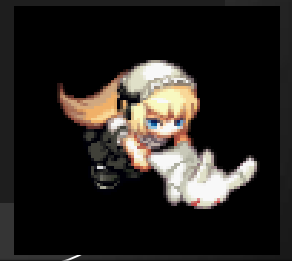
帧动画是一种常见的动画形式（Frame By Frame），其原理是在“连续的关键帧”中分解动画动作，也就是在时间轴的每帧上逐帧绘制不同的内容，使其连续播放而成动画。因为逐帧动画的帧序列内容不一样，不但给制作增加了负担而且最终输出的文件量也很大，但它的优势也很明显：逐帧动画具有非常大的灵活性，几乎可以表现任何想表现的内容，而它类似与电影的播放模式，很适合于表演细腻的动画。

COCOS2DX

帧动画Animate使用方法和例子

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

//创建一张贴图
auto texture = Director::getInstance()->getTextureCache()->addImage("$lucia_2.png");
//从贴图中以像素单位切割，创建关键帧
auto frame0 = SpriteFrame::createWithTexture(texture, CC_RECT_PIXELS_TO_POINTS(Rect(0, 0, 113, 113)));
//使用第一帧创建精灵
Sprite* sp = Sprite::createWithSpriteFrame(frame0);
sp->setPosition(Vec2(origin.x + visibleSize.width / 2,
                     origin.y + visibleSize.height - sp->getContentSize().height));
addChild(sp, 1);
//将所有关键帧放入VECTOR容器中
Vector<SpriteFrame*> sf;
sf.reserve(17);
for (int i = 0; i < 17; i++) {
    auto frame = SpriteFrame::createWithTexture(texture, CC_RECT_PIXELS_TO_POINTS(Rect(113*i,0,113,113)));
    sf.pushBack(frame);
}
//创建一个Animation,参数：SpriteFrame*的Vector容器，每一帧之间的间隔。
auto animation = Animation::createWithSpriteFrames(sf, 0.1f);
//使用animation创建一个animate,animate继承了ActionInterval,可以当做动作来使用
auto animate = Animate::create(animation);

sp->runAction(RepeatForever::create(animate));
```

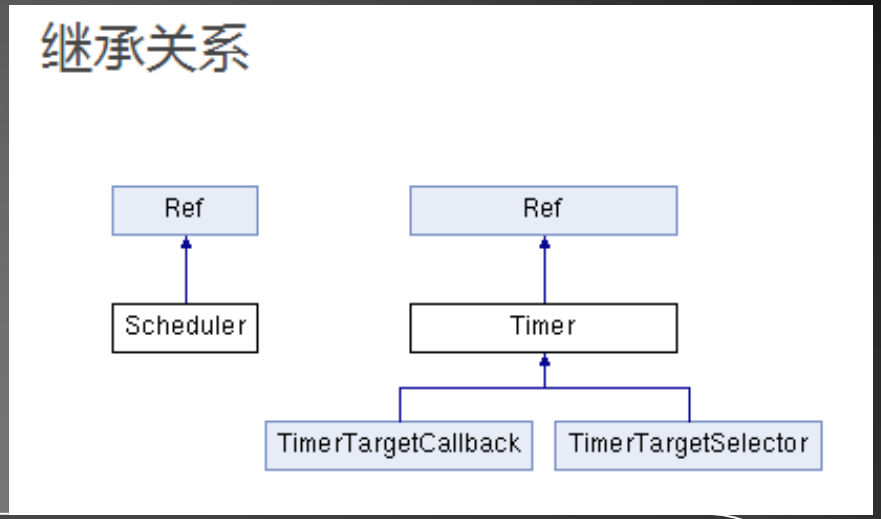




section4

调度器Scheduler,Update

调度器Schduler



游戏中我们经常会随时间的变化而做一些逻辑判断，如碰撞检测。为了解决以上问题，我们引入了调度器，这使得游戏能够更好的处理动态事件。Cocos2d-x提供了多种调度机制，在开发中我们通常会用到3种调度器：

默认调度器:schedulerUpdate()

自定义调度器:schedule(SEL_SCHEDULE selector, float interval, unsigned int repeat, float delay)

单次调度器:scheduleOnce(SEL_SCHEDULE selector, float delay)

默认调度器(schedulerUpdate)

该调度器是使用Node的刷新事件update方法，该方法在每帧绘制之前都会被调用一次。由于每帧之间时间间隔较短，所以每帧刷新一次已足够完成大部分游戏过程中需要的逻辑判断。

Cocos2d-x中Node默认是没有启用update事件的，因此你需要重载update方法来执行自己的逻辑代码。

通过执行schedulerUpdate()调度器每帧执行 update方法，如果需要停止这个调度器，可以使用unschedulerUpdate()方法。

COCOS2DX

HelloWorldScene.h

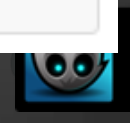
```
void update(float dt) override;
```

HelloWorldScene.cpp

```
bool HelloWorld::init()
{
    ...
    scheduleUpdate();
    return true;
}

void HelloWorld::update(float dt)
{
    log("update");
}
```

你会看到控制台不停输出如下信息

```
cocos2d: update
cocos2d: update
cocos2d: update
cocos2d: update
```



自定义调度器(scheduler)

游戏开发中，在某些情况下我们可能不需要频繁的进行逻辑检测，这样可以提高游戏性能。所以Cocos2d-x还提供了自定义调度器，可以实现以一定的时间间隔连续调用某个函数。

由于引擎的调度机制，自定义时间间隔必须大于两帧的间隔，否则两帧内的多次调用会被合并成一次调用。所以自定义时间间隔应在0.1秒以上。

同样，取消该调度器可以用unschedule(SEL_SCHEDULE selector, float delay)。

HelloWorldScene.h

```
void updateCustom(float dt);
```

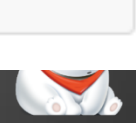
HelloWorldScene.cpp

```
bool HelloWorld::init()
{
    ...
    schedule(schedule_selector(HelloWorld::updateCustom), 1.0f, kRepeatForever, 0);
    return true;
}

void HelloWorld::updateCustom(float dt)
{
    log("Custom");
}
```

scheduler(SEL_SCHEDULE selector, float interval, unsigned int repeat, float delay)函数里面的参数：
第一个参数selector即为你要添加的事件函数
第二个参数interval为事件触发时间间隔
第三个参数repeat为触发一次事件后还会触发的次数，默认值为kRepeatForever，表示无限触发次数
第四个参数delay表示第一次触发之前的延时

在控制台你会看到每隔1秒输出以下信息

```
cocos2d: Custom
cocos2d: Custom
cocos2d: Custom
cocos2d: Custom
cocos2d: Custom
```



单次调度器(schedulerOnce)

游戏中某些场合，你只想进行一次逻辑检测，Cocos2d-x同样提供了单次调度器。

该调度器只会触发一次，用unschedule(SEL_SCHEDULE selector, float delay)来取消该触发器。

HelloWorldScene.h

```
void updateOnce(float dt);
```

HelloWorldScene.cpp

```
bool HelloWorld::init()
{
    ...
    scheduleOnce(schedule_selector(HelloWorld::updateOnce), 0.1f);
    return true;
}

void HelloWorld::updateOnce(float dt)
{
    log("Once");
}
```

这次在控制台你只会看到一次输出

```
cocos2d: Once
```

```
//取消所有的定时器，包括Update()定时器，不包括动作特效
void unscheduleAllSelectors(void);
```

案例分析：《崩坏学园2》

游戏模式是常见的横版打怪，打boss，横版动作类游戏，游戏的操作通过屏幕上的虚拟按键实现，左边是操纵杆，右边是武器，每点击一下武器，先判断是不是现在使用的武器，如果是，则开火，否则进行换武器操作。

游戏特色，角色和武器可以更换，boss和关卡有武器掉落，版本迭代更新不停有新内容可攻略。

作业：

实现一个横版游戏，具体要求：

1.左边wasd4个虚拟按键能控制角色移动

2.右边2个虚拟按键x，y能控制角色播放不同的帧动画

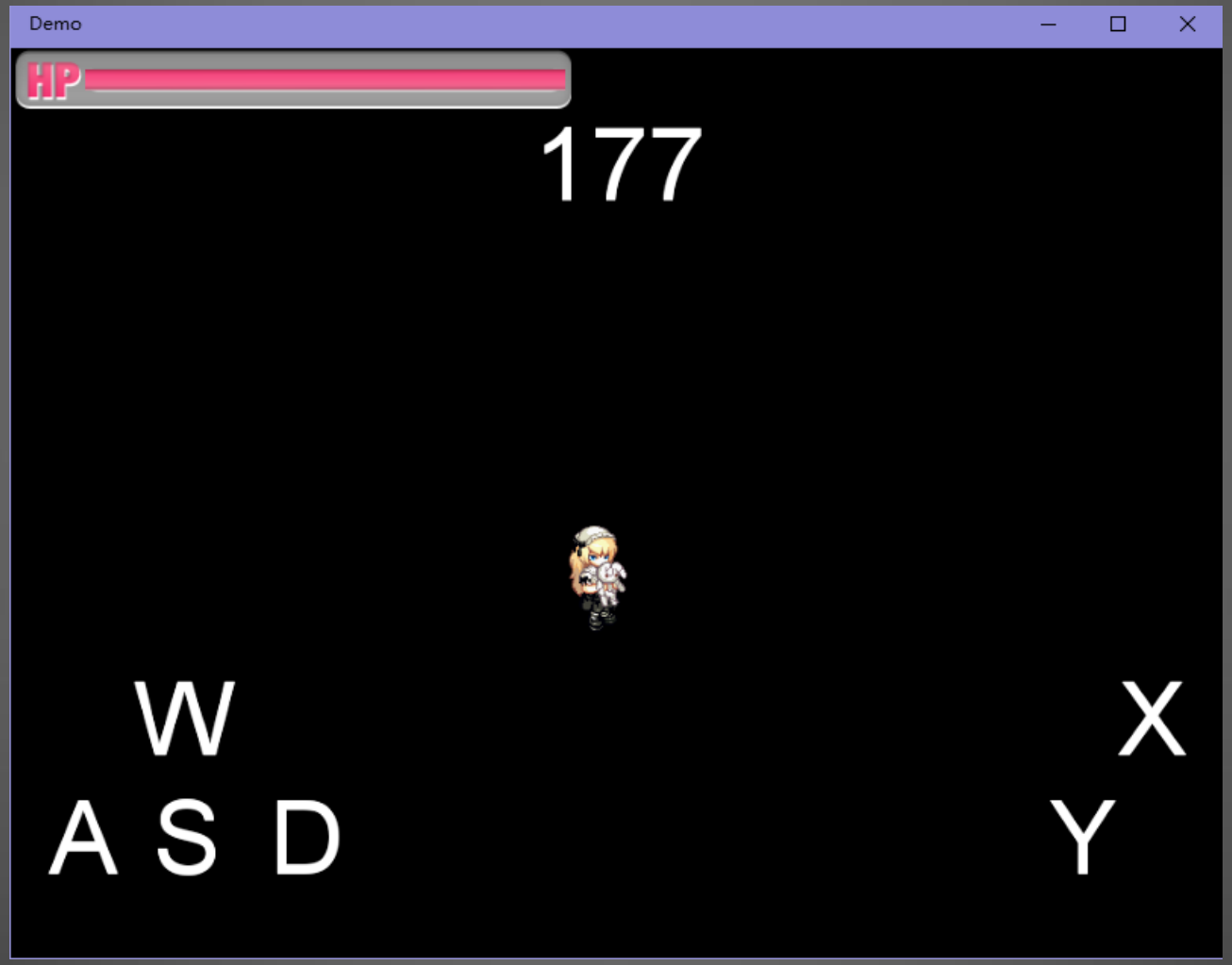
3.角色不会移动到可视窗口外（加分项）

4.添加倒计时

5.添加人物血条

6.点击虚拟按键x播放帧动画并让血条减少,点击y播放帧动画并让血条增加（加分项）

Demo演示：



THE END

THANKS FOR WATCHING