Cocos2dx 游戏开发教程

目录 / contents

Cocos2dx 游戏开发教程

CONTENTS

Label

你可以使用Label类创建LabelTTF和LabelBMFont中任意一种标签，标签类继承于SpriteBatchNode类，这样一来大大提高了渲染速度。

```
auto newLabel1 = Label::create("New Label", "Arial", 30);
auto newLabel2 = Label::createWithBMFont("bitmapFontTest.fnt", "New Label");
newLabel1->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
newLabel2->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y - 100));
addChild(newLabel1);
addChild(newLabel2);
TTFConfig ttfConfig; ttfConfig.fontSize = 30;
ttfConfig.fontFilePath = "Paint Boy.ttf";
auto label2 = Label::createWithTTF(ttfConfig, "New Label");
label2->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y + 100));
addChild(label2);
```

create方法默认创建一个LabelTTF标签，参数也和创建LabelTTF标签一样
createWithBMFont方法创建一个LabelBMFont标签，第一个参数为文件名，第二个参数为要显示的内容
createWithTTF方法使用.ttf文件来创建一个LabelTTF标签，需要注意的是要设置字体大小必须先配置好TTFConfig

LabelTTF

LabelTTF类使用系统中自带的字体，如果创建LabelTTF对象时未给出字体名字或者给出的名字系统中不存在，则使用引擎默认字体初始化对象。
引擎提供两种方式创建LableTTF:
用LabelTTF类的create方法创建
用Label类的createWithTTF方法创建，但是Label类是通过.ttf文件来创建的
以下代码分别使用LabelTTF和Label来创建Label:
现在推荐使用createWithSystemFont创建系统字体，方法与createWithTTF类似。

```
auto label1 = LabelTTF::create("Create with LabelTTF", "Arial", 30);
label1->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
 addChild(label1); TTFConfig ttfConfig; ttfConfig.fontSize = 30;
ttfConfig.fontFilePath = "Paint Boy.ttf"; auto label2 = Label::createWithTTF(ttfConfig, "Create with Label");
label2->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y - 100));
addChild(label2);
```

BMFont

LabelBMFont类是一个基于位图的字体图集，是一个包含所有你需要于坐标数据一起显示在屏幕上的字符的图像，它允许字符从主图中剪切出来。
以下代码用来创建LabelBMFont对象：


```
auto label = Label::createWithBMFont ("BMFont Test", "bitmapFontTest.fnt");
label->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
addChild(label);
```

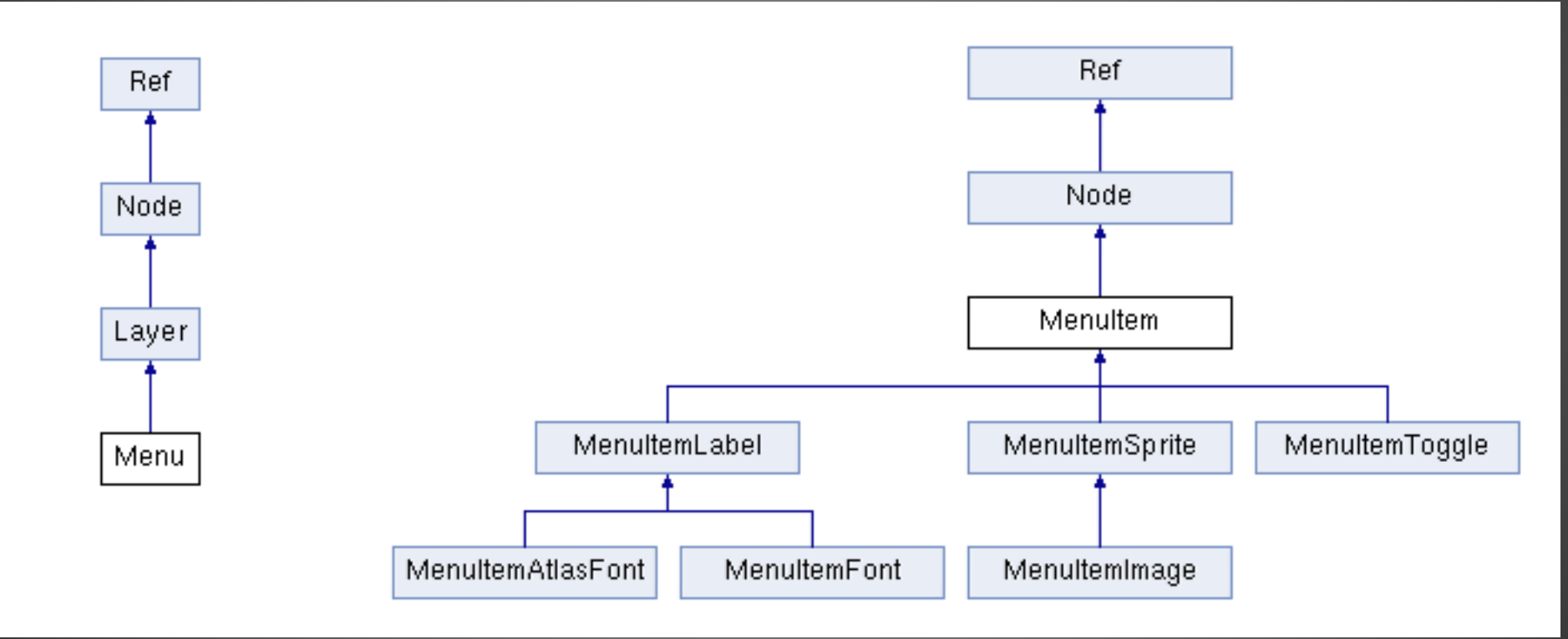
CONTENTS

section2

菜单: Menu, MenuItem


My Game
捕鱼达人
开始游戏
选择场景
GL verts: 726
GL calls: 7
58.2 / 0.00

菜单和菜单项继承图

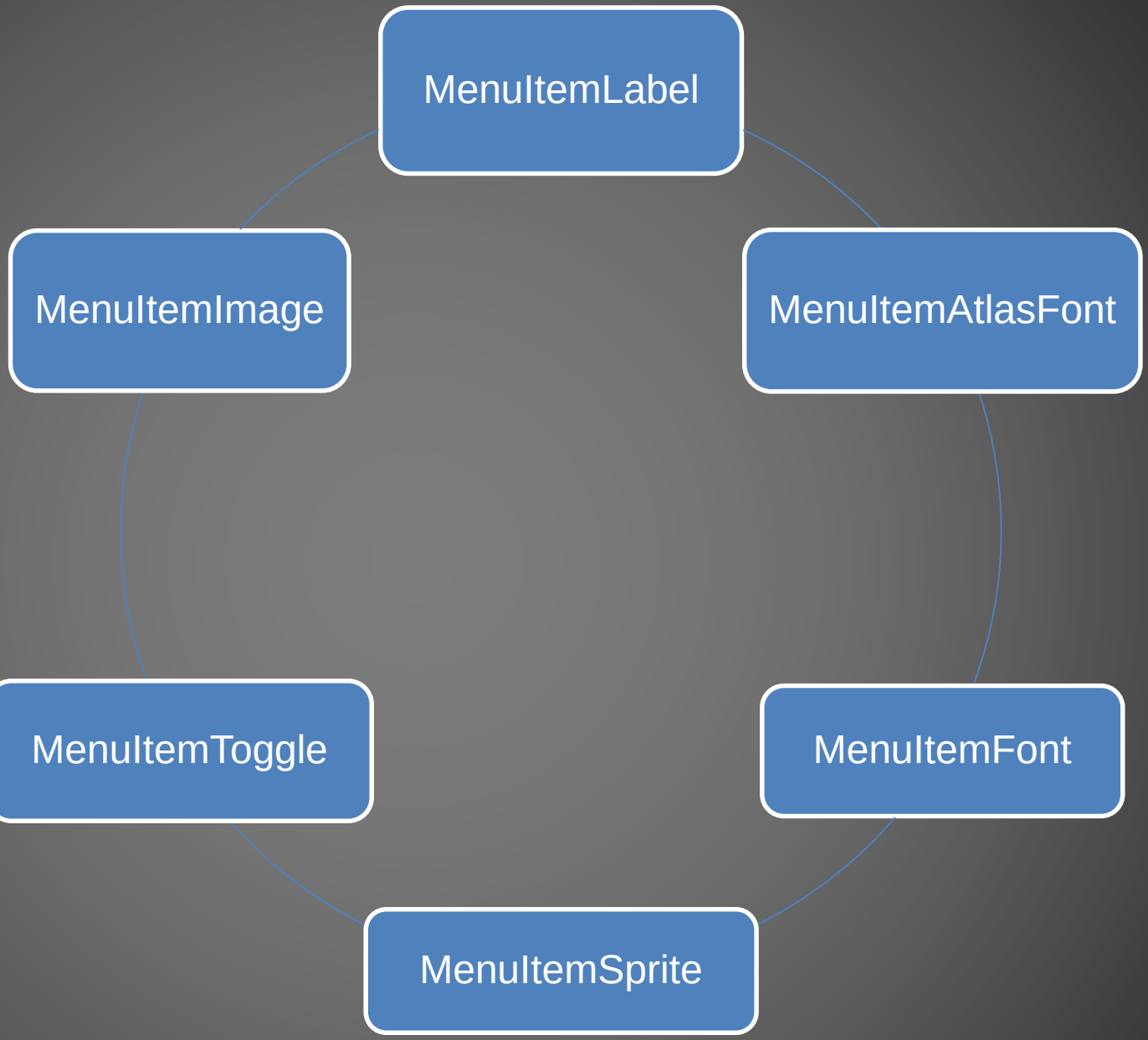

Ref
Node
Layer
Menu
Ref
Node
MenuItem
MenuItemLabel
MenuItemSprite
MenuItemToggle
MenuItemAtlasFont
MenuItemFont
MenuItemImage

Menu类

Menu：这是一个以按钮对象为集合的UI图层。它专门用于处理与玩家之间操作界面的交互。菜单图层中主要存放着MenuItem类以及他的子类。MenuItem是引擎中各种各样的按钮父类。大家可以将菜单图层类看作是按钮类组成的图层。他经常用在游戏中用户选择的界面，比如游戏开始时的主菜单或者关卡选择菜单。

```
//通过多个MenuItem来创建一个MenuItemToggle ,最后要以NULL结尾
 static Menu* create(MenuItem* item, ...) CC_REQUIRES_NULL_TERMINATION;

//通过MenuItem的容器创建一个MenuItemToggle，最后要以NULL结尾
 static Menu* createWithArray(const Vector<MenuItem*>& arrayOfItems);
```

MenuItem


MenuItemLabel
MenuItemAtlasFont
MenuItemFont
MenuItemSprite
MenuItemToggle
MenuItemImage


中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX

MenuItemLabel类

MenuItemLabel：包含了文字标签的按钮。类MenuItemLabel继承了LabelProtocol协议。这使得它可以持有一个用于显示文字的对象。同样具备了在玩家中选中按钮时的文字放大效果。此类的对象会将一个基本的文字标签转换为一个菜单按钮。

```
/** creates a MenuItemLabel with a Label and a callback */
static MenuItemLabel * create(Node*label, const ccMenuCallback& callback);

/** creates a MenuItemLabel with a Label. Target and selector will be nil */
static MenuItemLabel* create(Node *label);
```

MenuItemImage类

MenuItemImage：从类MenuItemSprite继承而来，没有太大的变化。只是提供了一个更为便捷的方式，将原本按钮中的精灵对象换位了三张纹理图片。开发者无须创建精灵对象，就可以直接创建一个按钮对象。

```
//根据按钮状态不同的需求来创建MenuItem
static MenuItemImage* create();
static MenuItemImage* create(const std::string& normalImage, const std::string& selectedImage);
static MenuItemImage* create(const std::string& normalImage, const std::string& selectedImage, const std::string&disabledImage);
static MenuItemImage* create(const std::string&normalImage, const std::string&selectedImage, const ccMenuCallback& callback);
static MenuItemImage* create(const std::string&normalImage, const std::string&selectedImage, const std::string&disabledImage, const ccMenuCallback& callback);
```

MenuItemSprite类

MenuItemSprite：顾名思义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。换句话来说，此按钮对象中三个按钮状态，将会以三个精灵对象的方式来表示。这就丰富了按钮的表现方式，给予开发者更大的自由。算得上是精灵与按钮功能的结合体。

```
//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);
```

section3

进度条:ProgressBar


65%
GBIN.COM

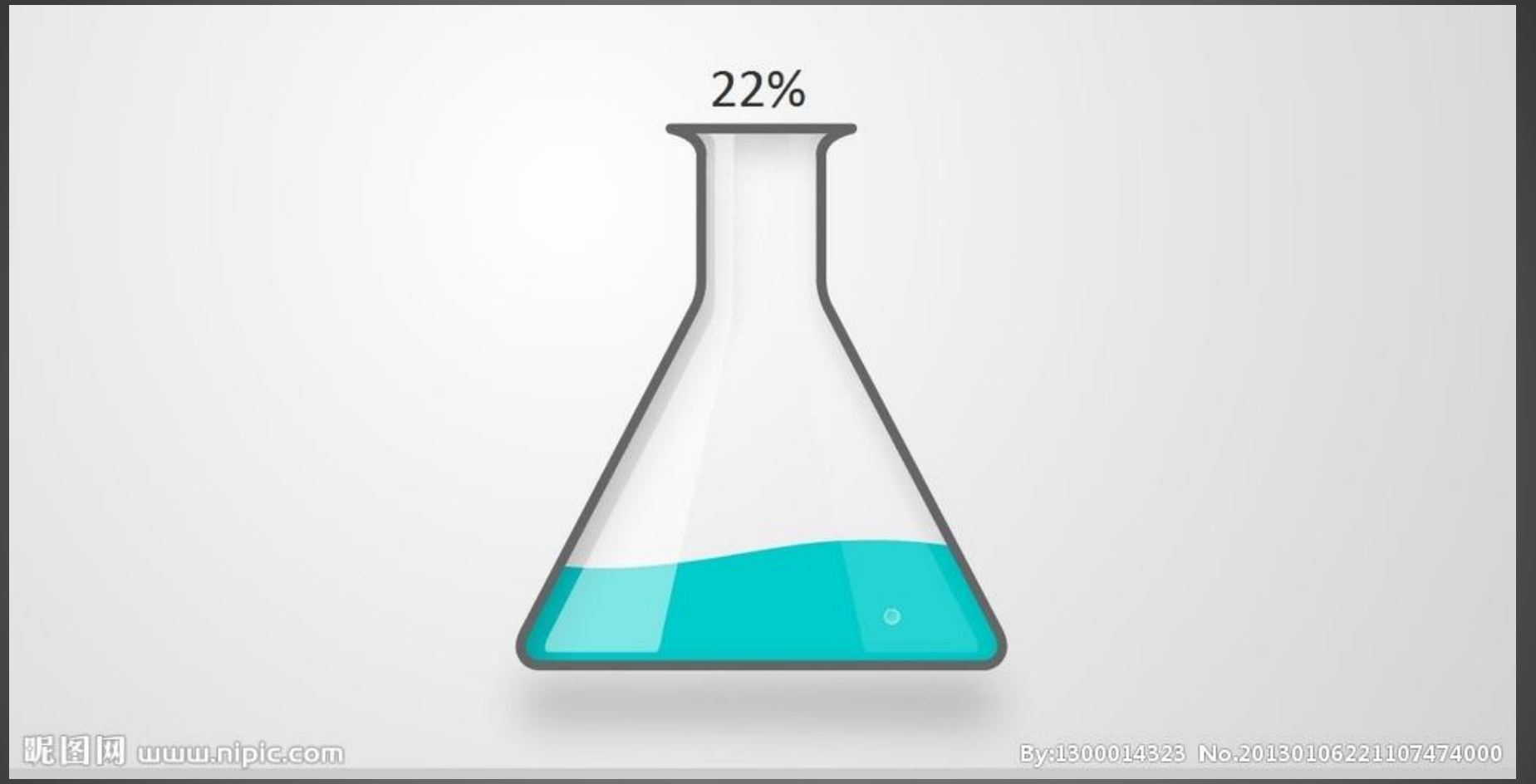

中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX


57%


中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX


22%
昵图网 www.nipic.com
By:1300014323 No.20130106221107474000


中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX

ProgressTimer

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(const Point& 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(1,1); //x,y轴都变化

Point(0,1); //y轴变化

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

COCOS2DX

CONTENTS

section4

Action，动作

Action：动作特效。引擎内部封装了很多Action的子类，它们分别实现了各种各样的特效，如移动、旋转、跳动、缩放、闪烁等等。每一个Node都可以通过runAction(Action* action)来播放一个特效。

动作特效代码示例：

```
//创建一个精灵
auto sprite = Sprite::create(“fish.png”);
 //创建一个移动的动作特效
auto moveAction = MoveTo::create(2, Point(0, 0));
 //让精灵执行这个动作
sprite->runAction(moveAction);
```

常用动作特效

MoveTo/MoveBy：移动动作。继承自ActionInterval

```
//参数: 持续时间，移动到的坐标
auto actionTo = MoveTo::create(2, Point(80,80));

//参数: 持续时间，移动的位移
auto actionBy = MoveBy::create(2, Point(80,80));
```

常用动作特效

RotateTo/RotateBy：旋转动作。继承自ActionInterval。

```
//参数: 持续时间，x和y轴上旋转到的角度
auto actionTo = RotateTo::create( 2, 45);

//参数: 持续时间，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();
```

常用动作特效

ScaleTo/ScaleBy：缩放特效。继承自ActionInterval。

```
//参数: 持续时间，缩放到的倍数
auto actionTo = ScaleTo::create(2.0f, 0.5f);

//参数: 持续时间，缩放的倍数
auto actionBy = ScaleBy::create(2.0f, 1.0f, 10.0f);
```

常用动作特效

SkewTo/SkewBy：倾斜动作。继承自ActionInterval。

```
//参数: 持续时间，x轴上倾斜到的角度，y轴上倾斜到的角度
auto actionTo = SkewTo::create(2, 37.2f, -37.2f);

//参数: 持续时间，x轴上倾斜的角度，y轴上倾斜的角度
auto actionBy = SkewBy::create(2, 0.0f, -90.0f);
```

常用动作特效

JumpTo/JumpBy：弹跳动作。继承自ActionInterval。

```
//参数: 持续时间，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: 持续时间，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);
```

BezierTo/BezierBy：贝塞尔曲线动作。继承自ActionInterval。

```
//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);
```

```
//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 bezier2;
bezier2.controlPoint_1 = Point(100, s.height/2);
bezier2.controlPoint_2 = Point(200, -s.height/2);
bezier2.endPosition = Point(240,160);

//创建一个贝塞尔曲线特效，参数：持续的时间，贝塞尔曲线
auto bezierTo1 = BezierTo::create(2, bezier2);
```

此时的控制点变成相对位移

常用动作特效

Blink：闪烁特效。继承自ActionInterval。

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

常用动作特效

FadeIn/FadeOut：淡入淡出特效。继承自ActionInterval。

```
//参数: 淡入淡出持续时间
auto action1 = FadeIn::create(1.0f);
auto action2 = FadeOut::create(1.0f);
```

常用动作特效

TintTo/TintBy：染色特效。继承自ActionInterval。

```
//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);
```

常用组合动作特效

Spawn：同时进行。继承自ActionInterval。

```
//创建一个组合特效，使这些特效同时进行，执行时间以最长的特效为准
//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);
```

常用组合动作特效

Sequence：顺序执行动作序列。继承自ActionInterval。

```
//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/RepeatForever：重复和永久重复特效。继承自ActionInterval。

```
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);
```

变速动作

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。参考ActionEaseTest。

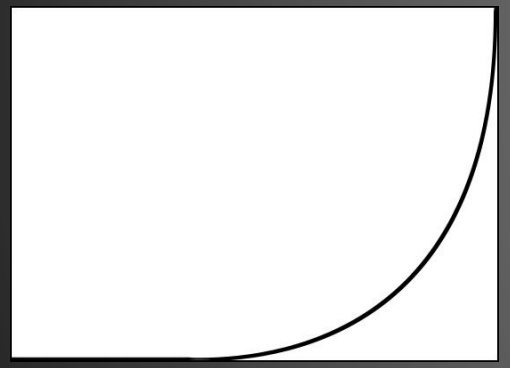
```
//参数: 要改变速度的ActionInterval类的目标动作，改变的速度倍率
 auto action = Speed::create(RepeatForever::create(spawn), 1.0f);
```

变速动作

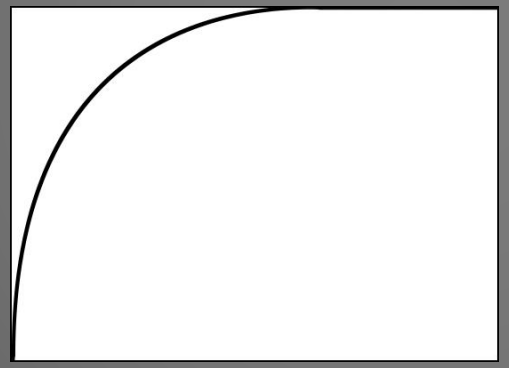
EaseExponentialIn：指数缓冲动作。继承自Action。用于指数曲线（如下图）的变换某个动作的速度。参考ActionEaseTest。

```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseExponentialIn::create(move->clone());
auto move_ease_out = EaseExponentialOut::create(move->clone());
auto move_ease = EaseExponentialInOut::create(move->clone() );
```

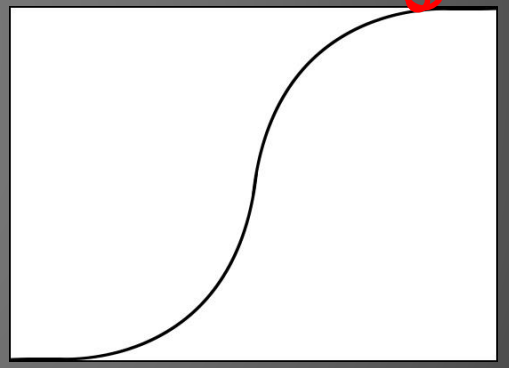
横轴代表实际动画时间，竖轴代表变换后的动画时间。



EaseExponentialIn



EaseExponentialOut

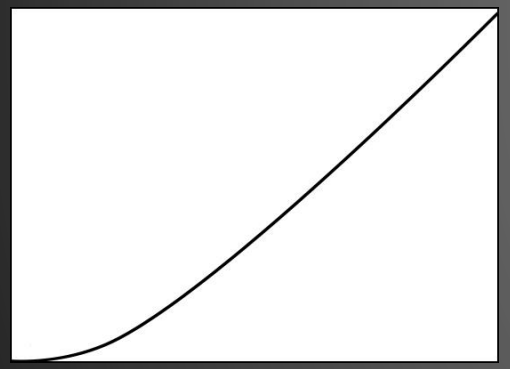


EaseExponentialInOut

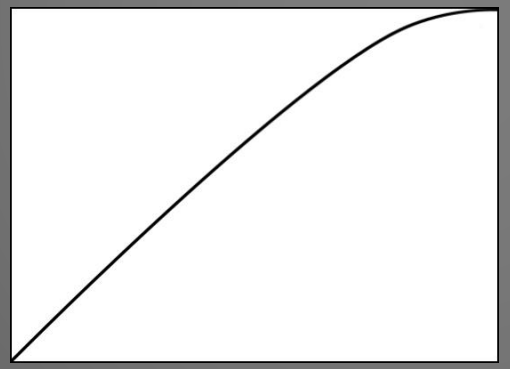
变速动作

EaseSineInOut：正弦缓冲。继承自Action。用于Sine曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

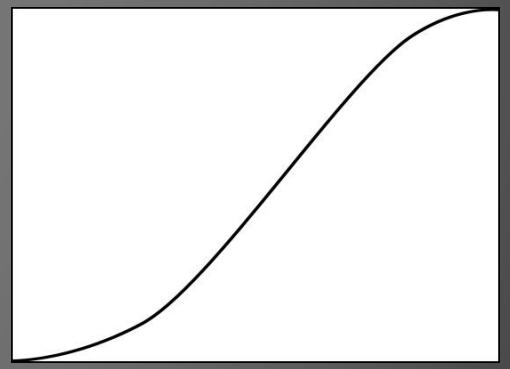
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseSineIn::create(move->clone() );
auto move_ease_out = EaseSineOut::create(move->clone() );
auto move_ease = EaseSineInOut::create(move->clone() );
```



EaseSineIn



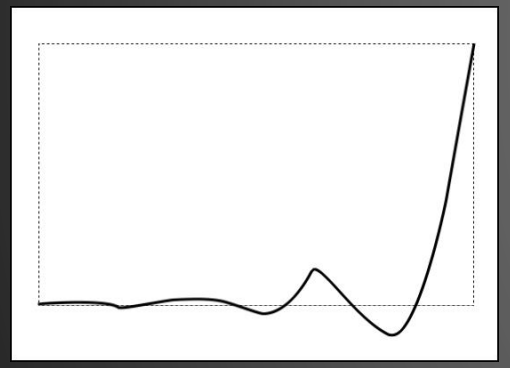
EaseSineOut



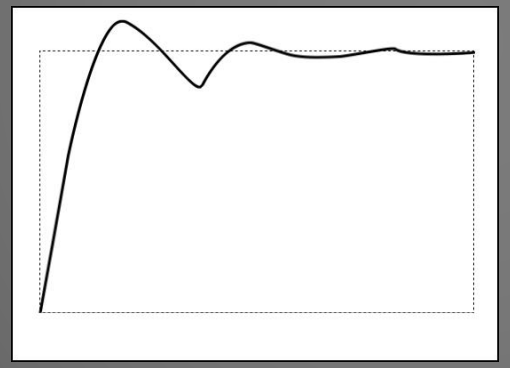
EaseSineInOut

EaseElasticInOut：弹性缓冲。继承自Action。用于弹性曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

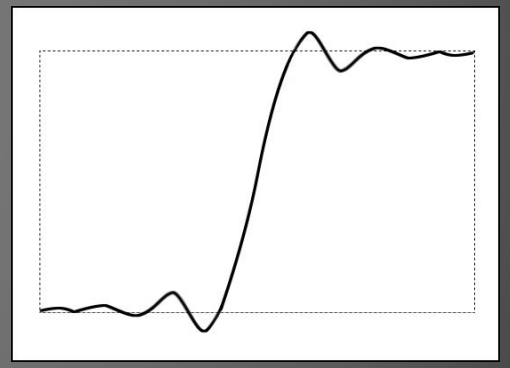
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseElasticIn::create(move->clone() );
auto move_ease_out = EaseElasticOut::create(move->clone() );
auto move_ease_inout1 = EaseElasticInOut::create(move->clone(), 0.3f);
```



EaseElasticIn



EaseElasticOut

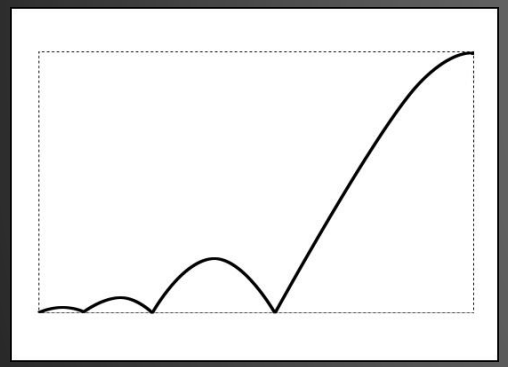


EaseElasticInOut

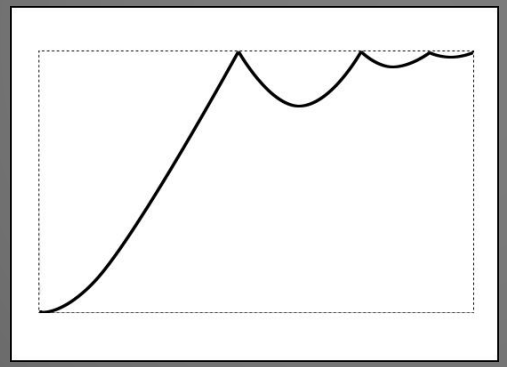
变速动作

EaseBounceInOut：跳跃缓冲。继承自Action。用于跳跃曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

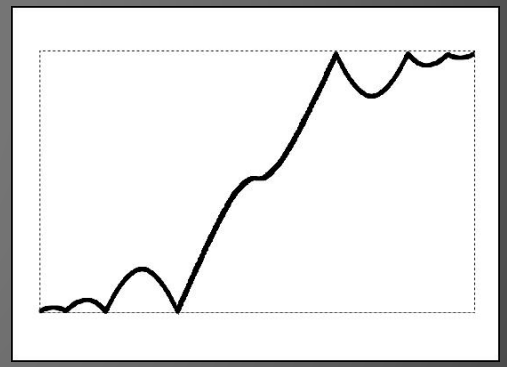
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseBounceIn::create(move->clone() );
auto move_ease_out = EaseBounceOut::create(move->clone() );
auto move_ease = EaseBounceInOut::create(move->clone() );
```



EaseBounceIn



EaseBounceOut

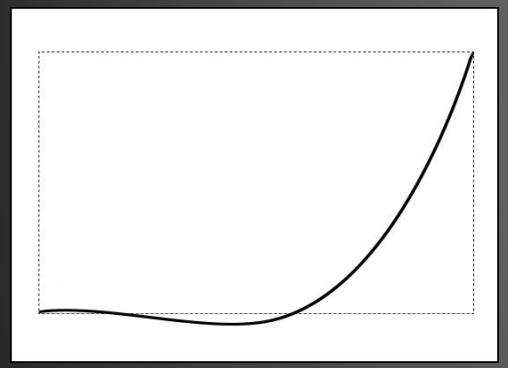


EaseBounceInOut

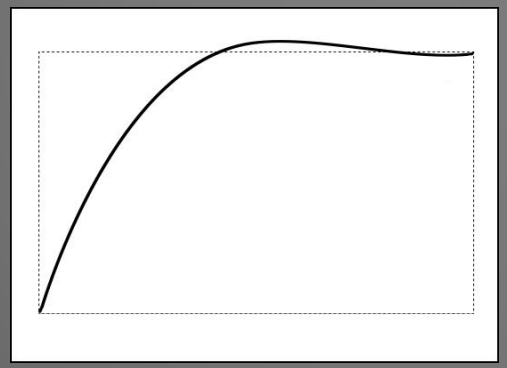
变速动作

EaseBackInOut：回震缓冲。继承自Action。用于回震曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

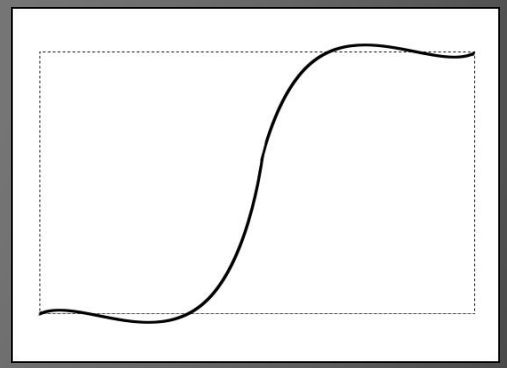
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseBackIn::create(move->clone());
auto move_ease_out = EaseBackOut::create( move->clone());
auto move_ease = EaseBackInOut::create(move->clone() );
```



EaseBackIn



EaseBackOut



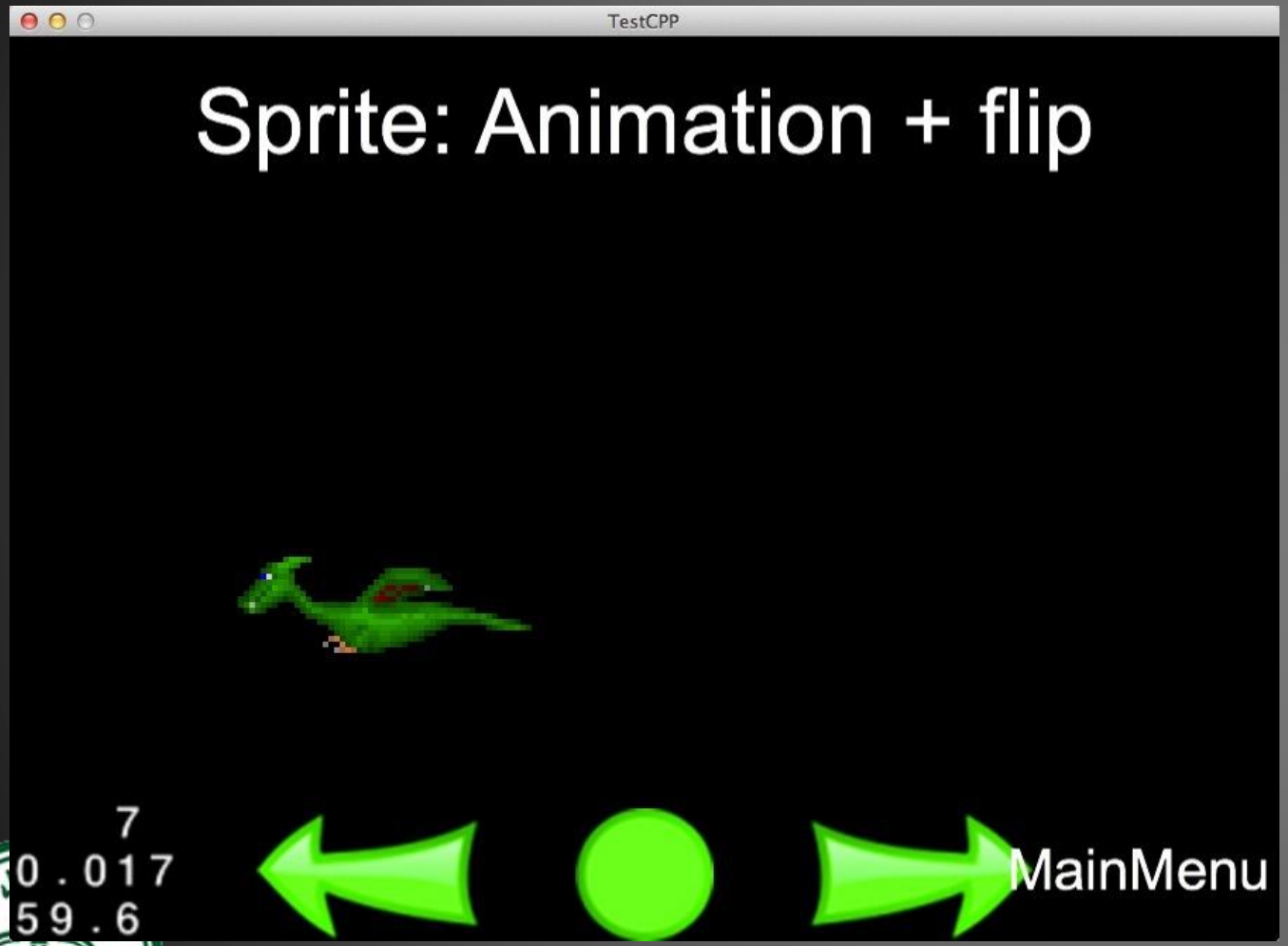
EaseBackInOut

CONTENTS

section5

SpriteFrame，帧动画

SpriteFrame Test

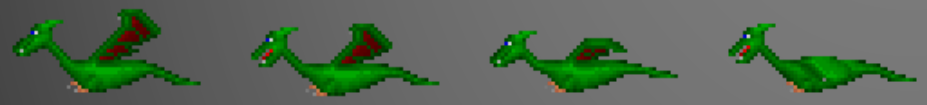


帧动画是一种常见的动画形式（Frame By Frame），其原理是在“连续的关键帧”中分解动画动作，也就是在时间轴的每帧上逐帧绘制不同的内容，使其连续播放而成动画。因为逐帧动画的帧序列内容不一样，不但给制作增加了负担而且最终输出的文件量也很大，但它的优势也很明显：逐帧动画具有非常大的灵活性，几乎可以表现任何想表现的内容，而它类似与电影的播放模式，很适合于表演细腻的动画。

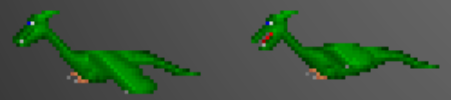
SpriteFrame Test

```
SpriteAnimationSplit::SpriteAnimationSplit()
{
    auto s = Director::getInstance()->getWinSize();

    // 创建一张贴图
    auto texture = Director::getInstance()->getTextureCache()-
>addImage("animations/dragon_animation.png");

    // 从贴图中切割范围，以此来创建动画的关键帧
    auto frame0 = SpriteFrame::createWithTexture(texture, Rect(132*0, 132*0, 132, 132));
    auto frame1 = SpriteFrame::createWithTexture(texture, Rect(132*1, 132*0, 132, 132));
    auto frame2 = SpriteFrame::createWithTexture(texture, Rect(132*2, 132*0, 132, 132));
    auto frame3 = SpriteFrame::createWithTexture(texture, Rect(132*3, 132*0, 132, 132));
    auto frame4 = SpriteFrame::createWithTexture(texture, Rect(132*0, 132*1, 132, 132));
    auto frame5 = SpriteFrame::createWithTexture(texture, Rect(132*1, 132*1, 132, 132));
```

参数：
切割的纹理的x轴坐标，
切割的纹理的y轴坐标，
切割的纹理的宽度，
切割的纹理的高度。



frame0 frame1 frame2 frame3



frame4 frame5

Sprite Test 续上一页代码

```
    // 从第一针开始创建一个精灵
    auto sprite = Sprite::createWithSpriteFrame(frame0);
    sprite->setPosition( Point( s.width/2-80, s.height/2) );
    addChild(sprite);

    // 创建一个容器，用来存储关键帧
    Vector<SpriteFrame*> animFrames(6);
    animFrames.pushBack(frame0);
    animFrames.pushBack(frame1);
    animFrames.pushBack(frame2);
    animFrames.pushBack(frame3);
    animFrames.pushBack(frame4);
    animFrames.pushBack(frame5);

    // 通过关键怎创建一个Animation，参数：SpriteFrame类的Vector容器，每一帧之间的间隔
    auto animation = Animation::createWithSpriteFrames(animFrames, 0.2f);

    // 通过animation创建好的动画来创建一个Animate，Animate继承自ActionInterval，它能被当做动作特效播放
    auto animate = Animate::create(animation);

    // 创建一个序列动作特效，播放刚才创建的动画，当播放完毕的时候翻转精灵，再次播放一次
    auto seq = Sequence::create(animate, FlipX::create(true), animate->clone(),  FlipX::create(false), NULL);

    // 播放序列动作特效
    sprite->runAction(RepeatForever::create( seq ) );
}
```

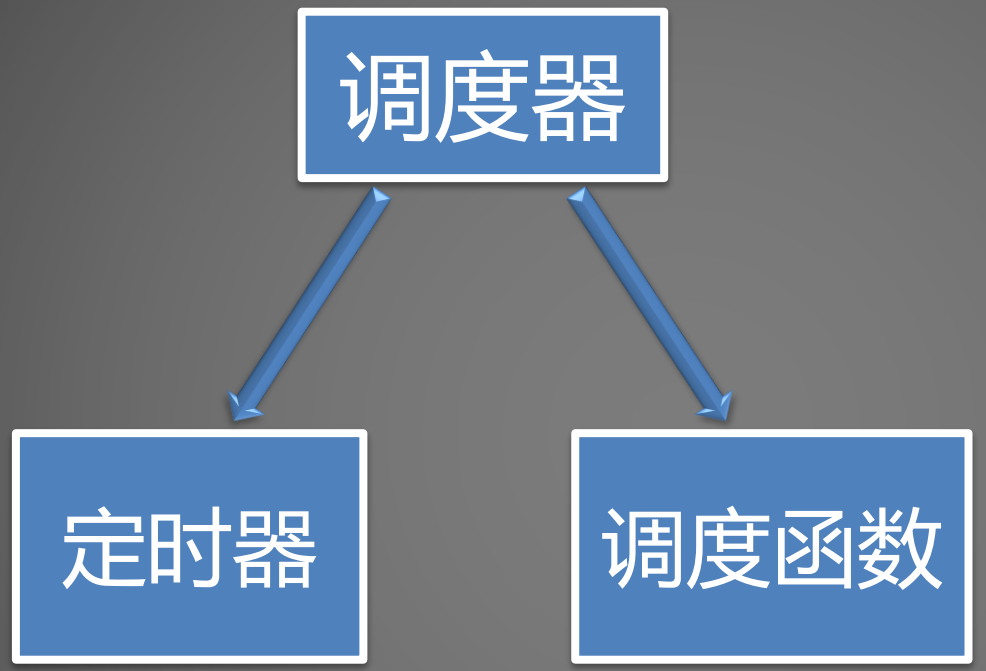
CONTENTS


section6
Scheduler,Update


中山大學
SUN YAT-SEN UNIVERSITY


COCOS2DX

Scheduler



Schedule 调度器注册

Schedule定时器可以实现一定时间的间隔连续调用某个函数。

```
//创建一个每一帧都调用的定时器，参数：需要调用的回调函数
void schedule(SEL_SCHEDULE selector);

//创建一个每单位时间内都调用的定时器，参数：需要调用的回调函数，调用的间隔（秒）
void schedule(SEL_SCHEDULE selector, float interval);

//创建一个每单位时间内都调用的定时器，参数：需要调用的回调函数，调用的间隔（秒），重复调用次数，延迟的时间
void schedule(SEL_SCHEDULE selector, float interval, unsigned int repeat, float delay);

//创建一个调用一次的定时器，参数：需要调用的回调函数，调用前的延迟时间
void scheduleOnce(SEL_SCHEDULE selector, float delay);

//取消调用一个定时器
void unschedule(SEL_SCHEDULE selector);

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

//恢复所有的定时器以及监听事件
void resume(void);

//暂停所有的定时器、动作特效以及监听事件
void pause(void);
```

Schedule定时器示例

```
void SchedulerAutoremove::onEnter()
{
    SchedulerTestLayer::onEnter();

    //创建一个2个定时器，分别调用2个方法，每0.5秒调用一次
    schedule(schedule_selector(SchedulerAutoremove::autoremove), 0.5f);
    schedule(schedule_selector(SchedulerAutoremove::tick), 0.5f);
    accum = 0;
}
void SchedulerAutoremove::autoremove(float dt)
{
    accum += dt;
    CCLOG("Time: %f", accum);

    //当计数器大于3的时候停止定时器调用
    if( accum > 3 )
    {

        unschedule(schedule_selector(SchedulerAutoremove::autoremove));
        CCLOG("scheduler removed");
    }
}
void SchedulerAutoremove::tick(float dt)
{
    CCLOG("This scheduler should not be removed");
}
```

由于定时器可以在间隔时间内调用的原理，利用Update()定时器和Schedule()定时器对节点各个属性进行不断的变化操作可以制作出动态控制的动画或者特效效果。参考ScheduleTest。

COCOS2DX

作业：

本次作业至少制作两个界面，界面中至少填充有两种Label，两种MenuItem，三种Action，进度条，帧动画，定时器各使用一次，素材自寻。

以界面的美观，控件之间的配合协作程度（如：进度条与计时器，Label相结合）加分。

请在实验报告中截图注明你所填充的内容，以免改作业时漏看。

THE END

THANKS FOR WATCHING