Cocos2d-x_UI_调度器_帧动画

骆铭涛

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。

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

```
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

菜单Menu，MenuItem

- 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);
```

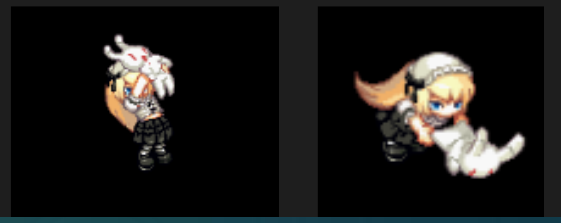
帧动画

- 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。

```
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));
```



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。

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);
```

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

HelloWorldScene.cpp

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

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

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
```

案例分析：崩坏学园2

作业

- 实现一个横版游戏，具体要求：
- 左边wasd4个虚拟按键能控制角色移动
- 右边2个虚拟按键x，y能控制角色播放不同的帧动画
- 界面所有字体要求：使用fonts目录下的arial.ttf，字体大小为36
- 角色不会移动到可视窗口外
- 添加倒计时
- 添加人物血条
- X、Y播放的动画不能同时播放
- 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）