Cocos2d-x 事件处理与音效

目 录 / contents

section1

事件分发与响应

现在大部分移动游戏应用中，都是通过玩家触摸屏幕的方法进行人机交互，桌面游戏中，则以鼠键输入为主。这一节我们会涉及到cocos2dx自带的各种事件的分发与响应的知识，以及如何注册、分发一个自定义事件。

游戏输入设备

传统的游戏主机以按钮输入为主

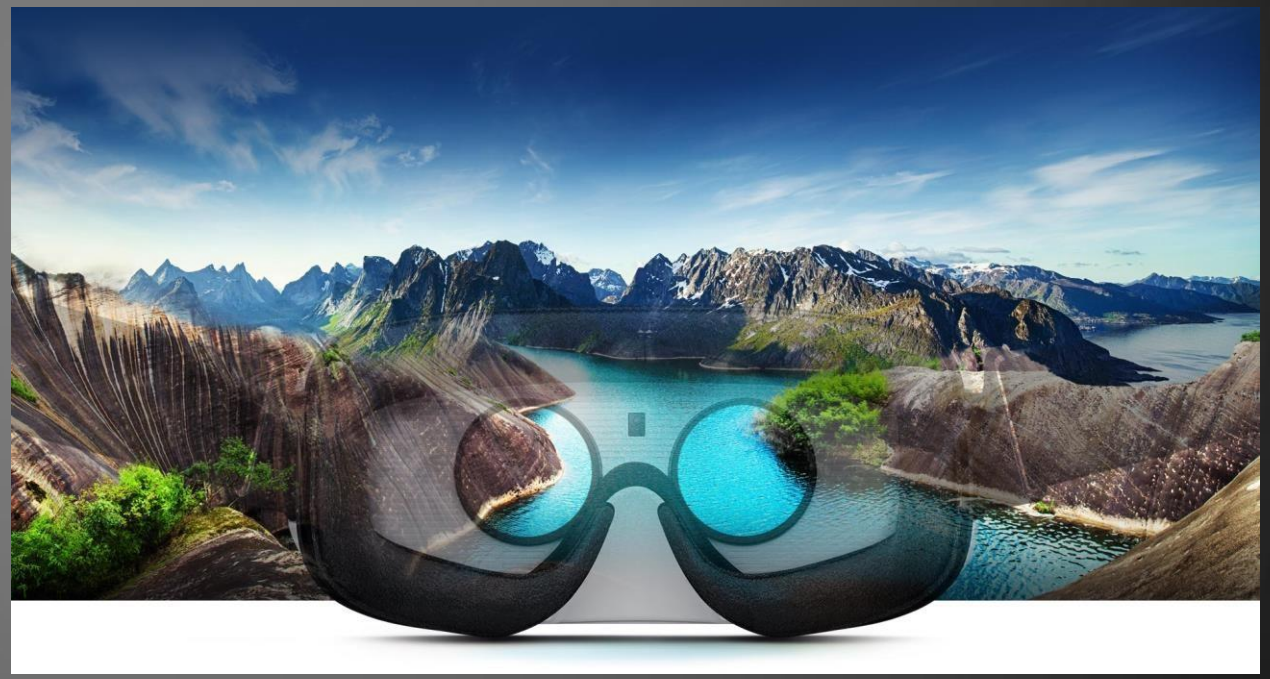
PC游戏则以键鼠为主要输入设备

游戏输入设备

随着移动智能设备的普及，移动端游戏也发展起来，主要以触摸屏交互和加速度感应器等为输入设备。

游戏输入设备

现在虚拟现实、运动捕捉正发展迅猛，在可见的将来，也许身临其境不再是夸张



索尼VR宣传视频

从输入到响应

有这么多种多样的输入，游戏程序是如何响应的？

人有五官，有视觉、嗅觉、触觉等，从周围世界接受信息。

一个人的行为、物的状态的改变或事的进展可以引发一个事件。
事件发生后，通过人的感知、大脑的反映，作出决策，付诸行动

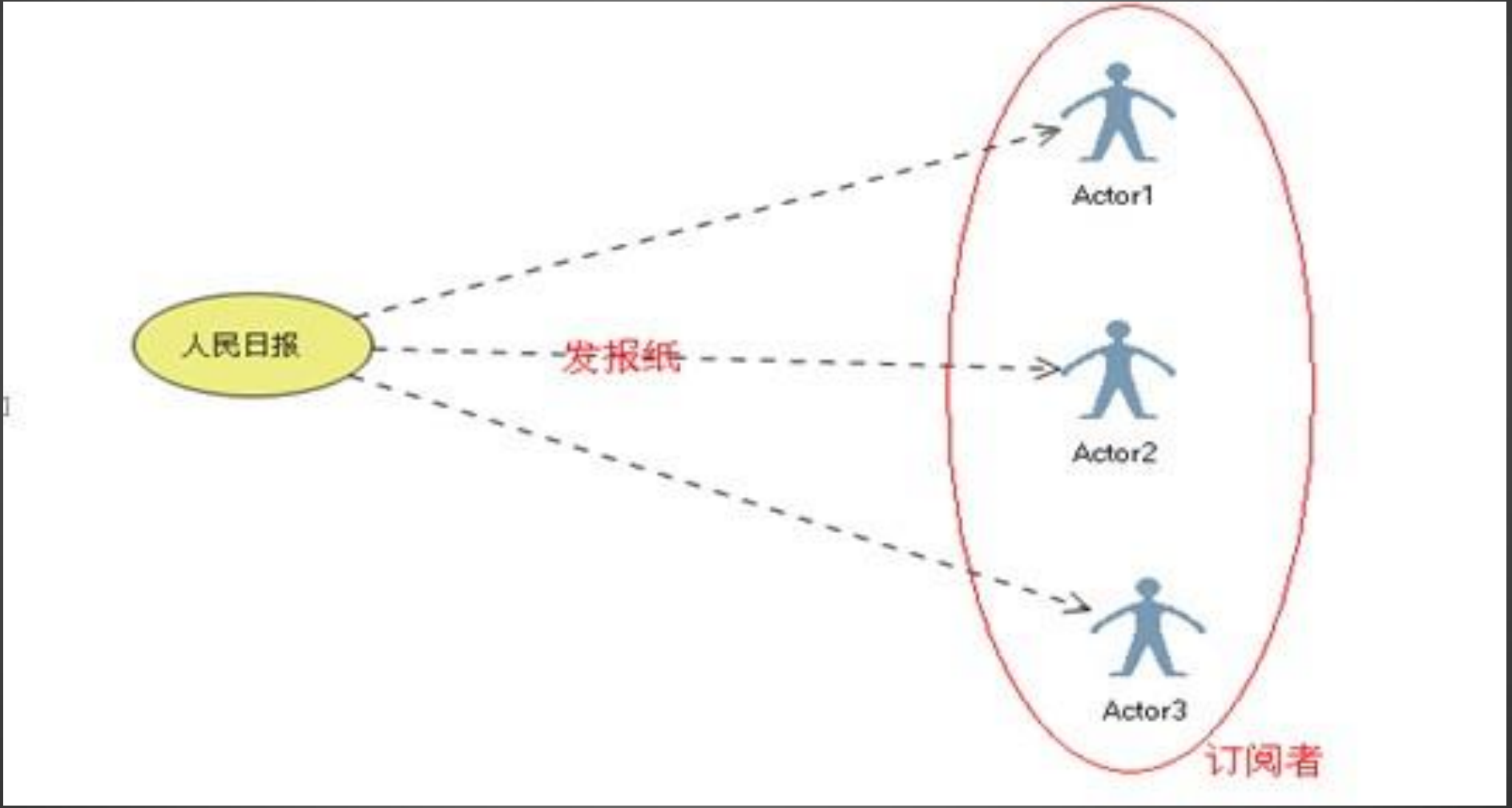
从输入到响应

如何用程序代码模拟实现这种模式？

订阅者模式

- 设计模式的一种，可用于对象间信息传递与事件发布，降低对象间的耦合性，也叫观察者模式
- 由订阅者与发布者组成，游戏开发中，订阅者向事件发布器（发布者）注册监听器以监听某个事件的发生，当此事件发生时，事件发布器根据事件ID向监听器分发事件，然后监听器进行事件响应

订阅者模式--示意图



Cocos2d-x的事件分发机制

Cocos2d-x对于事件的处理就使用了订阅者模式

- **事件监听器**：即订阅者，封装了事件处理的代码，负责响应事件的处理
- **事件分发器**：即发布者，当事件发生时通知对应事件的监听器
- **事件对象**：包含了关于事件的信息。

COCOS2D-X

Cocos2d-x的事件分发机制

在手游中，最重要的事件是触摸事件。它们很容易被创建来提供通用的功能。下面我们以触摸事件为例了解cocos2d-x的事件分发机制。

Cocos2d-x的事件分发机制

我们要明确什么是触摸事件。当你触摸移动设备的屏幕时，屏幕会接收到这个触摸行为，并检查你触摸了哪里以及决定你触摸到了什么，然后你的触摸行为就会被响应。

你所触摸的或许不是响应对象，但很有可能是它下面的东西。通常会给触摸事件分配优先级，优先级最高的就是被先响应的。

Cocos2d-x的事件分发机制

首先我们需要获取一个事件分发器，注意事件分发器是单例模式，只有一个实例。

可以使用getInstance()获取，或者直接使用_eventDispatcher

```
auto dispatcher = Director::getInstance()->getEventDispatcher();
// 或者
_eventDispatcher
```

COCOS2D-X

Cocos2d-x的事件分发机制

然后我们使用create()函数创建一个单点触摸的监听器EventListenerTouchOneByOne：

```
auto touchListener = EventListenerTouchOneByOne::create();
```

再将具体的事件和其响应函数绑定：

```
touchListener->onTouchBegan =
    CC_CALLBACK_2(HelloWorld::onTouchBegan, this);
```

// 响应函数的编写方式后面会提到

Cocos2d-x的事件分发机制

监听器配置好后，在事件分发器中，添加触摸监听器及其相应的绑定Sprite：

```
_eventDispatcher->addEventListenerWithSceneGraphPriority(listener1, sprite1);
```

需要重复使用某个listener时要调用clone函数，因为用addEventListenerWithSceneGraphPriority 或者addEventListenerWithFixedPriority 方法添加监听时，会对监听器添加已注册标记，使其不能被添加多次：

```
_eventDispatcher->addEventListenerWithSceneGraphPriority(listener1, sprite1);
_eventDispatcher->addEventListenerWithSceneGraphPriority(listener1->clone(), sprite2);
 eventDispatcher->addEventListenerWithSceneGraphPriority(listener1->clone(), sprite3);
```

Cocos2d-x的事件分发机制

移除监听器：

```
// 移除特定监听器
_eventDispatcher->removeEventListener(listener);
// 移除所有监听器
_eventDispatcher->removeAllEventListeners();
```

需要注意的是，使用removeAllEventListeners的时候，该节点的所有的监听将被移除，之后菜单也不能响应，因为它也需要接受触摸事件。

编写事件响应函数

方法一：自定义响应函数，然后使用CC_CALLBACK_X：

```
bool Thunder::onTouchBegan(Touch *touch, Event *unused_event) {
    Vec2 pos = touch->getLocation();
    // do something
    return true;
}
```

onTouchBegan函数接收两个参数，回调函数使用CC_CALLBACK_2

```
auto touchListener = EventListenerTouchOneByOne::create();
touchListener->onTouchBegan = CC_CALLBACK_2(Thunder::onTouchBegan,this);
```

COCOS2DX

CC_CALLBACK_2

是一个宏定义

```
#define CC_CALLBACK_2(__selector__,__target__, ...)
        std::bind(&__selector__, __target__, std::placeholders::_1, std::placeholders::_2, ##__VA_ARGS__
```

C++11特性std::bind、std::placeholders，使得函数的绑定更为方便

详细可参考：

http://www.jellythink.com/archives/773

http://blog.csdn.net/sh15285118586/article/details/47148287

http://shahdza.blog.51cto.com/2410787/1553051

http://en.cppreference.com/w/cpp/utility/functional/placeholders

编写事件响应函数

//方法二：使用lambda函数表达式

```
touchListener->onTouchBegan = [](Touch *touch, Event *unused_event) {
    Vec2 pos = touch->getLocation();
    // do something
    return true;
}
```

事件监听器的5种类型

- EventListenerTouch - 响应触摸事件
- EventListenerKeyboard - 响应键盘事件
- EventListenMouse - 响应鼠标事件
- EventListenerAcceleration - 响应加速度计的事件
- EventListenerCustom - 响应自定义事件

EventListenerTouch- 响应触摸事件

```
//创建单点触摸监听器
auto touchListener = EventListenerTouchOneByOne::create();
```

- onTouchBegan - 响应触摸初始事件
- onTouchMoved - 响应触摸移动事件
- onTouchEnded - 响应触摸结束事件
- onTouchCancelled - 响应触摸中断事件(如突然来电)

EventListenerTouch- 响应触摸事件

```
//创建多点触摸监听器
auto touchListener = EventListenerTouchAllAtOnce::create();
```

- onTouchesBegan - 响应触摸初始事件
- onTouchesMoved - 响应触摸移动事件
- onTouchesEnded - 响应触摸结束事件
- onTouchesCancelled - 响应触摸中断事件

EventListenerKeyboard- 响应键盘事件

```
//创建键盘事件监听器
auto keyboardListener = EventListenerKeyboard::create();
```

- onKeyPressed - 响应键盘按下事件
- onKeyReleased - 响应键盘放开事件
- 通过参数中的KeyCode判断按下哪个按钮

void onKeyPressed(EventKeyboard::KeyCode code, Event* event)

```
void Thunder::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {
  switch (code) {
  case cocos2d::EventKeyboard::KeyCode::KEY_LEFT_ARROW:
  case cocos2d::EventKeyboard::KeyCode::KEY_A:
    movekey = 'A';
    isMove = true;
    break;
  case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
  case cocos2d::EventKeyboard::KeyCode::KEY_D:
    movekey = 'D';
    isMove = true;
    break;
  case cocos2d::EventKeyboard::KeyCode::KEY_SPACE:
    fire();
    break;
  default:
    break;
  }
}
```

EventListenMouse- 响应鼠标事件

```
//创建单点触摸监听器
auto mouseListener = EventListenerMouse::create();
```

- onMouseDown - 响应鼠标点击按下事件
- onMouseUp - 响应鼠标点击松开事件
- onMouseMove - 响应鼠标移动事件
- onMouseScroll - 响应鼠标滚轮事件

EventListenerAcceleration - 响应加速度计的事件

除了触摸，移动设备上一个很重要的输入源是设备的方向，因此大多数设备都配备了加速计，用于测量设备静止或匀速运动时所受到的重力方向。
重力感应来自移动设备的加速计，通常支持X,Y和Z三个方向的加速度感应，所以又称为三向加速计。在实际应用中，可以根据3个方向的力度大小来计算手机倾斜的角度或方向。

EventListenerAcceleration - 响应加速度计的事件

要使用加速度计监听器，必须首先启用加速度计：

```
Device::setAccelerometerEnabled(true);
```

然后创建加速度计监听器

```
auto listener = EventListenerAcceleration::create(
    [](Acceleration* acceleration, Event* event) {
        log("X: %f; Y: %f; Z:%f; ",
            acceleration->x, acceleration->y, acceleration->z);
    }
);
```

加速计信息类Acceleration

```
class CC_DLL Acceleration
    : public Ref
{
public:
    double x;
    double y;
    double z;

    double timestamp;

    Acceleration(): x(0), y(0), z(0), timestamp(0) {}
};
```

EventListenerCustom- 响应自定义事件

- 以上是系统自带的事件类型，事件由系统内部自动触发，如 触摸屏幕，键盘响应等。
- EventListenerCustom 自定义事件，它不是由系统自动触发，而是人为的干涉。

自定义事件订阅

需要使用EventListenerCustom::create()函数来创建，参数一是事件名，在触发事件时会用到。创建同样有两种方式：

```
void Thunder::meet(EventCustom* event)
```

```
void Thunder::addCustomListener() {
    auto meetListener =
        EventListenerCustom::create("meet", CC_CALLBACK_1(Thunder::meet, this));
    _eventDispatcher->addEventListenerWithFixedPriority(meetListener, 1);
}
```

方法二：

```
_eventDispatcher->addCustomEventListener("meet", [](EventCustom* event){
    //do something
});
```

自定义事件的消息传递与发布

```
// 创建一个自定义事件
// EventCustom(string eventName)
EventCustom e("meet");

// 设置需要传递的消息，参数类型为void*
// 这样通过强制类型转换，传递的信息类型就没有限制
// setUserData(void*)
int a = 1;
e->setUserData(&a);

// 发布事件
// dispatchEvent(event*)
_eventDispatcher->dispatchEvent(&e);
```

section2

音乐与音效

好的游戏会俘虏玩家的感官，出色的背景音乐会让玩家更容易沉浸在你的虚拟世界中，应景的音效会让人玩起来更带感。

为此，Cocos2d-x为开发者提供了对声音和音效的支持，能够十分方便地实现音乐与音效的播放、暂停和循环功能。

SimpleAudioEngine

- Cocos2d-x自带的CocosDension库的一个实现
- 提供音乐与音效的播放暂停
- 方便易用

简易使用方法及例子

// 引入头文件

```
#include "SimpleAudioEngine.h"
```

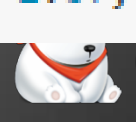
// SimpleAudioEngine是一个单例类，使用时需先获取单例

// 播放背景音乐：

```
auto audio = SimpleAudioEngine::getInstance();

// set the background music and continuously play it.
audio->playBackgroundMusic("mymusic.mp3", true);

// set the background music and play it just once.
audio->playBackgroundMusic("mymusic.mp3", false);
```

// 播放音效：

```
auto audio = SimpleAudioEngine::getInstance();

// play a sound effect, just once.
audio->playEffect("myEffect.mp3", false, 1.0f, 1.0f, 1.0f);
```



简易使用方法及例子

//暂停播放

```
auto audio = SimpleAudioEngine::getInstance();

// pause background music.
audio->pauseBackgroundMusic();

// pause a sound effect.
audio->pauseEffect();

// pause all sound effects.
audio->pauseAllEffects();
```

// 停止播放

```
auto audio = SimpleAudioEngine::getInstance();

// stop background music.
audio->stopBackgroundMusic();

// stop a sound effect.
audio->stopEffect();

// stops all running sound effects.
audio->stopAllEffects();
```

简易使用方法及例子

//恢复播放

```
auto audio = SimpleAudioEngine::getInstance();

// resume background music.
audio->resumeBackgroundMusic();

// resume a sound effect.
audio->resumeEffect();

// resume all sound effects.
audio->resumeAllEffects();
```

// 判断是否正在播放

```
// 返回是否正在播放音乐
// 注意：暂停也算正在播放，只有停止了才算未播放。
bool isBackgroundMusicPlaying();
```

特殊场景

SimpleAudioEngine的API很简单，但是在游戏中使用还是有一些注意事项，尤其是在手机和平板的等移动设备中使用时。

比如在多个APP中切换时应如何处理？或者当你玩着游戏时有电话打进来又该怎么办？

这些异常在制作游戏时都必须提前想好处理方法，不过幸运的是，你能想到的异常SimpleAudioEngine都帮我们做好了，我们只需使用就好。

特殊场景

在AppDelegate.cpp中，注意以下几个方法：

```
// This function will be called when the app is inactive. When comes a phone call,
// it's be invoked too
void AppDelegate::applicationDidEnterBackground() {
    Director::getInstance()->stopAnimation();

    // if you use SimpleAudioEngine, it must be pause
    // SimpleAudioEngine::getInstance()->pauseBackgroundMusic();
}

// this function will be called when the app is active again
void AppDelegate::applicationWillEnterForeground() {
    Director::getInstance()->startAnimation();

    // if you use SimpleAudioEngine, it must resume here
    // SimpleAudioEngine::getInstance()->resumeBackgroundMusic();
}
```

预加载资源

加载音乐和音效通常是一个耗时的过程，为了防止由即时加载产生的延迟导致实际播放与游戏不协调的现象发生，在播放音效和背景音乐之前，记得要预加载音乐文件。另外需要根据不同的平台，选择不同的预加载音乐格式。

```
auto audio = SimpleAudioEngine::getInstance();

// pre-loading background music and effects. You could pre-load
// effects, perhaps on app startup so they are already loaded
// when you want to use them.
audio->preloadBackgroundMusic("myMusic1.mp3");
audio->preloadBackgroundMusic("myMusic2.mp3");

audio->preloadEffect("myEffect1.mp3");
audio->preloadEffect("myEffect2.mp3");

// unload a sound from cache. If you are finished with a sound and
// you wont use it anymore in your game. unload it to free up
// resources.
audio->unloadEffect("myEffect1.mp3");
```

音量控制

wim32平台未实现

https://github.com/cocos2d/cocos2d-x/issues/11867

```
////////////////////////////////////////////////////////////////////////////////
// volume interface
////////////////////////////////////////////////////////////////////////////////

float SimpleAudioEngine::getBackgroundMusicVolume()
{
    return 1.0;
}

void SimpleAudioEngine::setBackgroundMusicVolume(float volume)
{
}

float SimpleAudioEngine::getEffectsVolume()
{
    return 1.0;
}
```

支持平台与格式

CocosDesion支持的音乐格式如下：

平台	支持的常见文件格式	备注
Android	mp3, mid, oggg, wav	可以播放android.media.MediaPlayer所支持的所有格式
iOS	aac, caf, mp3, m4a, wav	可以播放AVAudioPlayer所支持的所有格式
Windows	mid, mp3, wav	无

支持平台与格式

CocosDesion支持的音效格式如下：

平台	支持的常见文件格式	备注
Android	oggg, wav	对wav的支持不完美
iOS	caf, m4a	可以播放Cocos2d-iPhone CocosDesion所支持的所有格式
Windows	mid, wav	无

THE END

THANKS FOR WATCHING