Cocos2d-x用户事件

1. 事件处理机制

事件处理机制一般都有三个重要的角色：事件、事件源和事件处理者。事件源是事件发生的场所，通常是各个视图或控件；事件处理者是接收事件并对其进行处理的一段程序

1. 事件

事件类是Event（是Ref的子类），其子类主要有EventTouch(触摸事件)、EventMouse(鼠标事件)、EventKeyboard(键盘事件)、EventAcceleration(加速度事件)和EventCustom(自定义事件)

1. 事件源

事件源是Cocos2d-x中的精灵、层、菜单等节点对象

1. 事件处理者

Cocos2d-x中的事件处理者是事件监听器类EventListener,其子类主要有

EventListenerTouchOnoByOne(单点触摸事件监听器)、EventListenerTouchAllAtOnce(多点事件监听器)、

EventListenerKeyboard(键盘事件监听器)、EventListenerMouse(鼠标事件监听器)、EventListenerAcceleration(加速度事件监听器)和EventListenerCustom(自定义事件监听器)

1. 事件分发器

事件分发器负责注册监听器（事件和事件监听器建立联系），注销监听器和事件分发

Director::getInstance()->getEventDispatcher()可以获得事件分发器对象

分发监听器：

1. void addEventListenerWithFixedPriority(EventListener \*listener,int fixedPriority)

//指定固定的事件优先级注册监听器，事件优先级决定事件相应的优先级别，值越小优先级越高

1. void addEventListenerWithSceneGraphPriority

(EventListener\*listener,Node \*node);

//精灵显示优先级作为事件优先级，参数bode是要触摸的精灵对象

撤销事件监听器

1. void removeEventListener(EventListener \*listener)//注销指定的事件监听器
2. void removeCustomEventListeners(const std::string& customEventName);//注销自定义事件监听器
3. void removeAllEventListeners()//注销所有事件监听器，注意使用这一操作后菜单也不能响应事件了，因为它也需要接受触摸事件
4. 触摸事件

①时间方面

触摸事件有两个事件监听器：EventListenerTouchOneByOne和

EventListenerTouchAllAtOnce分别对应单点触摸和多点触摸

EventListenerTouchOneByOne中触摸事件响应属性

1. std::function<bool(Touch\*,Event\*)>onTouchBegan

\\当一个手指触碰屏幕时回调该属性所指定的函数。若返回true,则可以回调后面的两个属性(onTouchMoved和onTouchEnded)所指定的函数，否则不回调

void touchMoved(Touch\* touch, Event\* event);

void touchEnded(Touch\* touch, Event\* event);

//以上两个回调函数需要自己定义

1. std::function<void(Touch\*,Event\*)>onTouchMoved

//当一个手指在屏幕移动时回调该属性所指定的函数

1. std::function<void(Touch\*,Event\*)>onTouchEnded

//当一个手指离开屏幕时回调该属性所指定的函数

（4）std::function<void(Touch\*,Event\*)>onTouchCanalled

//当单点触摸事件被取消时回调该属性所指定的函数

EventListenerTouchAllAtOnce中触摸事件响应属性

1. std::function<void(const std::vector<Touch\*>&,Event\*)>onTouchesBegan

//当多个手指触碰屏幕是回调该属性所指的函数

（2）std::function<void(const std::vector<Touch\*>&,Event\*)>

onTouchesMoved//当多个手指触碰在屏幕上移动时回调该属性所指定的函数

（3）std::function<void(const std::vector<Touch\*>&,Event\*)>

onTouchesEnded//当多个手指离开屏幕时回调该属性所指定的函数

（4）std::function<void(const std::vector<Touch\*>&,Event\*)>

onTouchesCancelled//当多点触摸事件被取消时回调该属性所指定的函数

注：上述的std::function<void(const std::vector<Touch\*>&,Event\*)>

在定义的时候直接写成void就可

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

具体建立步骤：

auto listener=EventListenerTouchOneByOne::create();

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

bool HelloWorld::touchBegan(Touch\*touch,Event \*event)

{

……return false;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

②空间方面

1. 获得触摸点之前的位置信息

Vec2 getPreviousLocationInView()//UI坐标

Vec2 getPreviousLocation()//OpenGL坐标

1. 获得触摸点当前的位置信息

Vec2 getLocationInView()//UI坐标

Vec2 getLocation()//OpenGL坐标

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

重要补充：在注册和销毁监听器的时候，可以在这一层类声明中声明两个函数

virtual void onEnter()//注册监听器并初始化

virtual void onExit();//销毁监听器和释放一些资源

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

事件与监听器的建立方法

（1）在要建立监听器的场景的类声明中 添加

virtual void onEnter();//注册监听器

virtual void onExit();//注销监听器

（2）在类声明中添加事件回调函数

如bool touchBegan(Touch\* touch,Event \*event);

void touchMoved(Touch \*touch,Event \*event);

void touchEnded(Touch\* touch,Event \*event);

（3）在onEnter()和onExit()中注册、注销监听器

如：

void HelloWorld::onEnter()

{

Scene::onEnter();

log("HelloWorld onEnter");

auto listener = EventListenerTouchOneByOne::create();

listener->setSwallowTouches(true);

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

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

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

//注册监听器

EventDispatcher\* eventDispatcher = Director::getInstance()->getEventDispatcher();

eventDispatcher->addEventListenerWithSceneGraphPriority(listener, getChildByTag(kBoxA\_Tag));

eventDispatcher->addEventListenerWithSceneGraphPriority(listener->clone(), getChildByTag(kBoxB\_Tag));

eventDispatcher->addEventListenerWithSceneGraphPriority(listener->clone(), getChildByTag(kBoxC\_Tag));

}

void HelloWorld::onExit()

{

Scene::onExit();

log("HelloWorld onExit");

Director::getInstance()->getEventDispatcher()->removeAllEventListeners();

}

（4）给出2步骤中回调函数的定义

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

3.Lamba表达式