CCAction 类

CCAction的基类

CCAction (void)
virtual ~CCAction (void)

char * description ()

virtual CCObject * copyWithZone (CCZone *pZone)

virtual bool isDone (void) 动作是否完成

virtual void startWithTarget (CCNode *pTarget)

virtual void stop (void)
动作完成后自动调用，不能手动的用一个动作调用stop() 需要的时候可以在CCSpeed, CCFollow, CCActionEase, CCSequence, CCRepeat, CCSpawn, CCReverseTime, 和 CCAnimate中实现它

virtual void step (ccTime dt)
每帧都会调用 一般不用重写 如果真的需要就在CCSpeed, CCFollow, CCActionInstant, CCActionInterval, 和CCRepeatForever中重新实现它) 参数 每帧时间差

virtual void update (ccTime time)
在动作过程中调用一次，需要的时候重新实现 参数是0到1 0这个方法在动作刚开始的时候调用 0.5方法在进行到一半的时候调用 1方法在动作完成的时候调用

CCNode * getTarget (void)

获得执行某个动作的节点指针 CCNode* 语法 CCNode *mynode = p_action->getTarget();返回节点指针

void setTarget (CCNode *pTarget)
设置动作的执行者 参数 节点指针 目标对象

CCNode * getOriginalTarget (void)
得到执行动作的上一个实体 返回值 CCNode *节点对象

void setOriginalTarget (CCNode *pOriginalTarget)
设置动作原来的执行者目标，可以为空 参数CCNode *节点对象

int getTag (void)
获得动作标记

void setTag (int nTag)
设置动作的标记

static CCAction * action ()
生成一个动作
例子
CCAction *newaction = CCAction::action();
newaction = CCRepeatForever::actionWithAction((CCActionInterval*)p_action);
myactionsprite->runAction(newaction);

CCNode * m_pOriginalTarget
动作原来的执行目标
例子
CCNode *originaltarget = repeat->getOriginalTarget();
originaltarget->setPosition(ccp(10, 10));
repeat->setOriginalTarget(originaltarget);

CCNode* m_pTarget

int m_nTag

动作标记，私有属性通过方法访问

CCActionManager 类

CCActionManager (void)

~CCActionManager (void)

bool init (void)

初始化函数用来在CCScheduler里边加载CCActionManager对象

例子代码

CCActionManager * gSharedManager = new CCActionManager();

gSharedManager->init();

void addAction (CCAction *pAction, CCNode *pTarget, bool paused) 通过ActionManager给CCNode对象pTarget添加一个动作pAction，一般在CCNode自身的runAction函数里边调用

void removeAllActions (void)

移除CCAcionManager::sharedManager()所有的Action动作，包括所有对象的所有动作

语法 CCAcionManager::sharedManager()->removeAllActions()

void removeAllActionsFromTarget (CCObject *pTarget)

从一个特定的对象中移除所有的Action，一般用在CCNode的stopAllAction函数中

例子

```
CCActionManager::sharedManager()->removeAllActionsFromTarget(t  his);
```

void removeAction (CCAction *pAction)

移除某个具体的pAction,会从所有的

CCAction *pActcion = this->getActionByTag(0);

CCActionManager::sharedManager()-> removeAction(pAction);

参数 pAction: CCAction对象，指具体的一个动作对象，比如CCMoveTo, CCScale

void removeActionByTag (unsigned int tag, CCObject *pTarget)

从一个具体的对象pTarger中移除指定的Action, Action的标识为tag

例子

```
void CCNode::stopActionByTag(int tag){
    CCAssert( tag != kCCActionTagInvalid, "Invalid tag");
    CCctionManager::sharedManager()->removeActionByTag(tag, this);
}
```

CCAction * getActionByTag (unsigned int tag, CCObject *pTarget)

从某个pTarget中获取特定标识(tag)的Action 对象，常用在CCNode对象 的getActionByTag(int tag)函数中

例子

```
CCAction * CCNode::getActionByTag(int tag)    {
    CCAssert( tag != kCCActionTagInvalid, "Invalid tag");
    return CCActionManager::sharedManager()->getActionByTag(tag, this);
 }
```

unsigned int numberOfRunningActionsInTarget (CCObject *pTarget)

返回行动的数量，在某些特定的目标上运行的。 组合的动作都算作1行动。例如：您正在运行的7动作上的1序列，它会返回1。如果您正在运行的2动作上的7序列，它会返回7。

```
unsigned int CCNode::numberOfRunningActions(){
                                        return
CCActionManager::sharedManager()->numberOfRunningActionsInTarget(this);
}
```

void pauseTarget (CCObject *pTarget)

暂停pTarget 的所有动作，常用于CCNode 的pauseSchedulerAndActions函数

例子

```
void CCNode::pauseSchedulerAndActions(){
    CCScheduler::sharedScheduler()->pauseTarget(this);
    CCActionManager::sharedManager()->pauseTarget(this)
```

}

void resumeTarget (CCObject *pTarget)
恢复特定的pTarget对象的所有动作， 常用于， CCNode对resumeSchedulerAndActions 函数

void purgeSharedManager (void)
清空sharedManager对象

static CCActionManager * sharedManager (void)

void removeActionAtIndex (unsigned int uIndex, struct _hashElement *pElement)
从ActionManager中移除某个索引(uIndex)的动作，用于CCActionManager对象的 removeAction函数中， 属于protected类型

void deleteHashElement (struct _hashElement *pElement)
从CCActionManager对象中释放pElement相关的 Actions, protected类型， 自身调 函数，常用于在判断CCObject对象的Action数目为0时调用

void actionAllocWithHashElement (struct _hashElement *pElement)
用于复制pElement对象的所有Action对象， protected类型，如果pElement无Action， 自动为它创建

void update (ccTime dt)
CCActionManager的更新函数，用于更新和处理所有对象的状态,在 init 函数中由CCScheduler来启动
例子 `CCActionManager::sharedManager()->update();`

struct _hashElement * m_pTargets

struct _hashElement * m_pCurrentTarget

bool m_bCurrentTargetSalvaged

CCAnimation 类

使用一个CCAnimation对象可以CCSprite对象上执行动画。 CCAnimation对象包含CCSpriteFrame对象，与帧之间可能出现的延误。你可以使用CCAnimate的方法 播放CCAnimation对象。

const char * getName (void)

获取对应CCAnimation对象的名称标识

例子代码 CCAnimation::animation()->getName();

返回值 const char*，对应CCAnimation对象的名称标识

void setName (const char *pszName)

为相应的 CCAnimation 对象设置名称标识 void setName (const char* pszName)

参数 const char *pszName，将要赋给 CCAnimation 对象的字符串

例子代码 CCAnimation::animation()->setName("Test");

float getDelay (void)

获取动画桢之间的延迟时间

例子代码 CCAnimation::animation()->getDelay(); 返回值 float类型，延迟时间

void setDelay (float fDelay)

设置动画桢之间的延迟时间

例子代码 CCAnimation::animation()->setDelay(0.1f);

CCMutableArray< CCSpriteFrame * > * getFrames (void)

获取CCAnimation对象中的动画桢数组返回值 CCMutableArray* ，存放CCSpriteFrame的数组

例子代码 CCAnimation::animation()->getFrames();

void setFrames (CCMutableArray< CCSpriteFrame * > *pFrames)

~CCAnimation (void)

bool initWithFrames (CCMutableArray< CCSpriteFrame * > *pFrames)

使用动画桢初始化CCAnimation对象参数 (CCMutableArray< CCSpriteFrame * > *pFrames)，存放CCSPriteFrame的h动画桢数组。

例子代码 CCMutaleArray<CCSpriteFrame*>* frames;
CCAnimation *pAnimation = new CCAnimation();
pAnimation->initWithFrames(frames);
pAnimation->autorelease();

bool initWithFrames (CCMutableArray< CCSpriteFrame * > *pFrames, float delay)

使用动画桢和延迟时间初始化CCAnimation对象 参数 (CCMutableArray< CCSpriteFrame * > *pFrames)，存放CCSPriteFrame的h动画桢数组。 float delay，延迟时间

void addFrame (CCSpriteFrame *pFrame)

为CCAnimation对象增加动画桢

例子代码 CCTexture2D *pTexture = CCTextureCache::sharedTextureCache()->addImage("Test.png");
CCRect rect = CCRectZero;
rect.size = pTexture->getContentSize();
CCSpriteFrame *pFrame = CCSpriteFrame::frameWithTexture(pTexture, rect);
CCAnimation::animation()->addFrame(pFrame);

void addFrameWithFileName (const char *pszFileName)

从文件名为CCAnimation对象添加动画桢

例子代码 CCAnimation::animation()->addFrameWithFileName("Test.png");

void addFrameWithTexture (CCTexture2D *pobTexture, const CCRect &rect)

使用CCTexture2D对象和CCRect对象为CCAnimation对象添加动画桢例子代码

CCTexture2D *pTexture = CCTextureCache::sharedTextureCache()->addImage("Test.png");
CCRect rect = CCRectZero;

```
rect.size = pTexture->getContentSize();
CCAnimation::animation()->addFrameWithTexture(pTexture,&rect);
```

bool init (void)
初始化函数

例子代码
```
CCAnimation *pAnimation = new CCAnimation();
pAnimation->init();
pAnimation->autorelease();
```

static CCAnimation * animation (void)
静态方法，创建并初始化一个CCAnimation对象

static CCAnimation * animationWithFrames (CCMutableArray< CCSpriteFrame * > *frames)
使用动画桢数组，创建并初始化一个CCAnimation对象 参数是存放CCSpriteFrame对象的数组

static CCAnimation * animationWithFrames (CCMutableArray< CCSpriteFrame * > *frames, float delay)
使用动画桢数组和延迟时间，创建并初始化一个

std::string m_nameStr
CCAnimation对象的字符串名称标识

float m_fDelay
CCAnimation对象的动画桢之间的延迟时间

CCMutableArray< CCSpriteFrame * > * m_pobFrames
存放 CCAnimation 的动画桢的数组

CCAnimationCache 类

单独的动画管理。 它保存动画到缓存中。如果你想保存动画到缓存中，你应该使用这个类。 v0.99.5之前，他们的推荐方式是保存在CCSprite上。 v0.99.5以后，你应该使用这个类。

~CCAnimationCache ()

CCAnimationCache ()

void addAnimation (CCAnimation *animation, const char *name)
添加一个CCAnimation对象，并设置一个名字 参数 CCAnimation * animation，要添加的CCAnimation对象指针 const char * name，为要添加的CCAnimation对象设置的名称标识
例子代码 CCAnimationCache::sharedAnimationCache()->addAnimation(CCAnimation::animation(), "Test");

void removeAnimationByName (const char *name)
从缓存中通过名称标识删除相应的动画数据 参数 const char *name，对应的动画数据的名称标识

CCAnimation * animationByName (const char *name)
通过名称标识从缓存中获取动画数据 参数 const char *name，对应的动画数据的名称标识。
例子代码 CCAnimationCache::sharedAnimationCache()->animationByName("Test");

bool init (void)

static CCAnimationCache * sharedAnimationCache (void)
静态方法，初始化并返回一个全局的静态CCAnimationCache对象

static void purgeSharedAnimationCache (void)
清除 CCAnimationCache, 并且释放所有的 CCAnimation 对象和共享的 CCAnimationCache 的实例对象
例子代码 CCAnimationCache::purgeSharedAnimationCache();

CCArray 类

~CCArray ()

bool init ()

bool initWithCapacity (unsigned int capacity) 按规定的容量初始化一个对象
例子代码
```
CCArray* tempArray = CCArray::array();
tempArray->initWithCapacity(5);
```

bool initWithArray (CCArray *otherArray)

unsigned int count ()

unsigned int capacity ()
返回CCArray对象的容量

unsigned int indexOfObject (CCObject *object)
返回CCArray对象的容量

CCObject * objectAtIndex (unsigned int index)
返回数组内位于index 的对象

CCObject * lastObject ()

CCObject * randomObject ()
在数组内随机取一个对象

bool containsObject (CCObject *object)

void addObject (CCObject *object)

void addObjectsFromArray (CCArray *otherArray)
将另一个数组里的元素加入当前数组

void insertObject (CCObject *object, unsigned int index)

void removeLastObject ()

void removeObject (CCObject *object)

void removeObjectAtIndex (unsigned int index)
删除数组内位于index位置的对象

void removeObjectsInArray (CCArray *otherArray)
删除本数组与otherArray相同的对象

void removeAllObjects ()
删除数组内的所有对象 清空

void fastRemoveObject (CCObject *object)
快速删除数组内的一个对象

void fastRemoveObjectAtIndex (unsigned int index)

void exchangeObject (CCObject *object1, CCObject *object2)
交换数组内object1，和object2的位置，如果数组内不存在object1或object2，函数无效

void exchangeObjectAtIndex (unsigned int index1, unsigned int index2)
交换数组内位于index1和index2位置的对象

void reverseObjects ()
反向排列数组内的数据

void reduceMemoryFootprint ()
减少数组的占用空间

static CCArray * array ()
静态声明一个CCArray对象

static CCArray * arrayWithCapacity (unsigned int capacity)
静态声明一个CCArray对象，并初始化它的大小为capacity

static CCArray * arrayWithArray (CCArray *otherArray)
用另一个数组内的对象初始化一个数组

ccArray * data

CCCamera 类

CCCamera 应用于每个 CCNode 中。 在从不同视野观察时起作用，OpenGL 中的 gluLookAt 函数用来定位摄像机。 假如这个物体通过绽放、旋转、或者平移，那么这些操作将修改摄像机。 重要信息：使用摄像机或者操作旋转、绽放、平移等属性，你只能选其中一个，假如你使用摄像机，世界坐标就会失去作用。 局限性： 几种节点，如 CCParallaxNode，CCParticle 使用世界节点坐标，并且他们并不能正常工作，假如你使用摄像机移动他们（或者他们的任何父类）。 在成批的结点上（即一个结点上包含多个节点）如 CCSprite 对象不会正常工作当他们都从属于一个 CCSpriteBatchNode 对象时。推荐你只在创建 3d 特效时使用。对于 2d 特效来说，用 CCFollow 动作或者平移、绽放、旋转更好。

char * description (void)
返回摄像机的中心点的描述信息。

void setDirty (bool bValue)
设置当前摄像机为脏状态

bool getDirty (void)
返回当前摄像机是否为脏状态,内联函数。

void restore (void)

设置摄像机到默认位置。 例子代码 CCCamera * pCamera; pCamera->restore ();

void locate (void)

用 gluLookAt 设置摄像机的信息，参数为 eye, center,up_vector,分别为眼的位置，观察的中心，以及向上的向量，具体可参见 openGL 中的 gluLookAt 函数。
例子代码 CCCamera * pCamera; pCamera->locate ();

void setEyeXYZ (float fEyeX, float fEyeY, float fEyeZ)

用浮点数设置 eye 的位置

void setCenterXYZ (float fCenterX, float fCenterY, float fCenterZ)

用浮点数设置中心的坐标

void setUpXYZ (float fUpX, float fUpY, float fUpZ)

用浮点数设置相机实例的 UP 值

void getEyeXYZ (float *pEyeX, float *pEyeY, float *pEyeZ)

void getCenterXYZ (float *pCenterX, float *pCenterY, float *pCenterZ)

void getUpXYZ (float *pUpX, float *pUpY, float *pUpZ)

static float getZEye ()

float m_fEyeX

float m_fEyeY

float m_fEyeZ

float m_fCenterX

float m_fCenterY

float m_fCenterZ

float m_fUpX

float m_fUpY

float m_fUpZ

bool m_bDirty

CCSpriteFrame 类

一个 CCSpriteFrame 有： 纹理：将由 CCSprite 的 CCTexture2D 矩形：纹理的区域 您可以修改 CCSprite 的播放桢， CCSprite ： CCSpriteFrame *frame = CCSpriteFrame::frameWithTexture(texture, rect, offset); sprite->setDisplayFrame(frame);

const CCRect & getRectInPixels (void)

获取 CCSpriteFrame 的像素区域

例子代码 CCSpriteFrame *test = new CCSpriteFrame(); test->getRectInPixels();

void setRectInPixels (const CCRect &rectInPixels)

设置 CCSpriteFrame 对象的像素区域

例子代码

CCSpriteFrame* test = new CCSPriteFrame(); CCRect testRect = CCRectZero; test->setRectInPixels(&testRect);

bool isRotated (void)

获取 CCSpriteFrame 是否经过旋转

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame(); pFrame->isRotated();

void setRotated (bool bRotated)

设置 CCSpriteFrame 旋转属性

例子 CCSpriteFrame* pFrame = new CCSpriteFrame(); pFrame->setRotated(true);

const CCRect & getRect (void)

获取 CCSpriteFrame 的作用区域

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame(); pFrame->getRect();

void setRect (const CCRect &rect)

设置 CCSpriteFrame 的作用区域

例 CCSpriteFrame* pFrame = new CCSpriteFrame(); pFrame->setRect(RectZero);

const CCPoint & getOffsetInPixels (void)

获取 CCSpriteFrame 的偏移

void setOffsetInPixels (const CCPoint &offsetInPixels)

设置 CCSpriteFrame 的偏移

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame();

CCPoint sPos = CCPointMake(1, 1);

pFrame->setOffsetInPixels(&sPos);

void setOriginalSizeInPixels (const CCSize &sizeInPixels)

设置裁减后图片的初始大小

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame();

CCSize sSize = CCSizeMake(10, 10);

pFrame->setOriginalSizeInPixels(&sSize);

const CCSize & getOriginalSizeInPixels (void)

获取剪裁后的图像的原始大小

CCTexture2D * getTexture (void)

获取 CCSpriteFrame 的纹理图片 返回值 CCTexture2D*,CCTexture2D 对象指针

例 CCSpriteFrame* pFrame = new CCSpriteFrame(); pFrame->getTexture();

void setTexture (CCTexture2D *pobTexture)

设置 CCSpriteFrame 的纹理图片 参数： 纹理对象指针

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame();

CCTexture2D* pTex = new CCTexture2D();

pFrame->setTexture(pTex);

bool initWithTexture (CCTexture2D *pobTexture, const CCRect &rect)

使用纹理对象和区域对象初始化CCSpriteFrame对象 参数：纹理对象、区域对象

例子代码 CCSpriteFrame* pFrame = new CCSpriteFrame();

CCTexture2D* pTex = new CCTexture2D();

CCRect sRect = CCRectMake(0,0, 5, 5);

pFrame->initWithTexture(pTex,&sRect);

bool initWithTexture (CCTexture2D *pobTexture, const CCRect &rect, bool rotated, const CCPoint &offset, const CCSize &originalSize)

对象初始化属性 参数：纹理对象、区域对象、旋转属性、偏移属性、大小属性

static CCSpriteFrame * frameWithTexture (CCTexture2D *pobTexture, const CCRect &rect)

创建并初始化一个全局的静态 CCSpriteFrame 对象 参数：纹理对象、区域对象。

例子代码 CCTexture2D* pTex = new CCTexture2D();

CCRect sRect = CCRectMake(0,0, 5, 5);

CCSpriteFrame::frameWithTexture(pTex,&sRect);

CCRect m_obRectInPixels

CCSpriteFrame 的像素矩形区域

bool m_bRotated

是否要进行旋转

CCRect m_obRect

CCSpriteFrame 的矩形区域

CCPoint m_obOffsetInPixels

CCSpriteFrame 的像素偏移

CCSize m_obOriginalSizeInPixels

裁减后的图片初始像素大小

CCTexture2D * m_pobTexture

CCSpriteFrame 的纹理对象

CCTimer 类

轻量级的计时器

CCTimer (void)

ccTime getInterval (void)

void setInterval (ccTime fInterval)

bool initWithTarget (CCObject *pTarget, SEL_SCHEDULE pfnSelector)

bool initWithTarget (CCObject* pTarget, SEL_SCHEDULE pfnSelector, ccTime fSeconds)

bool initWithScriptFuncName (const char *pszFuncName, ccTime fSeconds)

void update (ccTime dt)

static CCTimer* timerWithTarget (CCObject* pTarget, SEL_SCHEDULE pfnSelector)

static CCTimer* timerWithScriptFuncName (const char *pszFuncName, ccTime fSeconds)

static CCTimer* timerWithTarget (CCObject* pTarget, SEL_SCHEDULE pfnSelector, ccTime fSeconds)

SEL_SCHEDULE m_pfnSelector

ccTime m_fInterval

std::string m_scriptFunc

CCObject * m_pTarget

ccTime m_fElapsed

CCTouch 类

CCTouch ()

默认无参构造函数

例子代码 CCTouch *pTouch = new CCTouch();

CCPoint locationInView (int nViewId)

获取触摸对象在 View 中的坐标 CCPoint locationInView (int nViewId)

例子代码 CCTouch *ptouch = new CCTouch(0,0,0);

CCPoint point = ptouch->locationInView(ptouch->view());

point = CCDirector::sharedDirector()->convertToGL(point);

CCPoint previousLocationInView (int nViewId)

获取触摸对象在 View 中的前一次坐标 参数：int nViewId View 的索引值

例子代码 CCTouch *ptouch = new CCTouch(0,0,0);

CCPoint point = ptouch-> previousLocationInView(ptouch->view());

point = CCDirector::sharedDirector()->convertToGL(point);

int view ()

返回 CCTouch 对象的 View 索引值

例子代码 CCTouch *pTouch = new CCTouch();

pTouch->view();

int id ()

返回 CCTouch 对象的唯一标识

例子代码 CCTouch *pTouch = new CCTouch();

pTouch->id();

void SetTouchInfo (int nViewId, float x, float y, int iID=0)

设置 CCTouch 对象的属性 参数 int nViewId，CCTouch 对象响应的 View 索引值。

float x，CCTouch 对象的 x 坐标。float y，CCTouch 对象的 y 坐标。int iID = 0， CCTouch 对象的唯一 ID 标识。

例子代码
```
CCTouch *pTouch = new CCTouch();
int iViewId = 0;      float posX = 0.0f;
float posY = 0.0f;    int unUsedIndex = 0;
pTouch->SetTouchInfo(iViewId,posX,posY,unUsedIndex);
```

CCTouchDispatcher 类

CCTouchDispatcher 是触摸分发器，该类用单例模式处理所有的触摸事件，该分发器将事件分发给注册过的 TouchHandlers，TouchHandlers 有两种不同的类型： 标准触摸句柄(Standard Touch Handlers) 目标触摸句柄(Target Touch Handlers) 标准触摸句柄就像 CocoaTouch 里的句柄一样，所有的触摸集合都会传送给该句柄，而目标触摸句柄一次仅仅接受一个触摸，并且他们可以截获这些触摸(避免事件的传播) 首先，分发器发送接收到的触摸给指定的目标触摸句柄，这些触摸事件可以被该目标触摸句柄截获。如果有剩余的触摸事件，这些触摸事件将被发送给标准触摸句柄

bool isDispatchEvents (void)
判断事件是否将被分发，默认:是

void setDispatchEvents (bool bDispatchEvents)
设置事件是否被分发

例子
```
CCTouchDispatcher *dispatch = CCTouchDispatcher::sharedDispatcher();
if(dispatch->isDispatchEvents()){
    CCLog("事件将被分发");
}
dispatch->setDispatchEvents(0);
if(!dispatch->isDispatchEvents()){
    CCLog("事件将不被分发");
}
```

void addStandardDelegate (CCTouchDelegate *pDelegate, int nPriority)
添加标准的 touché处理机制(像 cocoa touch 中的 touch 处理机制)，触摸集将会传递

给代理
语法 myDispatcher->addStandardDelegate(touchDelegate,1);)
例子 CCTouchDispatcher::sharedDispatcher()->addStandardDelegate(this,0);

void addTargetedDelegate (CCTouchDelegate *pDelegate, int nPriority, bool bSwallowsTouches)
添加一个具体的目标触摸代理到分发列表中，注意：pDelegate 将被 retained
语法 myDispatcher->addTargetedDelegate(touchDelegate,1,yes);
参数 bSwallowsTouches:触摸时间是否被该目标截获

例子代码 -(void) registerWithTouchDispatcher{
[[TouchDispatcher sharedDispatcher] addTargetedDelegate:self priority: INT_MIN+1 swallowsTouches:YES];
}

void removeDelegate (CCTouchDelegate *pDelegate)
移除指定的触摸代理，指定的代理将被释放
例子代码 -(void) end{
[[CCTouchDispatcher sharedDispatcher] removeAllDelegates];
[super end];
}

void removeAllDelegates (void)
移除所有触摸代理，一般在重置导演类的时候调用
语法 CCTouchDispatcher::sharedDispatcher()->removeAllDelegates();(void
例子代码 -(void) end{
[[CCTouchDispatcher sharedDispatcher] removeAllDelegates];
[super end];
}

void setPriority (int nPriority, CCTouchDelegate *pDelegate)
设置指定代理的优先级,nPriority 越小，优先级越高
语法 myDispatcher->setPriority(1,myDelegate);

void touches (CCSet *pTouches, CCEvent *pEvent, unsigned int uIndex)

分发事件

例子代码 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{

if(dispatchEvents){

[self touches:touches withEvent:event withTouchType:kCCTouchBegan];

}

}

virtual void touchesBegan (CCSet *touches, CCEvent *pEvent)

虚函数，实现 EGLTouchDelegate 中相应的方法，检测手指触摸开始

参数 touches 集合类型 pEvent 事件类型

语法 m_pDelegate->touchesBegan(set, NULL);

例子代码

void CCEGLView::touchesBegan(CCSet *set){

if (m_pDelegate) {

m_pDelegate->touchesBegan(set, NULL);

}

}

virtual void touchesMoved (CCSet *touches, CCEvent *pEvent)

虚函数，实现 EGLTouchDelegate 中相应的方法，检测手指触摸移动的过程

virtual void touchesMoved (CCSet* touches,CCEvent * pEvent) 参数、例子同上

virtual void touchesEnded (CCSet *touches, CCEvent *pEvent)

虚函数，实现 EGLTouchDelegate 中相应的方法，检测手指触摸结束

参数、例子同上

virtual void touchesCancelled (CCSet *touches, CCEvent *pEvent)

虚函数，实现 EGLTouchDelegate 中相应的方法，检测手指触摸失效

参数、例子同上

static CCTouchDispatcher * sharedDispatcher ()

返回静态的 CCTouchDispatcher 的单例，函数内部实现用单例模式

语法 CCTouchDispatcher::sharedDispatcher()->removeAllDelegates();

例 子 CCTouchDispatcher::sharedDispatcher()->setDispatchEvents(false); 或 者 CCTouchDispatcher *touchDispatcher = [CCTouchDispatcher sharedDispatcher];

void forceRemoveDelegate (CCTouchDelegate *pDelegate)

强制移除指定代理 void forceRemoveDelegate(CCTouchDelegate* pDelegate)

例子 if(! locked) {

[self forceRemoveDelegate:delegate];

}

else {

[handlersToRemove addObject:delegate];

toRemove = YES;

}

void forceAddHandler (CCTouchHandler *pHandler, CCMutableArray< CCTouchHandler * > *pArray)

强制添加句柄 参数：pHandler:CCTouchHandler 类型指针 pArray:动态指针

语法 forceAddHandler(pHandler, m_pTargetedHandlers);

例子代码 if (! m_bLocked){

forceAddHandler(pHandler, m_pStandardHandlers);

}

void forceRemoveAllDelegates (void)

强制移除所有的代理

例子代码 CCTouchDispatcher::sharedDispatcher()->removeAllDelegates();

void rearrangeHandlers (CCMutableArray< CCTouchHandler * > *pArray)

重新排列 ccTouchHandler 参数:CCMutableArray 类型数组指针

语法 this->rearrangeHandlers(m_pTargetedHandlers);

例 子 代 码 CCTouchDispatcher::sharedDispatcher()-> rearrangeHandlers (m_pTargetedHandlers);

CCTouchHandler * findHandler (CCMutableArray< CCTouchHandler * > *pArray, CCTouchDelegate *pDelegate)
在 pArray 句柄数组中找出指定代理 pDelegate 的处理句柄
参数：数组指针 、 CCTouchDelegate 类型代理
返回值 CCTouchHandler 类型，返回指定代理的处理句柄

m_pTargetedHandlers 方法： 分发器发送收到的触摸事件给目标触摸集。这些触摸集可能被目标触摸处理句柄截获，如果有剩余的触摸，剩余的触摸将被发送给标准触摸句柄

CCMutableArray< CCTouchHandler * > * m_pTargetedHandlers
参数：CCMutableArray 类型数组指针
bool m_bLocked

bool m_bToAdd

bool m_bToRemove

CCMutableArray< CCTouchHandler * > * m_pHandlersToAdd

struct _ccCArray * m_pHandlersToRemove

bool m_bToQuit

bool m_bDispatchEvents

struct ccTouchHandlerHelperData m_sHandlerHelperData [ccTouchMax]

CCTouchHandler 类
包含事件处理器（event handler)的代理和优先级的对象。

virtual ~CCTouchHandler (void) 析构函数
CCTouchDelegate * getDelegate () 获取代理对象

void setDelegate (CCTouchDelegate *pDelegate) 设置一个代理对象

int getPriority (void) 获取事件处理器的调用优先级。

void setPriority (int nPriority) 设置事件处理器的优先级。

int getEnabledSelectors (void) 获取可用的 selector

void setEnalbedSelectors (int nValue) 设置可用的 selectors

virtual bool initWithDelegate (CCTouchDelegate *pDelegate, int nPriority) 初始化一个 CCTouchHandler 对象，虚函数，参数 pDelegate 为一个 CCTouchDelegate 对象，nPriority 为要设置的优先级。

static CCTouchHandler * handlerWithDelegate (CCTouchDelegate *pDelegate, int nPriority) 创建一个 CCTouchHandler 对象，静态成员函数，参数 pDelegate 为一个 CCTouchDelegate 对象，nPriority 为要设置的优先级。

CCTouchDelegate * m_pDelegate

int m_nPriority

int m_nEnabledSelectors

CCSequence 类

顺序执行，用来把一系列动作组合起来，一个接一个执行。

~CCSequence (void)

bool initOneTwo (CCFiniteTimeAction *pActionOne, CCFiniteTimeAction *pActionTwo)

virtual CCObject * copyWithZone (CCZone *pZone)

virtual void startWithTarget (CCNode *pTarget)

virtual void stop (void)

virtual void update (ccTime time)

virtual CCActionInterval * reverse (void)

static CCFiniteTimeAction * actions (CCFiniteTimeAction *pAction1,...)

static CCFiniteTimeAction * actionsWithArray (CCArray *actions)

static CCSequence * actionOneTwo (CCFiniteTimeAction *pActionOne, CCFiniteTimeAction *pActionTwo)

CCFiniteTimeAction * m_pActions [2]

ccTime m_split

int m_last

CC Layer 类

CCLayer 是 CCNode 的子类 ，实现了 TouchEventsDelegate 接口，继承了 CCNode 所有的特性，并且附加了一些自己的特性，它能够接收 iPhone 的触摸事件，也能够接收 Accelerometer 的输入

CCLayer ()
virtual ~CCLayer ()
bool init ()
virtual void onEnter ()
onEnter 方法会在该 CCNode 初始化后调用。 如果使用了过渡场景，则 onEnter 方法会在过渡效果开始时调用。
例子代码

```
void CCLayer::onEnter(){
    if (m_bIsTouchEnabled){
        this->registerWithTouchDispatcher();
    }
    // then iterate over all the children
    CCNode::onEnter();
        // add this layer to concern the Accelerometer Sensor
        if (m_bIsAccelerometerEnabled){
            CCAccelerometer::sharedAccelerometer()->setDelegate(this);
        }
        // add this layer to concern the kaypad msg
        if (m_bIsKeypadEnabled){
            CCKeypadDispatcher::sharedDispatcher()->addDelegate(this);
        }
    }
```

virtual void onExit ()
onExit 方法会在该 CCNode 释放之前调用。 如果使用了过渡场景，则 onExit 方法会在过渡效果结束以后调用 语法 CCNode::onExit();

virtual void　onEnterTransitionDidFinish ()

onEnterTransitionDidFinish 方法会在调用 onEnter 方法后调用。 如果使用了过渡场景，则 onEnterTransitionDidFinish 方法会在过渡效果结束以后调用。

例子代码

```
void CCLayer::onEnterTransitionDidFinish(){
        if (m_bIsAccelerometerEnabled){
            CCAccelerometer::sharedAccelerometer()->setDelegate(this);
        }
        CCNode::onEnterTransitionDidFinish();
}
```

virtual bool　ccTouchBegan (CCTouch *pTouch, CCEvent *pEvent)

当手指首次触摸到屏幕时调用的方法.

在 CCLayer, CCMenu, CCTargetTouchDelegate 类出现

参数 CCTouch *pTouch CCTouch 类型的指针，CCTouch 类主要是用来表示触摸信息的。 CCEvent *pEven CCEvent 类型的指针，CCEvent 类，表示触发该方法的事件。 触摸的响应对象 CCEvent *pEvent 触摸的响应事件

返回值 bool 类型，为 true，表示该事件已经被处理，则该次事件不会被再次监听。为 false，则该次事件仍就被继续监听。

例子代码

```
bool CCLayer::ccTouchBegan(CCTouch *pTouch, CCEvent *pEvent){
        CC_UNUSED_PARAM(pTouch);
        CC_UNUSED_PARAM(pEvent);
        CCAssert(false, "Layer#ccTouchBegan override me");
        return true;
}
```

virtual void　ccTouchesBegan (CCSet *pTouches, CCEvent *pEvent)

virtual void　ccTouchesMoved (CCSet *pTouches, CCEvent *pEvent)

virtual void　ccTouchesEnded (CCSet *pTouches, CCEvent *pEvent)

virtual void　ccTouchesCancelled (CCSet *pTouches, CCEvent *pEvent)

virtual void　didAccelerate (CCAcceleration *pAccelerationValue)

接收加速计事件时调用的方法

参数 CCAcceleration *pAccelerationValue typedef struct { double x; double y; double z; double timestamp; } CCAcceleration;

例子代码

```
class CC_DLL CCAccelerometerDelegate{
public:
        virtual void didAccelerate(CCAcceleration* pAccelerationValue{
        CC_UNUSED_PARAM(pAccelerationValue);
        }
}
```

virtual void registerWithTouchDispatcher (void)

如果 layer 接收触摸事件，则这个方法在 onEnter 时调用。 这个方法作用是注册触摸事件，并设置事件响应的优先级。

例子代码

```
void CCLayer::registerWithTouchDispatcher(){
        CCTouchDispatcher::sharedDispatcher()->addStandardDelegate(this,0);
}
```

virtual void touchDelegateRetain ()

这个方法名在源文件里未找到， 我认为这个方法应该是 virtual void keep(void) 使 layer 的计数加 1.

virtual void touchDelegateRelease ()

说明：这个方法名在源文件里未找到， 我认为这个方法应该是 virtual void destroy(void) 使 layer 的计数减 1 语法 void CCLayer::destroy(void) { this->release(); }

virtual bool getIsTouchEnabled (void)

获取能否接收触摸事件的属性。例子代码

```
bool isTouchEnable =  layer->getIsTouchEnabled();
```

返回值 返回一个布尔值，表示是否可以接收触摸事件。

virtual void setIsTouchEnabled (bool var)

设置能否接收触摸事件的属性。

virtual bool getIsAccelerometerEnabled (void)

获取能否接收加速计事件的属性。

virtual void setIsAccelerometerEnabled (bool var)

设置能否接收加速计事件的属性。

virtual bool getIsKeypadEnabled (void)

获取能否接收键盘事件的属性。

virtual void setIsKeypadEnabled (bool var)

设置能否接收键盘事件的属性。

static CCLayer * node (void)

分配并初始化一个对象,对象是自动释放的返回值 返回一个 CCScene 对象

例子 CCScene* HelloWorld::scene(){

```
CCScene *scene = CCScene::node();
HelloWorld *layer = HelloWorld::node();
scene->addChild(layer);
return scene;
}
```

bool m_bIsTouchEnabled

bool m_bIsAccelerometerEnabled

bool m_bIsKeypadEnabled

CCLayerGradient 类

CCLayerGradient 是 CCLayerColor 的子类，用来创建一个渐变的背景。CCLayerColor 所有特征是有效的,再加上以下新特点: - direction - final color 是根据成员变量 m_AlongVector 这个 CCPoint 的 x 为起点到 y 为终点的连线，颜色 m_startColor 到 m_endColor 。 m_AlongVector 默认是（0， 1）- 从上到下渐变。 如果“compressedInterpolation”是 YES（默认）你会看到开始和结束的颜色。

virtual bool initWithColor (const ccColor4B &start, const ccColor4B &end)

初始化一个渐变的Layer，需要传入开始颜色和结束颜色

例子代码

CCScene* HelloWorld::scene(){

```
    CCScene *scene = CCScene::node();
        //   'cene '和  'layer'  都是autorelease对象
       CCLayerGradient* layer = CCLayerGradient::node();
       layer->initWithColor(ccc4(255, 255, 0, 255), ccc4(255, 0, 255, 255));
          // 添加渐变Layer到scene
          scene->addChild(layer)
          return scene;
}
```

virtual bool initWithColor (const ccColor4B &start, const ccColor4B &end, const CCPoint &v)

初始化一个渐变的Layer，需要传入开始颜色、结束颜色、向量

渐变色的向量（起始颜色从设置的ccp开始，方向是设置的ccp和0,0点的连线）

依上例，layer->initWithColor(ccc4(255, 255, 255, 255), ccc4(255, 0, 255, 255), ccp(2, 2));

virtual const ccColor3B & getStartColor (void)

获取当前CCLayerGradient对象的变量m_tStartColor（开始颜色）

virtual void setStartColor (const ccColor3B &var)

设置渐变Layer的属性m_startColor

virtual const ccColor3B & getEndColor (void)

返回渐变Layer的属性 m_endColor

virtual void setEndColor (const ccColor3B &var)

设置渐变Layer的属性 m_endColor

virtual GLubyte getStartOpacity (void)

返回渐变Layer的属性m_cStartOpacity 返回一个 GLubyte 对象，0-255之间

virtual void setStartOpacity (GLubyte var)

设置渐变Layer的属性m_cStartOpacity

语法 CCLayerGradient* layer = CCLayerGradient::node(); layer->initWithColor (ccc4(255, 0, 255, 255), ccc4(255, 0, 0, 255)); layer->setStartOpacity(88);

virtual GLubyte getEndOpacity (void)
virtual void setEndOpacity (GLubyte var)
设置、返回 渐变Layer的属性 m_cEndOpacity

virtual const CCPoint & getVector (void)
virtual void setVector (const CCPoint &var)
设置、返回Vector属性

virtual bool getIsCompressedInterpolation (void)
virtual void setIsCompressedInterpolation (bool var)
获取、设置渐变layer的m_bCompressedInterpolation变量值

static CCLayerGradient * layerWithColor (const ccColor4B &start, const ccColor4B &end)
初始化一个渐变的Layer，需要传入开始颜色和衰退颜色。

static CCLayerGradient * layerWithColor (const ccColor4B &start, const ccColor4B &end, const CCPoint &v)
初始化一个渐变的Layer，需要传入开始颜色、衰退颜色、向量。

static CCLayerGradient * node ()
开辟一个CCLayerGradient的空间并返回

virtual void updateColor ()
ccColor3B m_startColor
ccColor3B m_endColor
GLubyte m_cStartOpacity
GLubyte m_cEndOpacity
CCPoint m_AlongVector
bool m_bCompressedInterpolation

CCLayerMultiplex 类

CCMultipleLayer 拥有多个 CCLayer 的功能： 1、它支持一个或更多的 CCLayer 2、同一时间只有一个 CCLayer 可以被激活。

CCLayerMultiplex ()

CCLayerMultiplex 的构造函数

例子代码CCScene* HelloWorld::scene(){

CCScene *scene = CCScene::node();

CCLayerMultiplex* layerMultiplex = new CCLayerMultiplex();

scene->addChild(layerMultiplex);

return scene;

}

virtual ~CCLayerMultiplex ()

void addLayer (CCLayer *layer)

添加一个CCLayer对象

bool initWithLayer (CCLayer *layer)

bool initWithLayers (CCLayer *layer, va_list params)

外部需要调用CCLayerMultiplex * CCLayerMultiplex::layerWithLayers(CCLayer * layer, ...)

参数：要添加的layer 、va_list型变量

void switchTo (unsigned int n)

切换CCLayerMultiplex里面的CCLayer对象，从0开始算。

void switchToAndReleaseMe (unsigned int n)

切换到CCLayerMultiplex中的第N个CCLayer，并且这个CCLayer自身减一。

参数：第几个CCLayer

static CCLayerMultiplex * layerWithLayers (CCLayer *layer,...)

为CCLayerMultiplex添加多个Layer

static CCLayerMultiplex * layerWithLayer (CCLayer *layer)

初始化并且为CCLayerMultiplex添加一个CCLayer

static CCLayerMultiplex * node ()

初始化一个CCLayerMultiplex

unsigned int m_nEnabledLayer

CCMutableArray< CCLayer * > * m_pLayers