

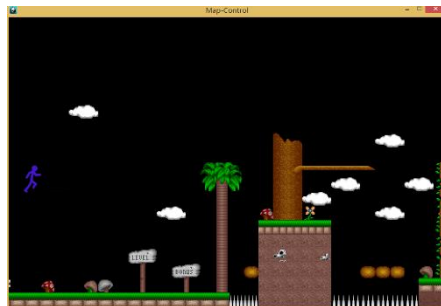
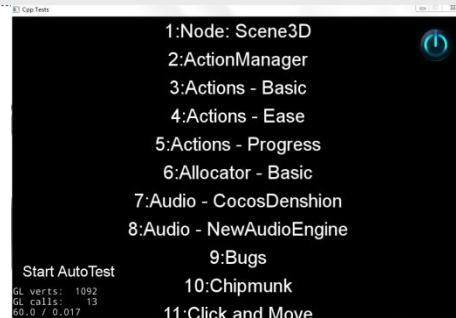
第四章 游戏引擎概览

第五章 二维游戏场景绘制

上节回顾

• 三个实例

- 官方测试项目、调试界面
- 命名空间、入口函数
- AppDelegate
- HelloWorldScene、初始化流程
- Menu、MenuItem
- Label
- TMXTiledMap
- Animation、Animate



本节目录

- Chapter 4 & Chapter 5
 - Cocos2d-x的内存管理机制
 - Cocos2d-x的交互响应机制
 - Cocos2d-x的UI组件

Cocos2d-x的内存管理机制

- 创建第一个场景

```
96 // create a scene. it's an autorelease object
97 auto scene = HelloWorld::createScene();
```



autorelease
自动释放

AutoreleasePool
自动释放池

引用计数内存管理机制

```
17 static HelloWorld* create()
18 {
19     HelloWorld *pRet = new HelloWorld();
20     if (pRet && pRet->init())
21     {
22         pRet->autorelease();
23         return pRet;
24     }
25     else
26     {
27         delete pRet;
28         pRet = NULL;
29         return NULL;
30     }
31 }
32 };
```

Cocos2d-x的内存管理机制

C++内存管理

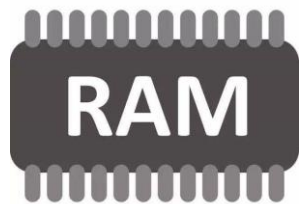
malloc/free 函数

new/delete 操作符

成对出现



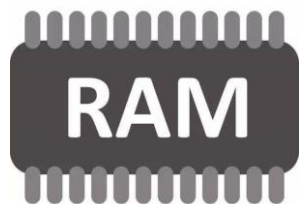
new 操作符用于获得一块内存空间，并返回指向内存的指针；delete 操作符用于释放内存空间。



内存的 5 个区

- 栈：在函数内局部变量都可以在栈上创建，函数执行结束时，局部变量自动释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

Cocos2d-x的内存管理机制



内存的 5 个区

- 堆：那些由 `new` 分配的内存块，它们的释放由程序员自己控制，所以一个 `new` 对应一个 `delete`。如果程序员没有释放内存块，那么在程序结束后，操作系统会自动回收。
- 自由存储区：由 `malloc` 等分配的内存块，和堆十分相似，不过由 `free` 释放。
- 全局/静态存储区：全局变量和静态变量被分配到同一块内存中。
- 常量存储区：这是一块比较特殊的存储区，里面存放的是常量，不允许修改。

Cocos2d-x的内存管理机制

Cocos2d 使用的是引用计数内存的管理机制，Cocos2d-x 是根据 Cocos2d 演化来的。我们在使用 Cocos2d-x 时，应该使用引用计数内存管理（虽然部分 Cocos2d-x 类还能使用 C++ 的内存管理方法，但是很多 Cocos2d-x 类如 Sprite 的构造函数已经声明为 protected，我们已经不能在其他类中使用“new”来创建对象了）。

Cocos2d-x 采用引用计数内存管理机制。引用计数内存管理机制简而言之，就是在对象的内部添加一个计数器，当外部引用增加时，引用计数加 1，当外部引用消失时，引用计数减 1，然后引擎通过这个计数器判断是否需要这个对象，当计数为 0 时，引擎会删除这个对象✕

Cocos2d-x 的类继承自 Ref，Ref 提供了引用计数的功能

Cocos2d-x的内存管理机制

- 手动修改引用计数

- new 创建对象，引用计数 加1
- 调用 retain() 方法，引用计数 加1
- 调用 release() 方法，引用计数 减1
- 若引用计数为0，回收对象内存

retain 和 release 一般成对出现，否则容易造成内存泄露！

Cocos2d-x的内存管理机制

在 Cocos2d-x 中应该如何使用引用计数管理内存呢？

```
void HelloWorld::memoryDemo()  
{  
    Sprite *sprite = Sprite::create("HelloWorld.png");  
    log("sprite retainCount = %i", sprite->getReferenceCount());  
    sprite->setPosition(Point(240, 160));  
    this->addChild(sprite);  
    log("sprite retainCount = %i", sprite->getReferenceCount());  
}
```



sprite retainCount = 1
sprite retainCount = 2

在该段代码中首先使用 `create` 创建了一个对象。在 `create` 方法中调用了 `new` 方法，并且将对象放到了自动释放池中，但是这时它还没有被释放，所以其应用计数为 1。然后将这个对象添加到 `HelloWorld` 层上，其应用计数加 1，所以应用计数为 2。**父视图**在销毁时会对**子视图**的应用计数减 1，**自动释放池**会对应用计数减 1，所以最终对象会被销毁。

Cocos2d-x的内存管理机制

日志输出

有时，在你的游戏正在运行的时候，为了了解程序的运行过程或是为了查找一个 BUG，你想看到一些运行时信息，可以! 这个需求引擎已经考虑到了，使用 `log()` 可以把信息输出到控制台，这样使用：

```
// a simple string
```

```
log("This would be outputted to the console");
```

```
// a string and a variable
```

```
string s = "My variable";
```

```
log("string is %s", s);
```

```
// a float and a variable
```

```
float f = 2.0f;
```

```
log("float is %f", f);
```

```
// a bool and a variable
```

```
bool b = true;
```

```
if (b == true)
```

```
    log("bool is true");
```

```
else
```

```
    log("bool is false");
```

Cocos2d-x的内存管理机制

节点关系

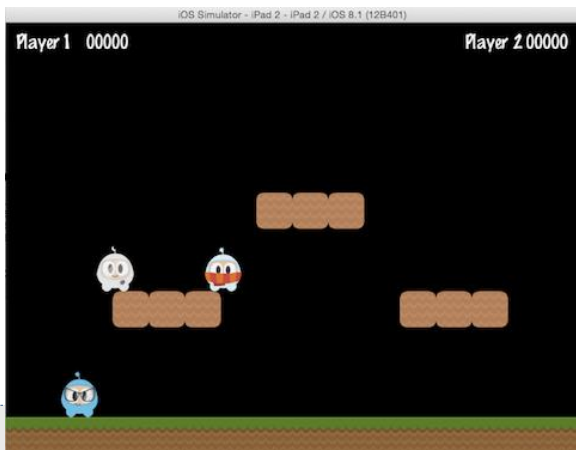
Cocos2d-x 的 **节点关系**，是被附属和附属的关系，就像数据结构中的父子关系，如果两个节点被添加到一个父子关系中，那么父节点的属性变化会被自动应用到子节点中。

这三个精灵被添加到了在一个父子关系中，

```
auto myNode = Node::create();
```

```
// rotating by setting  
myNode->setRotation(50);
```

```
// scaling by setting  
myNode->setScale(2.0);
```



Cocos2d-x的内存管理机制

自动释放池

在 Cocos2d-x 中还提供了自动释放池 `AutoreleasePool` 来管理内存。我们所有需要自动释放的对象都通过 `autorelease()` 方法将其放入自动释放池中，然后自动释放池 `AutoreleasePool` 帮我们进行内存管理。

将对象加入自动释放池

`autorelease()` 方法做了什么呢？在 `autorelease()` 方法中使用 `PoolManager::getInstance()→getCurrentPool()→addObject(this)`；将我们需要进行自动释放的对象添加到自动释放池中。`AutoreleasePool` 使用一个容器将我们所有需要管理的对象进行保存，引擎会在某个时间遍历整个容器，逐个调用对象的释放函数进行释放。

每帧结束时 清理

智能管理内存

在代码中如何使用自动释放池呢？



Cocos2d-x的内存管理机制



```
void HelloWorld::autoreleasePoolDemo()  
{  
    __String *str = new __String("HelloWorld");  
    str->autorelease();  
    log("referenceCount=%i", str->getReferenceCount());  
    __Array *array = __Array::create();  
    array->addObject(str);  
    log("referenceCount=%i", str->getReferenceCount());  
}
```



referenceCount=1
referenceCount=2

为什么会出现这样的结果呢，我们一起来分析一下。创建对象时对象的引用计数为 1，然后将对象放在自动释放池中，自动释放池会将对象的引用计数减 1，但是这个减 1 的执行时间是不确定的。然后我们对对象使用 `addObject`，使对象的引用计数加 1，但是这时自动释放池还没有对对象的引用计数减 1，所以这时的引用计数还是 2。

Cocos2d-x的内存管理机制

```
void HelloWorld::autoreleasePoolDemo()  
{  
    __String *str = new __String("HelloWorld");  
    str->autorelease();  
    log("referenceCount=%i", str->getReferenceCount());  
    __Array *array = __Array::create();  
    array->addObject(str);  
    log("referenceCount=%i", str->getReferenceCount());  
}
```



除了将视图添加到父视图中(`addChild`)、将视图从父视图中移除(`removeFromParent`)，还有将对象加入数组(`addObject`)、将对象从数组中移除(`removeObject`)、将对象加入到字典(`setObject`)和将对象从字典中移除(`removeObjectForKey`)。这两组方法可以改变对象的引用计数，并且都会对添加进来的对象进行内存管理，不用开发者自己管理。

Cocos2d-x的内存管理机制

对于引擎中的类的对象，例如 Scene、Sprite、Layer 等，引擎为我们提供了 create 方法，在这个方法中已经将对象放入自动释放池中了，我们不再需要将它加入自动释放池中。

Simple

```
void HelloWorld::autoReleasePoolDemo()
{
    //__String *str = new __String("HelloWorld");
    //str->autorelease();
    __String *str = __String::create("HelloWorld");
    log("referenceCount=%i", str->getReferenceCount());
    __Array *array = __Array::create();
    array->addObject(str);
    log("referenceCount=%i", str->getReferenceCount());
}
```

referenceCount=1
referenceCount=2

```
1 #define CREATE_FUNC(__TYPE__) \
2 static __TYPE__* create() \
3 { \
4     __TYPE__ *pRet = new __TYPE__(); \
5     if (pRet && pRet->init()) \
6     { \
7         pRet->autorelease(); \
8         return pRet; \
9     } \
```


Cocos2d-x的内存管理机制

转场前

本节目录

- Chapter 4 & Chapter 5
 - Cocos2d-x的内存管理机制
 - Cocos2d-x的交互响应机制
 - Cocos2d-x的UI组件

Cocos2d-x的交互响应机制

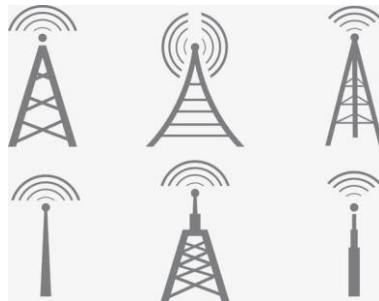
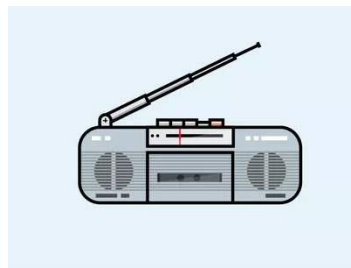
• 事件分发机制

Cocos2d-x 通过事件分发机制响应用户事件，已内置支持常见的事件如触摸事件，键盘事件等。同时提供了创建自定义事件的方法，满足我们在游戏的开发过程中，特殊的事件响应需求。

• 基本元素

- 事件监听器：负责接收事件，并执行预定义的事件处理函数
- 事件分发器：负责发起通知
- 事件对象：记录事件的相关信息

- EventListener
- EventDispatcher
- Eg: Touch



Cocos2d-x的交互响应机制



• 事件分发机制

对于事件分发, cocos2d官方定义为: 当事件发生(例如, 用户触摸屏幕, 或者敲键盘), EventDispatcher 会发布(Event objects)事件对象到合适的EventListeners, 并调用你的回调。各个Event object包含事件的信息(比如, 触摸点所在的坐标)。

事件处理代码

我的理解是对于一个事件(比如触摸事件、键盘相应事件等)可以与任意对象绑定, 那么当这个事件被用户触发时, 此时应该执行哪一个对象的回调函数(比如此时我们在好几个sprite上同时绑定了touch事件, 用户此时点击屏幕, 用户此时想点击的到底是哪一个sprite), 我们就需要作出判断。

STEP

1. 通过点击位置进行点击范围判断, 来确定执行哪一个sprite的事件监听器
2. 如果该位置存在重叠的sprite绑定了相同的事件, 则依据优先级(SceneGraphPriority显示优先级或FixedPriority固定优先级)来顺序执行函数回调
3. 通过设置swallowTouches属性为true, 并在onTouchBegan中返回true或者false来决定是否阻止事件的顺序传递。如果onTouchBegan返回true, 且swallowTouches为true, 则事件被吞没事件的顺序传递则被阻止。

Cocos2d-x的交互响应机制

- 监听器

事件的吞没

当你有一个监听器，已经接收到了期望的事件，这时事件应该被吞没。事件被吞没，意味着在事件的传递过程中，你消耗了此事件，**事件不再向下传递**。避免了下游的其它监听器获取到此事件。

设置吞没：

```
// When "swallow touches" is true, then returning 'true' from the  
// onTouchBegan method will "swallow" the touch event, preventing  
// other listeners from using it.
```

```
listener1->setSwallowTouches(true);
```

```
// you should also return true in onTouchBegan()
```

```
listener1->onTouchBegan = [](Touch* touch, Event* event){
```

```
    // your code
```

```
    return true;
```

```
};
```



Cocos2d-x的交互响应机制

• 优先级

事件的吞没中，我们提到了事件的传递。事件如何传递，先到哪个监听器？这是由优先级决定的。

- ✓ **固定值优先级** 使用一个整形的数值，数值较低的监听器比数值较高的监听器，先接收到事件。
- ✓ **场景图优先级** 是指向节点对象的指针，`z-order` 较高的节点中的监听器比 `z-order` 较低的节点中的，先接收到事件。由于 `z-order` 较高的节点在顶部绘制，所以使用这种优先级可以确保触摸事件被正确响应
 - ✓ **`addEventListenerWithFixedPriority`** 有两个参数：
参数1: Listener 监听器
参数2: `fixedPriority` 固定优先级，其中0是系统占有，不能设置为0
 - ✓ **`addEventListenerWithSceneGraphPriority`** 有两个参数：
参数1: Listener 监听器
参数2: `node` 根据`node`来确定监听器的优先级

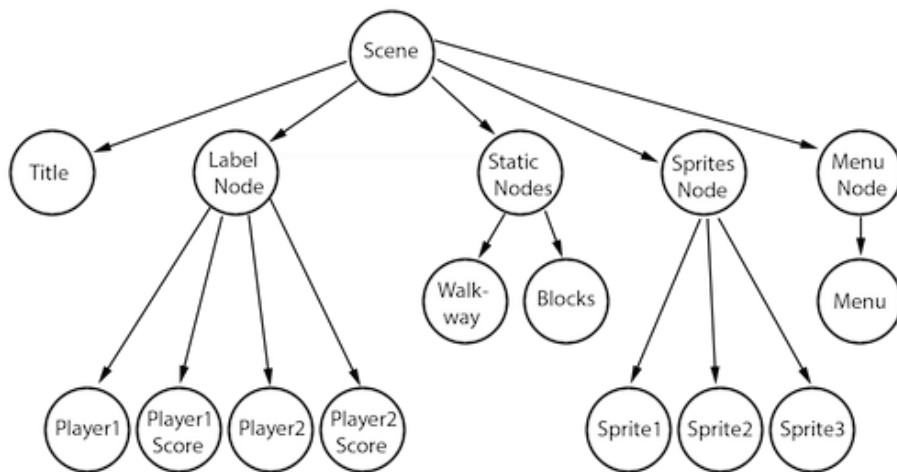
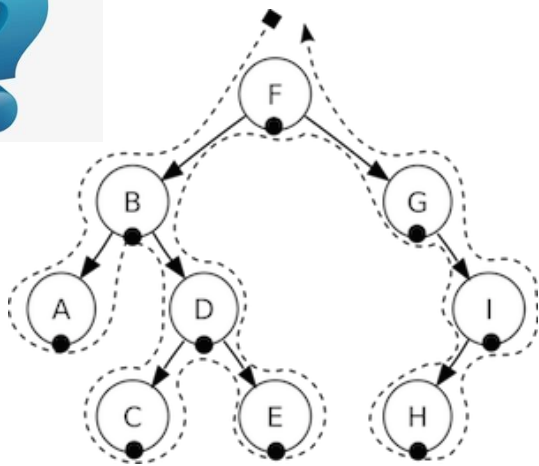
两种方法的区别就是一个是自己手动指定Listener的优先级，一个是根据`node`来决定。

Cocos2d-x的交互响应机制

事件分发器

单例模式

```
Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);
```



当使用 场景图优先级时，事件是按照绘制的反方向，即 I, H, G, F, E, D, C, B, A 传递。如果一个事件被触发，I 节点先接收到，如果在 I 节点中事件被吞没，则不会继续传递，未被吞没，事件将传递到 H 节点，每个节点都重复同样的逻辑，直到事件被吞没，或者传递结束，本次事件触发才完成。

Cocos2d-x的交互响应机制

• 监听器

- `EventListenerTouch` - 响应触摸事件

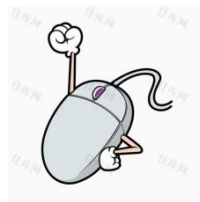


- `EventListenerKeyboard` - 响应键盘事件



- `EventListenerAcceleration` - 响应加速度事件

- `EventListenerMouse` - 响应鼠标事件



- `EventListenerCustom` - 响应自定义事件

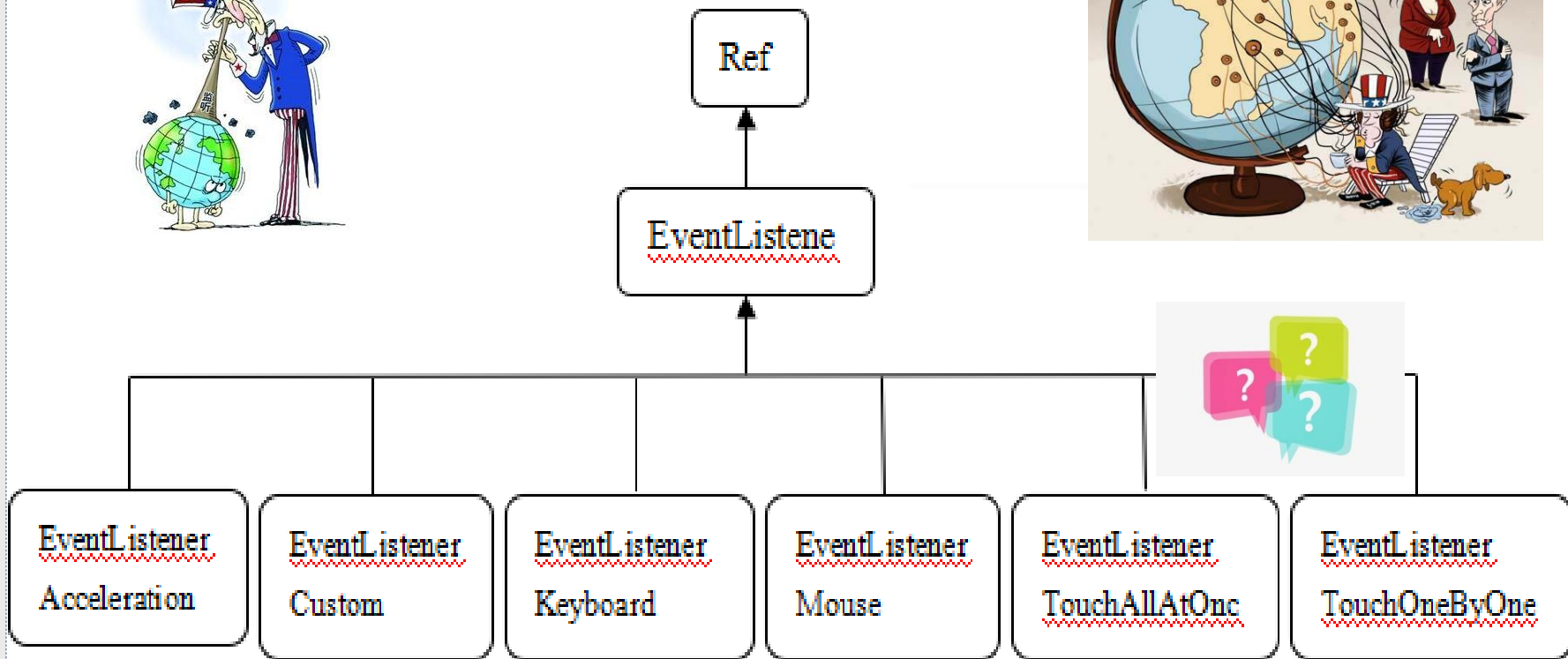


监听器与设备有关还是无关?



Cocos2d-x的交互响应机制

- 监听器类



Cocos2d-x的交互响应机制

• 触摸事件

触摸事件是手机游戏中最重要的事件，它易于创建，还能提供多种多样的功能。

让我们先了解一下什么是触摸事件，当你触摸移动设备的屏幕时，设备感受到被触摸，了解到被触摸的位置，同时取得触摸到的内容，然后你的触摸被回答。这就是触摸事件。

创建触摸事件监听器：

```
// Create a "one by one" touch event listener
// (processes one touch at a time)
auto listener1 = EventListenerTouchOneByOne::create();
// trigger when you push down
listener1->onTouchBegan = [](Touch* touch, Event* event){
    // your code
    return true; // if you are consuming it
};
```

Cocos2d-x的交互响应机制

- 触摸事件

```
// Create a "one by one" touch event listener  
// (processes one touch at a time)  
auto listener1 = EventListenerTouchOneByOne::create();
```

```
// trigger when moving touch  
listener1->onTouchMoved = [](Touch* touch, Event* event){  
    // your code  
};
```

2

```
// trigger when you let up  
listener1->onTouchEnded = [=](Touch* touch, Event* event){  
    // your code  
};
```

3

Cocos2d-x的交互响应机制

• 触摸事件

- 1**

```
// Create a "one by one" touch event listener
// (processes one touch at a time)
auto listener1 = EventListenerTouchOneByOne::create();
```
- 2**

```
// trigger when you push down
listener1->onTouchBegan = [](Touch* touch, Event* event){
    // your code
    return true; // if you are consuming it
};
```
- 3**

```
// Add listener
_eventDispatcher->addEventListenerWithSceneGraphPriority(listener1, this);
```

Cocos2d-x的交互响应机制

• 英雄快跑实例

```
Scene.h  Config.h  MapScene.cpp  X
ame      -> MapScene      update(float t)

206
207 void MapScene::addTouchListener()
208 {
209     // 加入触摸的处理
210     auto listener = EventListenerTouchOneByOne::create();
211     listener->onTouchBegan = CC_CALLBACK_2(MapScene::onTouchBegan, this);
212     Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);
213 }
```

```
Scene.h  Config.h  MapScene.cpp  X
ame      -> MapScene

221 }
222
223 bool MapScene::onTouchBegan(Touch *touch, Event *unused_event)
224 {
225     if (!this->m_isJump) {
226         m_isJump = true;
227         m_jumpDir = Dir::UP;
228     }
229     return true;
230 }
```



Cocos2d-x的交互响应机制

- 键盘事件

对于桌面游戏，一般需要通过键盘做一些游戏内的控制，这时你就需要监听键盘事件。

```
// creating a keyboard event listener
auto listener = EventListenerKeyboard::create();

listener->onKeyPressed = CC_CALLBACK_2(KeyboardTest::onKeyPressed, this);
listener->onKeyReleased = CC_CALLBACK_2(KeyboardTest::onKeyReleased, this);

_eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);
```

```
// Implementation of the keyboard event callback function prototype
void KeyboardTest::onKeyPressed(EventKeyboard::KeyCode keyCode, Event* event)
{
    log("Key with keycode %d pressed", keyCode);
}
```

```
void KeyboardTest::onKeyReleased(EventKeyboard::KeyCode keyCode, Event* event)
```



Cocos2d-x的交互响应机制

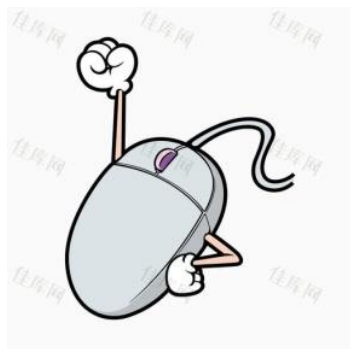
• 鼠标事件

```
auto mouseListener = EventListenerMouse::create();  
mouseListener->onMouseMove = CC_CALLBACK_1(HelloWorld::onMouseMove, this);  
mouseListener->onMouseUp = CC_CALLBACK_1(HelloWorld::onMouseUp, this);  
mouseListener->onMouseDown = CC_CALLBACK_1(HelloWorld::onMouseDown, this);  
mouseListener->onMouseScroll = CC_CALLBACK_1(HelloWorld::onMouseScroll, this);
```

```
Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(mouseListener, this);
```

```
void HelloWorld::onMouseDown(Event *event)  
{  
    log("Mouse Down detected.");  
}
```

```
void HelloWorld::onMouseMove(Event *event)  
{  
    // to illustrate the event....  
    EventMouse* e = (EventMouse*)event;  
    float x = e->getCursorX();  
    float y = e->getCursorY();  
    log("MousePosition : %f, %f", x, y);  
}
```



Cocos2d-x的交互响应机制

• 加速度传感器事件

现在一些移动设备配备有加速度传感器，我们可以通过监听它的事件获取各方向的加速度。

可以设想要完成一个游戏情景：通过来回移动手机，平衡小球在手机中的位置。这种场景的完成，就需要监听加速度传感器事件。

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

// creating an accelerometer event

1 auto listener = EventListenerAcceleration::create(CC_CALLBACK_2(
  AccelerometerTest::onAcceleration, this));

2 _eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);

// Implementation of the accelerometer callback function prototype

3 void AccelerometerTest::onAcceleration(Acceleration* acc, Event* event)
{
    // Processing logic here
}
```



Cocos2d-x的交互响应机制

• 注册/移除事件监听

当我们需求多个节点对象有相同的事件响应时，可以创建一个事件监听器，然后通过 `eventDispatcher`，将其注册到多个对象。



需要注意的是，在添加到多个对象时，需要使用 `clone()` 方法。

```
// Add listener
```

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

```
// Add the same listener to multiple objects.
```

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



```
_eventDispatcher->removeEventListener(listener);
```

本节目录

- Chapter 4 & Chapter 5
 - Cocos2d-x的内存管理机制
 - Cocos2d-x的交互响应机制
 - Cocos2d-x的UI组件

Cocos2d-x的UI组件



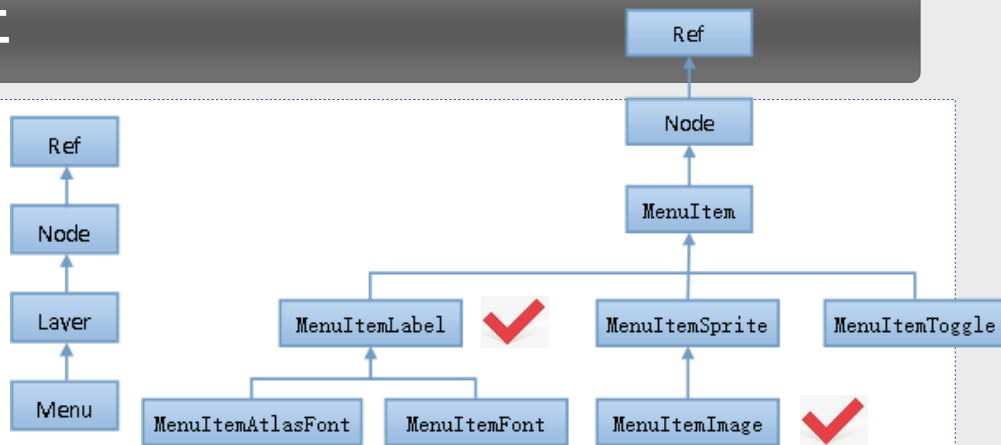
UI 组件不是游戏专用的，是个应用程序都可能会用几个。看一看你常使用的应用程序，肯定能发现它有使用 UI 组件。UI 代表什么，UI 组件是做什么的？

UI 代表用户界面，是 **User Interface** 的缩写，你看到的屏幕上的东西就是用户界面。界面组件有标签，按钮，菜单，滑动条等。Cocos2d-x 提供了一套易用的 UI 组件，游戏开发过程中，你能很容易的把它们添加到游戏中。

听起来这可能很简单，但创建像 标签 (Label) 这样的核心类实际要考虑很多问题。可以想象创建一套自定义的组件是多么的困难！当然你根本没必要这样做，因为你需要的我们都考虑到了。

Cocos2d-x的UI组件

- 标签 (Label)
- 菜单 (Menu)
- 菜单项 (MenuItem)



菜单是什么，我们肯定都很熟悉了，在每个游戏中都会有菜单。我们使用菜单浏览游戏选项，更改游戏设置。菜单通常包含开始，退出，设置，关于等项，菜单当然也可以包含子菜单。在 Cocos2d-x 提供 `Menu` 对象支持菜单功能，`Menu` 对象是一种特殊的 `Node` 对象。

```
auto menu = Menu::create();
menu->setPosition(Vec2::ZERO);
addChild(menu);
```

像我们刚才提到的一个菜单，总会有一些菜单项，比如开始，退出，设置等，没有菜单项的菜单没有存在的意义。Cocos2d-x 提供了一些方法来创建菜单项，比如使用 `Label` 对象，或是使用一张图像。菜单项一般有正常状态和选择状态。菜单项显示时是正常状态，当你点击它时变为选择状态，同时点击菜单还会触发一个回调函数。

Cocos2d-x的UI组件

- MenuItemLabel

```
Label* labelMIL = Label::createWithSystemFont("MenuItemLabel", "bitfont",40);  
labelMIL->setColor(Color3B(0,250,0));
```

```
MenuItemLabel* testMIL = MenuItemLabel::create(labelMIL,  
                                                CC_CALLBACK_1>HelloWorld::testCallback,this));
```

```
testMIL->setPosition(Point(visibleSize.width/2,700));
```

```
menu->addChild(testMIL);
```



Cocos2d-x的UI组件

• 按钮 (Button)

按钮是什么，好像没有必要解释，我们都知道这东西是用来点击的，点击后使我们的游戏产生一些变化，比如更改了场景，触发了动作等等。按钮会拦截点击事件，事件触发时调用事先定义好的回调函数。按钮有一个正常状态，一个选择状态，还有一个不可点击状态，按钮的外观可以根据这三个状态而改变。Cocos2d-x 提供 **Button** 对象支持按钮功能，创建一个按钮并定义一个回调函数很简单，记得在操作的时候要有头文件包含：`#include "ui/CocosGUI.h"`。 `using namespace ui;`

```
auto button = Button::create("normal_image.png", "selected_image.png", "disabled_image.png");
button->setTitleText("Button Text");
button->addEventListener([&](Ref* sender, Widget::TouchEventType type){
    switch (type)
    {
        case ui::Widget::TouchEventType::BEGAN:
            break;

        case ui::Widget::TouchEventType::ENDED:
            std::cout << "Button 1 clicked" << std::endl;

    }
});
this->addChild(button);
```



Cocos2d-x的UI组件

• 复选框 (CheckBox)

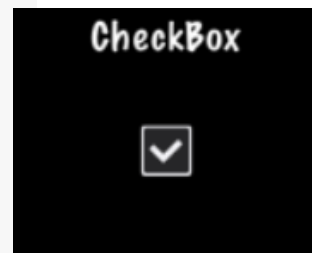
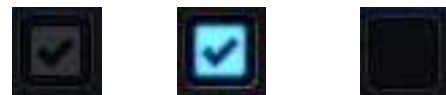
只有两种状态的项目经常被设计为复选框。Cocos2d-x 提供 **Checkbox** 对象支持复选框功能。

```
#include "ui/CocosGUI.h"    using namespace ui;
```

```
auto checkbox = CheckBox::create("check_box_normal.png",  
                                "check_box_normal_press.png",  
                                "check_box_active.png",  
                                "check_box_normal_disable.png",  
                                "check_box_active_disable.png");
```

```
checkbox->addEventListener([&](Ref* sender, Widget::TouchEvent type){  
    switch (type)  
    {  
        case ui::Widget::TouchEvent::BEGAN:  
            break;  
        case ui::Widget::TouchEvent::ENDED:  
            std::cout << "checkbox 1 clicked" << std::endl;
```

```
this->addChild(checkbox);
```



Cocos2d-x的UI组件

• 进度条 (LoadingBar)

如果你经常玩游戏，那肯定见过一个情景：屏幕上显示了一个进度条，提示资源正在加载中，这个条表示资源加载的进度。Cocos2d-x 提供 `LoadingBar` 对象支持进度条。

```
#include "ui/CocosGUI.h"    using namespace ui;

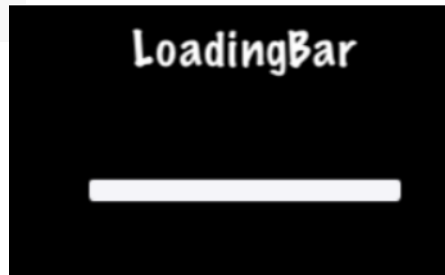
auto loadingBar = LoadingBar::create("LoadingBarFile.png");

// set the direction of the loading bars progress
loadingBar->setDirection(LoadingBar::Direction::RIGHT);

// something happened, change the percentage of the loading bar
loadingBar->setPercent(25);

// more things happened, change the percentage again.
loadingBar->setPercent(35);

this->addChild(loadingBar);
```



Cocos2d-x的UI组件

• 滑动条 (Slider)

有时候你想平滑的改变一个值，比如游戏设置中，调整背景音乐的音量，或着你有一个角色，允许用户设置攻击敌人的力量。这种场景最适合使用滑动条，Cocos2d-x 提供 `Slider` 对象支持滑动条。

```
#include "ui/CocosGUI.h"    using namespace ui;

auto slider = Slider::create();

slider->loadBarTexture("Slider_Back.png"); // what the slider looks like

slider->loadSlidBallTextures("SliderNode_Normal.png", "SliderNode_Press.png", "SliderNode_Disable.png");

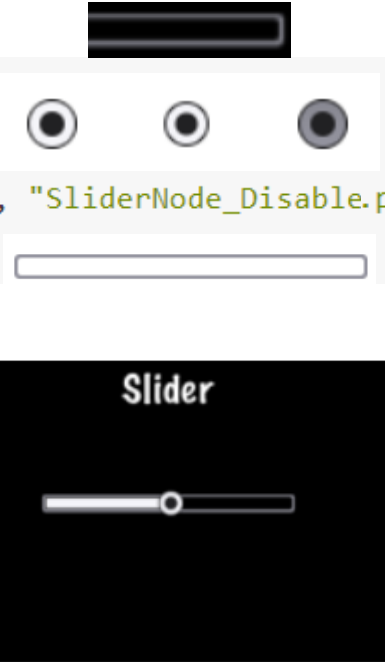
slider->loadProgressBarTexture("Slider_PressBar.png");

slider->addTouchEventListner([&](Ref* sender, Widget::TouchEvent type){
    switch (type)
    {
        case ui::Widget::TouchEvent::BEGAN:
            break;

        case ui::Widget::TouchEvent::ENDED:
            std::cout << "slider moved" << std::endl;

    }
});

this->addChild(slider);
```



The images on the right illustrate the visual components of a slider widget. From top to bottom: a solid black rectangular bar; three circular knobs with different states (normal, pressed, disabled); a white progress bar with a black outline; and a final assembled slider widget with the label "Slider" above it, showing a white bar and a circular knob.

Cocos2d-x的UI组件

• 文本框 (TextField)

如果你想让参与游戏的玩家可以自定义一个昵称怎么办，在哪里输入文本？Cocos2d-x 提供 `TextField` 满足这种需求。

```
#include "ui/CocosGUI.h" using namespace ui;

auto textField = TextField::create("", "Arial", 30);

// make this TextField password enabled
textField->setPasswordEnabled(true);

// set the maximum number of characters the user can enter for this TextField
textField->setMaxLength(10);

textField->addEventListener([&](Ref* sender, Widget::TouchEvent type){
    std::cout << "editing a TextField" << std::endl;
});

this->addChild(textField);
```



Cocos2d-x的UI组件

转场前

小结

- Cocos2d-x的内存管理机制
 - 引用计数
 - 自动释放池
- Cocos2d-x的交互响应机制
 - 事件分发机制
 - 监听器
 - 事件对象
- Cocos2d-x的UI组件
 - 菜单、按钮、复选框、进度条、滑动条、文本框