3.0物理引擎

cocos2d-x 3.0+ 中全新的封装的物理引擎给了开发者最大的便捷，你不用再繁琐与各种物理引擎的细节，完全的封装让开发者可以更快更好的将物理引擎的机制添加到自己的游戏中，简化的设计是从2.0到3.0+的一个质的飞跃。

**下面同样以一个小demo来展示一下物理引擎的运用，同时说一下我在运用物理引擎中遇到的一些小小的问题。**

**cocos2d-x 3.0+中的物理属性：**

**1、物理世界被集成到场景中，当你创建一个场景，你可以直接创建基于物理世界或者不使用物理世界的场景。**

**2、Node拥有它自己的body属性。（sprite也是node）‘**

3、cocos2d-x 3.0 已经封装了物理属性 Body(PhysicsBody),Shape(PhysicsShape),Contact(PhysicsContact),Joint(PhysicsJoint) 和World(PhysicsWorld),更加方便使用。

**4、方便的使用listener-EventListenerPhysicsContact进行碰撞检测。**

当然，封装好的物理引擎可以简化开发难度，如果有能力的话也可以直接使用Box2D和Chipmunk的原生的物理引擎进行开发，这样难度会有所提升。

**下面的代码创建一个带物理世界的场景，并传递到场景中的层上。**

**PhysicsLayer.h中**

class PhysicsLayer : public cocos2d::Layer { ... // add following codes设置层中的物理世界 void setPhyWorld(PhysicsWorld\* world){m\_world = world;} private: PhysicsWorld\* m\_world; ... }

PhysicsLayer.cpp中的createScene()方法中添加下面代码：

Scene\* PhysicsLayer::createScene() { ... // add following codes auto scene = Scene::createWithPhysics(); scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW\_ALL);//调试 auto layer = HelloWorld::create(); layer->setPhyWorld(scene->getPhysicsWorld());//将创建的物理世界传递到子层中 ... return scene; }

Scene类有一个新的static工厂方法createWithPhysics()创建一个带物理世界的场景。可以通过getPhysicsWorld()来获取PhysicsWorld的实例。

上述代码中注释为调试的代码在调试中非常有用，它会显示游戏中物体所带有的物理边界，便于观察碰撞中的细节等。

同时一个场景中只能有一个物理世界，所有属于这个场景的子层都共享这一个物理世界，所以在子层中用到物理世界时都会有这个定义的函数

void setPhyWorld(PhysicsWorld\* world){m\_world = world;}

进而来设置子层中的物理世界。

PhysicsWorld拥有默认的重力设置，Vector(0.0f,-98.0f),当然你也可以随意设置你想要的重力加速度，setGravity(Vect(0.0f,-200.0f)),设置重力加速度为20米每二次方秒。

**创建物理边界**

下面的代码创建一个物理边界

Size visibleSize = Director::getInstance()->getVisibleSize(); auto body = PhysicsBody::createEdgeBox(visibleSize, PHYSICSBODY\_MATERIAL\_DEFAULT, 3);//设置要添加到节点中的物理body auto edgeNode = Node::create(); edgeNode->setPosition(Point(visibleSize.width/2,visibleSize.height/2)); edgeNode->setPhysicsBody(body);//将物理body加入到创建的节点中 scene->addChild(edgeNode);场景中添加创建的物理节点

PhysicsWorld有很多工厂方法，如createEdgeBox创建一个矩形的边框，参数：

1、矩形区域，设置作为VisibleSize

2、可选参数，物理材料，默认为PHYSICSBODY\_MATERIAL\_DEFAULT。

3、可选参数，边框大小，默认为1.

**创建受重力影响的精灵**

下面的代码创建一个受重力影响的精灵，3.0中的创建精灵代码也大大简化

void HelloWorld::addNewSpriteAtPosition(Point p) { auto sprite = Sprite::create("circle.png");//创建精灵 sprite->setTag(1);//设置精灵的便签值 auto body = PhysicsBody::createCircle(sprite->getContentSize().width / 2);//创建一个附加在精灵身体上的圆形物理body sprite->setPhysicsBody(body);//将创建的body加到精灵身上 sprite->setPosition(p); this->addChild(sprite);//添加精灵 }

**下面讲一下真正的重点所在——物理碰撞检测**

谈到物理碰撞检测，真是感慨良多，在第一次运用碰撞检测时，遇到各种问题，最大的问题就是注册完物理碰撞响应时间，写好碰撞响应函数后，居然没反应，找各种论坛贴，各种google，最后都没有解决问题，最后还是在官网找到的解决方法，下面先介绍下如何设置碰撞检测。

前面的博客讲过了3.0中的事件分发机制，所以物理引擎的碰撞事件也不例外，统一的都有事件派发器来管理。

下面代码注册碰撞响应事件和回调函数

auto contactListener = EventListenerPhysicsContact::create(); contactListener->onContactBegin = CC\_CALLBACK\_1(HelloWorld::onContactBegin, this); \_eventDispatcher->addEventListenerWithSceneGraphPriority(contactListener, this);

每一次碰撞检测事件是有EventListenerPhysicsContact来进行监听的。监听到碰撞事件时，会回调响应事件 onContactBegin()来进行碰撞事件的处理。\_eventDispatcher是事件派发器，由它管理所有的注册事件。

EventListenerPhysicsContact是碰撞检测中的一种，也可以运用下面的来进行碰撞事件的注册 EventListenerPhysicsContactWithBodies,EventListenerPhysicsContactWithShapes,EventListenerPhysicsContactWithGroup 来进行你感兴趣的bodys,shape和group事件监听。

在上面说了这么多的东西，最重要的东西就是下面的，没有下面的东西，碰撞事件根本不起作用，这就是我第一次运用碰撞时遇到的问题。也就是设置物理接触相关的位掩码值，默认的接触事件不会被接受，需要设置一定的掩码值来使接触事件响应。

接触掩码值有三个值，分别是：

1、CategoryBitmask，默认值为0xFFFFFFFF

2、ContactTestBitmask,默认值为 0x00000000

3、CollisionBitmask，默认值为0xFFFFFFFF

这三个掩码值都有对应的set/get方法来设置和获取。

这三个掩码值由逻辑与来进行操作测试。

一个body的CategoryBitmask和另一个body的ContactTestBitmask的逻辑与的结果不等于0时，接触事件将被发出，否则不发送。

一个body的CategoryBitmask和另一个body的CollisionBitmask的逻辑与结果不等于0时，他们将碰撞，否则不碰撞

默认情况下的body属性会进行物理碰撞，但不会发送碰撞检测的信号，也就不会响应碰撞回调函数，这个可以看下默认情况下的掩码值的逻辑与

CategoryBitmask = 0xFFFFFFFF；

ContactTestBitmask = 0x00000000；

CategoryBitmask & ContactTestBitmask = 0,所以不会发送碰撞信号

CollisionBitmask = 0xFFFFFFFF；

CategoryBitmask & CollisionBitmask = 0xFFFFFFFF 所以物体会碰撞，但是不会响应碰撞回调函数。

上面介绍的掩码值是碰撞检测回调中最重要的，没有上面的掩码值，所有的碰撞回调函数都不会发生。

EventListenerPhysicsContact有四个接触回调函数：

1、onContactBegin,在接触开始时被调用，仅调用一次，通过放回true或者false来决定两个物体是否有碰撞。同时可以 使用PhysicsContact::setData()来设置接触操作的用户数据。当返回false时，onContactPreSolve和 onContactPostSolve将不会被调用，但是onContactSeperate将被调用一次。

2、onContactPreSlove ，会在每一次被调用，通过放回true或者false来决定两个物体是否有碰撞，同样可以用ignore()来跳过后续的onContactPreSolve和onContactPostSolve回调函数。(默认返回true)

3、onContactPostSolve，在两个物体碰撞反应中的每个步骤中被处理调用。可以在里面做一些后续的接触操作。如销毁body

4、onContactSeperate，在两个物体分开时被调用，在每次接触时只调用一次，和onContactBegin配对使用。

上述中最重要的就是碰撞检测事件的讲解，这是游戏中用到碰撞经常要用到的。

好了，这篇讲解了游戏中的物理碰撞机制。