**关于Scene与Camera的关系**

首先Camera继承与Node，

std::vector<Camera\*> \_cameras; //weak ref to Camera

Camera\* \_defaultCamera; //weak ref, default camera created by scene, \_cameras[0], **Caution that the default camera can not be added to \_cameras before onEnter is called**

查看Scene的构造函数，发现其创建了一个\_defaultCamera，同时将它加入了子节点：

Scene::Scene()

{

#if CC\_USE\_3D\_PHYSICS && CC\_ENABLE\_BULLET\_INTEGRATION

\_physics3DWorld = nullptr;

\_physics3dDebugCamera = nullptr;

#endif

#if CC\_USE\_NAVMESH

\_navMesh = nullptr;

\_navMeshDebugCamera = nullptr;

#endif

#if CC\_USE\_PHYSICS

\_physicsWorld = nullptr;

#endif

\_ignoreAnchorPointForPosition = true;

setAnchorPoint(Vec2(0.5f, 0.5f));

\_cameraOrderDirty = true;

//create default camera

**\_defaultCamera = Camera::create();**

**addChild(\_defaultCamera);**

\_event = Director::getInstance()->getEventDispatcher()->addCustomEventListener(Director::EVENT\_PROJECTION\_CHANGED, std::bind(&Scene::onProjectionChanged, this, std::placeholders::\_1));

\_event->retain();

Camera::\_visitingCamera = nullptr;

}

我们在切换Scene的时候，都会调用Scene子节点的onEnter和onExit函数，我们再查看Camera的这两个函数：

**void Camera::onEnter()**

{

if (\_scene == nullptr)

{

auto scene = getScene();

if (scene)

{

setScene(scene);

}

}

Node::onEnter();

}

**void Camera::onExit()**

{

// remove this camera from scene

setScene(nullptr);

Node::onExit();

}

**void Camera::setScene(Scene\* scene)**

{

if (\_scene != scene)

{

//remove old scene

if (\_scene)

{

auto& cameras = \_scene->\_cameras;

auto it = std::find(cameras.begin(), cameras.end(), this);

if (it != cameras.end())

cameras.erase(it);

\_scene = nullptr;

}

//set new scene

if (scene)

{

\_scene = scene;

auto& cameras = \_scene->\_cameras;

auto it = std::find(cameras.begin(), cameras.end(), this);

if (it == cameras.end())

{

\_scene->\_cameras.push\_back(this);

//notify scene that the camera order is dirty

\_scene->setCameraOrderDirty();

}

}

}

}

这就是为什么上面标红的说明，也就是camera其实一直存在于Scene的子节点中，但是在每次切换Scene的时候，都会重新添加或删除该Camera

**Scene的render**

void Scene::render(Renderer\* renderer)

{

auto director = Director::getInstance();

Camera\* defaultCamera = nullptr;

**const auto& transform = getNodeToParentTransform();**

for (const auto& camera : getCameras())

{

if (!camera->isVisible())

continue;

Camera::\_visitingCamera = camera;

if (Camera::\_visitingCamera->getCameraFlag() == CameraFlag::DEFAULT)

{

defaultCamera = Camera::\_visitingCamera;

}

director->pushMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION);

director->loadMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION, **Camera::\_visitingCamera->getViewProjectionMatrix());**

**camera->apply();**

//clear background with max depth

camera->clearBackground();

//visit the scene

**visit(renderer, transform, 0);**

#if CC\_USE\_NAVMESH

if (\_navMesh && \_navMeshDebugCamera == camera)

{

\_navMesh->debugDraw(renderer);

}

#endif

renderer->render();

director->popMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION);

}

#if CC\_USE\_3D\_PHYSICS && CC\_ENABLE\_BULLET\_INTEGRATION

if (\_physics3DWorld && \_physics3DWorld->isDebugDrawEnabled())

{

director->pushMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION);

director->loadMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION, \_physics3dDebugCamera != nullptr ? \_physics3dDebugCamera->getViewProjectionMatrix() : defaultCamera->getViewProjectionMatrix());

\_physics3DWorld->debugDraw(renderer);

renderer->render();

director->popMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_PROJECTION);

}

#endif

Camera::\_visitingCamera = nullptr;

experimental::FrameBuffer::applyDefaultFBO();

}

在上面的函数中首先调用当前Scene的模型视图变换矩阵，然后通过visit传递给所有的子节点，实际上是调用了Node的visit函数

**void Node::visit(Renderer\* renderer, const Mat4 &parentTransform, uint32\_t parentFlags)**

{

// quick return if not visible. children won't be drawn.

if (!\_visible)

{

return;

}

uint32\_t flags = processParentFlags(parentTransform, parentFlags);

// IMPORTANT:

// To ease the migration to v3.0, we still support the Mat4 stack,

// but it is deprecated and your code should not rely on it

\_director->pushMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_MODELVIEW);

\_director->loadMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_MODELVIEW, \_modelViewTransform);

bool visibleByCamera = isVisitableByVisitingCamera();

int i = 0;

if(!\_children.empty())

{

**sortAllChildren();**

// draw children zOrder < 0

for( ; i < \_children.size(); i++ ) **// 先绘制localZOrder<0**

{

auto node = \_children.at(i);

if (node && node->\_localZOrder < 0)

**node->visit(renderer, \_modelViewTransform, flags);**

else

break;

}

// self draw

if (visibleByCamera) **// 绘制自己本身**

**this->draw(renderer, \_modelViewTransform, flags);**

for(auto it=\_children.cbegin()+i; it != \_children.cend(); ++it) **// 绘制localZOrder>0**

(\*it)->visit(renderer, \_modelViewTransform, flags);

}

else if (visibleByCamera) **// 没有子节点**

{

**this->draw(renderer, \_modelViewTransform, flags);**

}

\_director->popMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_MODELVIEW);

// FIX ME: Why need to set \_orderOfArrival to 0??

// Please refer to https://github.com/cocos2d/cocos2d-x/pull/6920

// reset for next frame

// \_orderOfArrival = 0;

}