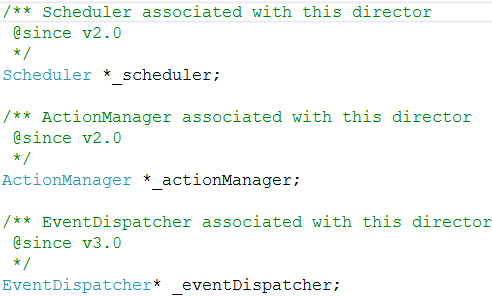
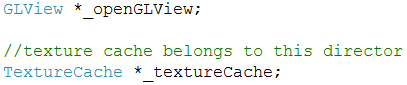
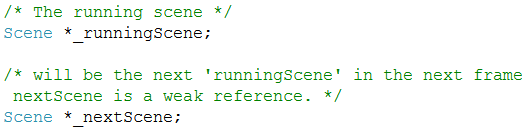
Director统筹管理整个游戏，成员变量主要包含：

















**Director是单例类，一旦获取之后即调用其init函数，实际上我们调用的单例Director是DisplayLinkDirector：**

Director\* Director::getInstance()

{

if (!s\_SharedDirector)

{

**s\_SharedDirector = new (std::nothrow) DisplayLinkDirector();**

CCASSERT(s\_SharedDirector, "FATAL: Not enough memory");

**s\_SharedDirector->init();**

}

return s\_SharedDirector;

}

初始化为：

bool Director::init(void)

{

**setDefaultValues();**

// scenes

**\_runningScene = nullptr;**

**\_nextScene = nullptr;**

\_notificationNode = nullptr;

**\_scenesStack.reserve(15);**

// FPS

\_accumDt = 0.0f;

\_frameRate = 0.0f;

\_FPSLabel = \_drawnBatchesLabel = \_drawnVerticesLabel = nullptr;

\_totalFrames = 0;

\_lastUpdate = new struct timeval;

\_secondsPerFrame = 1.0f;

// paused ?

\_paused = false;

// purge ?

\_purgeDirectorInNextLoop = false;

// restart ?

\_restartDirectorInNextLoop = false;

\_winSizeInPoints = Size::ZERO;

\_openGLView = nullptr;

\_defaultFBO = nullptr;

\_contentScaleFactor = 1.0f;

\_console = new (std::nothrow) Console;

// scheduler

**\_scheduler = new (std::nothrow) Scheduler();**

// action manager

**\_actionManager = new (std::nothrow) ActionManager();**

**\_scheduler->scheduleUpdate(\_actionManager, Scheduler::PRIORITY\_SYSTEM, false);**

**\_eventDispatcher = new (std::nothrow) EventDispatcher();**

**\_eventAfterDraw = new (std::nothrow) EventCustom(EVENT\_AFTER\_DRAW);**

\_eventAfterDraw->setUserData(this);

**\_eventAfterVisit = new (std::nothrow) EventCustom(EVENT\_AFTER\_VISIT);**

\_eventAfterVisit->setUserData(this);

**\_eventBeforeUpdate = new (std::nothrow) EventCustom(EVENT\_BEFORE\_UPDATE);**

\_eventBeforeUpdate->setUserData(this);

**\_eventAfterUpdate = new (std::nothrow) EventCustom(EVENT\_AFTER\_UPDATE);**

\_eventAfterUpdate->setUserData(this);

**\_eventProjectionChanged = new (std::nothrow) EventCustom(EVENT\_PROJECTION\_CHANGED);**

\_eventProjectionChanged->setUserData(this);

//init TextureCache

**initTextureCache();**

**initMatrixStack();**

**\_renderer = new (std::nothrow) Renderer;**

RenderState::initialize();

return true;

}

**通过配置文件设置纹理格式及游戏类型（3D、2D and so on，默认为3D）：**

void Director::setDefaultValues(void)

{

Configuration \*conf = Configuration::getInstance();

// default FPS

double fps = conf->getValue("cocos2d.x.fps", Value(kDefaultFPS)).asDouble();

\_oldAnimationInterval = \_animationInterval = 1.0 / fps;

// Display FPS

\_displayStats = conf->getValue("cocos2d.x.display\_fps", Value(false)).asBool();

// GL projection

std::string projection = conf->getValue("cocos2d.x.gl.projection", Value("3d")).asString();

if (projection == "3d")

\_projection = Projection::\_3D;

else if (projection == "2d")

\_projection = Projection::\_2D;

else if (projection == "custom")

\_projection = Projection::CUSTOM;

else

CCASSERT(false, "Invalid projection value");

// Default pixel format for PNG images with alpha

std::string pixel\_format = conf->getValue("cocos2d.x.texture.pixel\_format\_for\_png", Value("rgba8888")).asString();

if (pixel\_format == "rgba8888")

Texture2D::setDefaultAlphaPixelFormat(Texture2D::PixelFormat::RGBA8888);

else if(pixel\_format == "rgba4444")

Texture2D::setDefaultAlphaPixelFormat(Texture2D::PixelFormat::RGBA4444);

else if(pixel\_format == "rgba5551")

Texture2D::setDefaultAlphaPixelFormat(Texture2D::PixelFormat::RGB5A1);

// PVR v2 has alpha premultiplied ?

bool pvr\_alpha\_premultipled = conf->getValue("cocos2d.x.texture.pvrv2\_has\_alpha\_premultiplied", Value(false)).asBool();

Image::setPVRImagesHavePremultipliedAlpha(pvr\_alpha\_premultipled);

}

**设置纹理格式为RGBA4444后结果为：明显出现了很多条纹**

Texture2D::PixelFormat format = Texture2D::getDefaultAlphaPixelFormat();

Texture2D::setDefaultAlphaPixelFormat(Texture2D::PixelFormat::RGBA4444);

format = Texture2D::getDefaultAlphaPixelFormat();



**初始化矩阵栈（模型视图矩阵栈、投影矩阵栈、纹理矩阵栈）**

void Director::initMatrixStack()

{

while (!\_modelViewMatrixStack.empty())

{

\_modelViewMatrixStack.pop();

}

while (!\_projectionMatrixStack.empty())

{

\_projectionMatrixStack.pop();

}

while (!\_textureMatrixStack.empty())

{

\_textureMatrixStack.pop();

}

\_modelViewMatrixStack.push(Mat4::IDENTITY);

\_projectionMatrixStack.push(Mat4::IDENTITY);

\_textureMatrixStack.push(Mat4::IDENTITY);

}

**初始化纹理缓存：**

void Director::initTextureCache()

{

\_textureCache = new (std::nothrow) TextureCache();

}

**在Win32平台上Application的run方法实现为：**

int Application::run()

{

PVRFrameEnableControlWindow(false);

// Main message loop:

LARGE\_INTEGER nLast;

LARGE\_INTEGER nNow;

QueryPerformanceCounter(&nLast);

initGLContextAttrs();

// Initialize instance and cocos2d.

if (**!applicationDidFinishLaunching()**)

{

return 1;

}

auto director = Director::getInstance();

auto glview = director->getOpenGLView();

// Retain glview to avoid glview being released in the while loop

glview->retain();

while(!glview->windowShouldClose())

{

QueryPerformanceCounter(&nNow);

if (nNow.QuadPart - nLast.QuadPart > \_animationInterval.QuadPart)

{

nLast.QuadPart = nNow.QuadPart - (nNow.QuadPart % \_animationInterval.QuadPart);

**director->mainLoop();**

**glview->pollEvents();**

}

else

{

Sleep(1);

}

}

// Director should still do a cleanup if the window was closed manually.

if (glview->isOpenGLReady())

{

director->end();

director->mainLoop();

director = nullptr;

}

glview->release();

return 0;

}

**Director的mainLoop函数：**

void DisplayLinkDirector::mainLoop()

{

if (\_purgeDirectorInNextLoop)

{

\_purgeDirectorInNextLoop = false;

purgeDirector();

}

else if (\_restartDirectorInNextLoop)

{

\_restartDirectorInNextLoop = false;

restartDirector();

}

else if (! \_invalid)

{

**drawScene();**

// release the objects

PoolManager::getInstance()->getCurrentPool()->clear();

}

}

void Director::drawScene()

{

// calculate "global" dt

calculateDeltaTime();

if (\_openGLView)

{

\_openGLView->pollEvents();

}

//tick before glClear: issue #533

if (! \_paused)

{

\_eventDispatcher->dispatchEvent(\_eventBeforeUpdate);

\_scheduler->update(\_deltaTime);

\_eventDispatcher->dispatchEvent(\_eventAfterUpdate);

}

\_renderer->clear();

experimental::FrameBuffer::clearAllFBOs();

/\* to avoid flickr, nextScene MUST be here: after tick and before draw.

\* FIXME: Which bug is this one. It seems that it can't be reproduced with v0.9

\*/

if (\_nextScene)

{

setNextScene();

}

pushMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_MODELVIEW);

if (\_runningScene)

{

#if (CC\_USE\_PHYSICS || (CC\_USE\_3D\_PHYSICS && CC\_ENABLE\_BULLET\_INTEGRATION) || CC\_USE\_NAVMESH)

\_runningScene->stepPhysicsAndNavigation(\_deltaTime);

#endif

//clear draw stats

\_renderer->clearDrawStats();

//render the scene

\_runningScene->render(\_renderer);

\_eventDispatcher->dispatchEvent(\_eventAfterVisit);

}

// draw the notifications node

if (\_notificationNode)

{

\_notificationNode->visit(\_renderer, Mat4::IDENTITY, 0);

}

if (\_displayStats)

{

showStats();

}

\_renderer->render();

\_eventDispatcher->dispatchEvent(\_eventAfterDraw);

popMatrix(MATRIX\_STACK\_TYPE::MATRIX\_STACK\_MODELVIEW);

\_totalFrames++;

// swap buffers

if (\_openGLView)

{

\_openGLView->swapBuffers();

}

if (\_displayStats)

{

calculateMPF();

}

}