异步加载可以通过TextureCache中的addImageAsync实现，同时可以提供一个回调函数表示资源已经加载完毕，然后通过SpriteFrameCache进行解析获取所有的SpriteFrame，注意，如果里面有点九图，以.9.png进行判断，也可以正确进行解析，plist文件中没有存放解析的四个点，cocos2dx是再运行时对文件进行解析获取分割四个点的，同时这些点九图的信息存放在对应的Texture2D的NinePatchInfo\* \_ninePatchInfo中，这个信息是由addSpriteFramesWithFile函数中实现的，但是如果想获取这个SpriteFrame的四个点的信息是无法获取的，因为Texture2D中的所有点九图的函数接口都是private的。

同时，对于点九图Scale9Sprite而言，调用initWithSpriteFrame会在当前的SpriteFrame对应的Texuture2D中寻找点九的四个分割点的信息，与此同时它会设置ContenteSize为Sprite的大小，如果要设置点九图的大小，则需要重新设置Scale9Sprite的contentSize

注意：对于Sprite而言，我们获取了SpriteFrame之后不需要重新initWithSpriteFrame，直接setSpriteFrame即可，但是对于点九图还是需要init，因为在Scale9Sprite中的setSpriteFrame并没有从Texture中寻找SpriteFrame的四个分割点，但是如果父节点如果将opacity中串联的话，我们在重新设置Scale9Sprite的opacity后仍然会生效。

void LoadingLayer::loadResources()

{

resourcesCapacity = sizeof(loadingTextureName) / sizeof(loadingTextureName[0]);

for(int i = 0; i < resourcesCapacity; i++)

{

TextureCache::getInstance()->addImageAsync(loadingTextureName[i], CC\_CALLBACK\_1(LoadingLayer::loadTexturesCallback, this));

}

}

void LoadingLayer::loadTexturesCallback(Texture2D \*texture)

{

if(texture == nullptr)

{

log("loadcallback failed!");

}

else

{

SpriteFrameCache::getInstance()->addSpriteFramesWithFile(loadingPlistName[resourcesAmount], texture);

if(resourcesAmount == 0)

{

SpriteFrameCache \*spriteFrameCache = SpriteFrameCache::getInstance();

SpriteFrame \*background = spriteFrameCache->getSpriteFrameByName(backgroundName);

SpriteFrame \*progressBackground = spriteFrameCache->getSpriteFrameByName(progressBackgroundName);

SpriteFrame \*progress = spriteFrameCache->getSpriteFrameByName(progressName);

this->backgroundSprite->initWithSpriteFrame(background);

this->progressBackground->initWithSpriteFrame(progressBackground);

this->progressTimer->initWithSprite(progressSprite);

this->backgroundSprite->setContentSize(Size(960, 640));

this->progressSprite->initWithSpriteFrame(progress);

this->progressTimer->setPercentage(0.5f);

}

resourcesAmount++;

log("resourcesAmount = %d", resourcesAmount);

}

}

源码分析：

（1）加载Texture2D

void TextureCache::addImageAsync(const std::string &path, const std::function<void(Texture2D\*)>& callback)

{

Texture2D \*texture = nullptr;

std::string fullpath = FileUtils::getInstance()->fullPathForFilename(path);

auto it = \_textures.find(fullpath);

if( it != \_textures.end() )

texture = it->second;

if (texture != nullptr)

{

if (callback) callback(texture);

return;

}

// check if file exists

if ( fullpath.empty() || ! FileUtils::getInstance()->isFileExist( fullpath ) ) {

if (callback) callback(nullptr);

return;

}

// lazy init

if (\_loadingThread == nullptr)

{

// create a new thread to load images

**\_loadingThread = new std::thread(&TextureCache::loadImage, this);**

**\_needQuit = false;**

}

**if (0 == \_asyncRefCount) // 这里很重要，即使分几次加载也不会出现问题**

**{**

**Director::getInstance()->getScheduler()->schedule(CC\_SCHEDULE\_SELECTOR(TextureCache::addImageAsyncCallBack), this, 0, false);**

**}**

++\_asyncRefCount;

// generate async struct

AsyncStruct \*data = new (std::nothrow) AsyncStruct(fullpath, callback);

// add async struct into queue

\_asyncStructQueue.push\_back(data);

\_requestMutex.lock();

\_requestQueue.push\_back(data);

\_requestMutex.unlock();

**\_sleepCondition.notify\_one();**

}

**void TextureCache::loadImage() // 线程调用函数**

{

AsyncStruct \*asyncStruct = nullptr;

std::mutex signalMutex;

std::unique\_lock<std::mutex> signal(signalMutex);

while (!\_needQuit)

{

// pop an AsyncStruct from request queue

\_requestMutex.lock();

if(\_requestQueue.empty())

{

asyncStruct = nullptr;

}else

{

asyncStruct = \_requestQueue.front();

\_requestQueue.pop\_front();

}

\_requestMutex.unlock();

if (nullptr == asyncStruct) { // 如果请求队列为空，则阻塞该线程

**\_sleepCondition.wait(signal);**

continue;

}

// load image

asyncStruct->loadSuccess = asyncStruct->image.initWithImageFileThreadSafe(asyncStruct->filename);

// push the asyncStruct to response queue

\_responseMutex.lock();

\_responseQueue.push\_back(asyncStruct);

\_responseMutex.unlock();

}

}

**void TextureCache::addImageAsyncCallBack(float dt)**

{

Texture2D \*texture = nullptr;

AsyncStruct \*asyncStruct = nullptr;

while (true)

{

// pop an AsyncStruct from response queue

\_responseMutex.lock();

if(\_responseQueue.empty())

{

asyncStruct = nullptr;

}else

{

asyncStruct = \_responseQueue.front();

\_responseQueue.pop\_front();

// the asyncStruct's sequence order in \_asyncStructQueue must equal to the order in \_responseQueue

CC\_ASSERT(asyncStruct == \_asyncStructQueue.front());

\_asyncStructQueue.pop\_front();

}

\_responseMutex.unlock();

if (nullptr == asyncStruct) {

break;

}

// check the image has been convert to texture or not

auto it = \_textures.find(asyncStruct->filename);

if(it != \_textures.end())

{

texture = it->second;

}

else

{

// convert image to texture

if (asyncStruct->loadSuccess)

{

Image\* image = &(asyncStruct->image);

// generate texture in render thread

texture = new (std::nothrow) Texture2D();

texture->initWithImage(image);

//parse 9-patch info

this->parseNinePatchImage(image, texture, asyncStruct->filename);

#if CC\_ENABLE\_CACHE\_TEXTURE\_DATA

// cache the texture file name

VolatileTextureMgr::addImageTexture(texture, asyncStruct->filename);

#endif

// cache the texture. retain it, since it is added in the map

\_textures.insert( std::make\_pair(asyncStruct->filename, texture) );

texture->retain();

texture->autorelease();

} else {

texture = nullptr;

CCLOG("cocos2d: failed to call TextureCache::addImageAsync(%s)", asyncStruct->filename.c\_str());

}

}

// call callback function

if (asyncStruct->callback)

{

(asyncStruct->callback)(texture);

}

// release the asyncStruct

delete asyncStruct;

--\_asyncRefCount;

}

if (0 == \_asyncRefCount)

{

Director::getInstance()->getScheduler()->unschedule(CC\_SCHEDULE\_SELECTOR(TextureCache::addImageAsyncCallBack), this);

}

}

总的过程就是：

在第一个添加Image的时候会创建一个线程，在每次添加一个Image的时候会放入一个AsyncStruct进入请求队列，AsyncStruct的结构如下所示，同时也会通过条件变量告诉被阻塞的线程可以继续进行纹理加载，在loadImage的过程中，会将其中的image填充，同时将该AsyncStruct放入响应队列。由于在最开始**TextureCache::addImageAsyncCallBack**函数放入了调度器，因此每帧均会调用，如果有响应消息，则会通过AsyncStruct中的Image创建一个Texture2D，同时调用回调函数，如果没有响应消息，则直接退出循环，同时判断当前纹理加载的个数是否等于0，如果等于0，则将该函数从调度器上移除，如果不等于0，则不移除

多次加载的情况，创建线程仅仅在当前线程==nullptr的情况，这样不会创建多个加载线程，如果当前纹理加载的个数等于0，则重新将addImageAsyncCallBack函数放入调度器，这样可以保证多次加载也没有问题

异步添加Image做了如下三件事（都是在TextureCache中做得）：

1 如果加载线程为nullptr 则创建一个线程调用loadImage，用于加载ImageData

2 如果异步加载的数量为0，则将一个callBack函数放到调度器中，每帧调用

3 创建一个AsyncStuuct加入到请求队列，并唤醒等待条件变量的线程

线程loadImage完成做了如下的事：

创建一个响应消息放入响应队列

CallBack函数在调度的时候做了如下的事：

从响应队列中取出一个响应，通过ImageData创建一个Texture2D，如果之间设置了加载完成的回调函数，则通知Texture2D加载完成的回调函数，同时会将它Texture2D加入到TextureCache中，在这里还会检查点九图。如果此时请求队列为空，则从调度器中取消该CallBack函数

struct TextureCache::AsyncStruct

{

public:

AsyncStruct(const std::string& fn, std::function<void(Texture2D\*)> f) : filename(fn), callback(f), loadSuccess(false) {}

std::string filename;

std::function<void(Texture2D\*)> callback;

Image image;

bool loadSuccess;

};

（2）获取Texture2D及SpriteFrame

SpriteFrameCache::getInstance()->addSpriteFramesWithFile(loadingPlistName[resourcesAmount], texture);

通过上面的函数，可以将当前的纹理分成小图片，及放入SpriteFrameCache中，

上面的函数会调用

addSpriteFramesWithDictionary(dict, texture);

void SpriteFrameCache::addSpriteFramesWithDictionary(ValueMap& dictionary, Texture2D\* texture)

{

/\*

Supported Zwoptex Formats:

ZWTCoordinatesFormatOptionXMLLegacy = 0, // Flash Version

ZWTCoordinatesFormatOptionXML1\_0 = 1, // Desktop Version 0.0 - 0.4b

ZWTCoordinatesFormatOptionXML1\_1 = 2, // Desktop Version 1.0.0 - 1.0.1

ZWTCoordinatesFormatOptionXML1\_2 = 3, // Desktop Version 1.0.2+

Version 3 with TexturePacker 4.0 polygon mesh packing

\*/

if (dictionary["frames"].getType() != cocos2d::Value::Type::MAP)

return;

ValueMap& framesDict = dictionary["frames"].asValueMap();

int format = 0;

Size textureSize;

// get the format

if (dictionary.find("metadata") != dictionary.end())

{

ValueMap& metadataDict = dictionary["metadata"].asValueMap();

format = metadataDict["format"].asInt();

if(metadataDict.find("size") != metadataDict.end())

{

textureSize = SizeFromString(metadataDict["size"].asString());

}

}

// check the format

CCASSERT(format >=0 && format <= 3, "format is not supported for SpriteFrameCache addSpriteFramesWithDictionary:textureFilename:");

auto textureFileName = Director::getInstance()->getTextureCache()->getTextureFilePath(texture);

Image\* image = nullptr;

NinePatchImageParser parser;

for (auto iter = framesDict.begin(); iter != framesDict.end(); ++iter)

{

ValueMap& frameDict = iter->second.asValueMap();

std::string spriteFrameName = iter->first;

SpriteFrame\* spriteFrame = \_spriteFrames.at(spriteFrameName);

if (spriteFrame)

{

continue;

}

if(format == 0)

{

float x = frameDict["x"].asFloat();

float y = frameDict["y"].asFloat();

float w = frameDict["width"].asFloat();

float h = frameDict["height"].asFloat();

float ox = frameDict["offsetX"].asFloat();

float oy = frameDict["offsetY"].asFloat();

int ow = frameDict["originalWidth"].asInt();

int oh = frameDict["originalHeight"].asInt();

// check ow/oh

if(!ow || !oh)

{

CCLOGWARN("cocos2d: WARNING: originalWidth/Height not found on the SpriteFrame. AnchorPoint won't work as expected. Regenerate the .plist");

}

// abs ow/oh

ow = abs(ow);

oh = abs(oh);

// create frame

spriteFrame = SpriteFrame::createWithTexture(texture,

Rect(x, y, w, h),

false,

Vec2(ox, oy),

Size((float)ow, (float)oh)

);

}

else if(format == 1 || format == 2)

{

Rect frame = RectFromString(frameDict["frame"].asString());

bool rotated = false;

// rotation

if (format == 2)

{

rotated = frameDict["rotated"].asBool();

}

Vec2 offset = PointFromString(frameDict["offset"].asString());

Size sourceSize = SizeFromString(frameDict["sourceSize"].asString());

// create frame

spriteFrame = SpriteFrame::createWithTexture(texture,

frame,

rotated,

offset,

sourceSize

);

}

else if (format == 3)

{

// get values

Size spriteSize = SizeFromString(frameDict["spriteSize"].asString());

Vec2 spriteOffset = PointFromString(frameDict["spriteOffset"].asString());

Size spriteSourceSize = SizeFromString(frameDict["spriteSourceSize"].asString());

Rect textureRect = RectFromString(frameDict["textureRect"].asString());

bool textureRotated = frameDict["textureRotated"].asBool();

// get aliases

ValueVector& aliases = frameDict["aliases"].asValueVector();

for(const auto &value : aliases) {

std::string oneAlias = value.asString();

if (\_spriteFramesAliases.find(oneAlias) != \_spriteFramesAliases.end())

{

CCLOGWARN("cocos2d: WARNING: an alias with name %s already exists", oneAlias.c\_str());

}

\_spriteFramesAliases[oneAlias] = Value(spriteFrameName);

}

// create frame

spriteFrame = SpriteFrame::createWithTexture(texture,

Rect(textureRect.origin.x, textureRect.origin.y, spriteSize.width, spriteSize.height),

textureRotated,

spriteOffset,

spriteSourceSize);

if(frameDict.find("vertices") != frameDict.end())

{

std::vector<int> vertices;

parseIntegerList(frameDict["vertices"].asString(), vertices);

std::vector<int> verticesUV;

parseIntegerList(frameDict["verticesUV"].asString(), verticesUV);

std::vector<int> indices;

parseIntegerList(frameDict["triangles"].asString(), indices);

PolygonInfo info;

initializePolygonInfo(textureSize, spriteSourceSize, vertices, verticesUV, indices, info);

spriteFrame->setPolygonInfo(info);

}

}

**bool flag = NinePatchImageParser::isNinePatchImage(spriteFrameName);**

**if(flag)**

**{**

**if (image == nullptr) {**

**image = new Image();**

**image->initWithImageFile(textureFileName);**

**}**

**parser.setSpriteFrameInfo(image, spriteFrame->getRectInPixels(), spriteFrame->isRotated());**

**texture->addSpriteFrameCapInset(spriteFrame, parser.parseCapInset());**

}

// add sprite frame

\_spriteFrames.insert(spriteFrameName, spriteFrame);

}

CC\_SAFE\_DELETE(image);

}

void Texture2D::addSpriteFrameCapInset(SpriteFrame\* spritframe, const Rect& capInsets)

{

if(nullptr == \_ninePatchInfo)

{

\_ninePatchInfo = new NinePatchInfo;

}

if(nullptr == spritframe)

{

\_ninePatchInfo->capInsetSize = capInsets;

}

else

{

\_ninePatchInfo->capInsetMap[spritframe] = capInsets;

}

}

class NinePatchInfo

{

public:

Rect capInsetSize;

std::unordered\_map<SpriteFrame\*, Rect> capInsetMap;

};

这个函数会解析plist文件，同时检测当前的SpriteFrame是否是点九图，如果是点九图，则将它放在Texture2D中，Texture2D中存放了一个NinePatchInfo结构，用于保存SpriteFrame的点九信息（实际上就是四个分割点组成的一个矩形）