cocos2d-x中有三个雷可以在层或精灵中添加文字:

**LabelTTF LabelBMFont LabelAtlas**

LabelTTF直接支持TTF字体，可以支持全部的中文，但效率稍低。

LabelBMFont适合显示特定的文字，通过预先将文字生成图片，提高了效率，但是不能支 持全部中文

LabelAtlas适合使用的文字不多，组合多，但是文字的编码是连续的，比如数字或者英文字符

**LabelTTF**

**Label\* \_renderLabel;**

bool \_contentDirty;

FontDefinition \_fontDef;

实际上LabelTTF保存中了一个Label，并且这个Label作为该LabelTTF的子节点，查看LabelTTF中的构造函数：

**LabelTTF::LabelTTF()**

{

\_renderLabel = Label::create();

\_renderLabel->setAnchorPoint(Vec2::ANCHOR\_BOTTOM\_LEFT);

this->addChild(\_renderLabel);

this->setAnchorPoint(Vec2::ANCHOR\_MIDDLE);

\_contentDirty = false;

\_cascadeColorEnabled = true;

\_cascadeOpacityEnabled = true;

}

查看LabelTTF的visit函数：

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

{

if (\_contentDirty)

{

this->setContentSize(\_renderLabel->getContentSize());

\_contentDirty = false;

}

Node::visit(renderer,parentTransform, parentFlags);

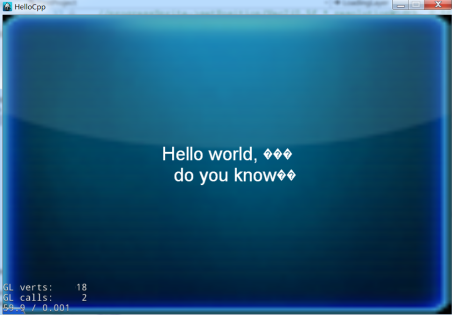
}

实际上这里操作的都是Label，这里获取的ContentSize也是Label的ContentSize，Node：：Visit中，因为Label作为LabelTTF的子节点，因此也最终会调用Label的draw函数。

LabelTTF \*label1 = LabelTTF::create("Hello world, 你好\n do you know！", "arial.ttf", 40);

label1->setPosition(Vec2(0.5f \* resolutionWidth, 0.5f \* resolutionHeight));

addChild(label1);



**LabelBMFont**

std::string \_fntFile;

Label\* \_label;

实际上LabelBMFont也是保存了一个Label，并且这个Label作为LabelBMFont的子节点，查看其构造函数：

LabelBMFont::LabelBMFont()

{

\_label = Label::create();

\_label->setAnchorPoint(Vec2::ANCHOR\_BOTTOM\_LEFT);

this->addChild(\_label);

this->setAnchorPoint(Vec2::ANCHOR\_MIDDLE);

\_cascadeOpacityEnabled = true;

#if CC\_LABELBMFONT\_DEBUG\_DRAW

\_debugDrawNode = DrawNode::create();

addChild(\_debugDrawNode);

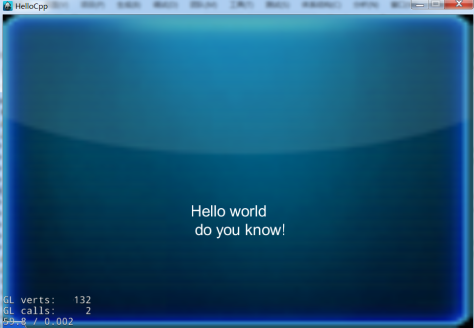
#endif

}

LabelBMFont \*label2 = LabelBMFont::create("Hello world \n do you know!", "font.fnt");

label2->setPosition(Vec2(0.5f \* resolutionWidth, 0.5f \* resolutionHeight - 100.0f));

addChild(label2);



**终极大boss：Label**

Label中包含了一个SpriteBatchNode\*的数组Vector

在通过bmfontFilePath进行创建LabelBMFont的时候，实际上是需要得到一个**FontAtlas，**实际上这些都存放在FontAtlasCache中，这里面存放的一个map映射中

**class CC\_DLL FontAtlas : public Ref**

{

public:

static const int CacheTextureWidth;

static const int CacheTextureHeight;

static const char\* CMD\_PURGE\_FONTATLAS;

static const char\* CMD\_RESET\_FONTATLAS;

/\*\*

\* @js ctor

\*/

FontAtlas(Font &theFont);

/\*\*

\* @js NA

\* @lua NA

\*/

virtual ~FontAtlas();

void addLetterDefinition(char16\_t utf16Char, const FontLetterDefinition &letterDefinition);

bool getLetterDefinitionForChar(char16\_t utf16Char, FontLetterDefinition &letterDefinition);

bool prepareLetterDefinitions(const std::u16string& utf16String);

inline const std::unordered\_map<ssize\_t, Texture2D\*>& getTextures() const{ return \_atlasTextures;}

void addTexture(Texture2D \*texture, int slot);

float getLineHeight() const { return \_lineHeight; }

void setLineHeight(float newHeight);

Texture2D\* getTexture(int slot);

const Font\* getFont() const { return \_font; }

/\*\* listen the event that renderer was recreated on Android/WP8

It only has effect on Android and WP8.

\*/

void listenRendererRecreated(EventCustom \*event);

/\*\* Removes textures atlas.

It will purge the textures atlas and if multiple texture exist in the FontAtlas.

\*/

void purgeTexturesAtlas();

/\*\* sets font texture parameters:

- GL\_TEXTURE\_MIN\_FILTER = GL\_LINEAR

- GL\_TEXTURE\_MAG\_FILTER = GL\_LINEAR

\*/

void setAntiAliasTexParameters();

/\*\* sets font texture parameters:

- GL\_TEXTURE\_MIN\_FILTER = GL\_NEAREST

- GL\_TEXTURE\_MAG\_FILTER = GL\_NEAREST

\*/

void setAliasTexParameters();

protected:

void relaseTextures();

void findNewCharacters(const std::u16string& u16Text, std::unordered\_map<unsigned short, unsigned short>& charCodeMap);

void conversionU16TOGB2312(const std::u16string& u16Text, std::unordered\_map<unsigned short, unsigned short>& charCodeMap);

/\*\*

\* Scale each font letter by scaleFactor.

\*

\* @param scaleFactor A float scale factor for scaling font letter info.

\*/

void scaleFontLetterDefinition(float scaleFactor);

**std::unordered\_map<ssize\_t, Texture2D\*> \_atlasTextures;**

**std::unordered\_map<char16\_t, FontLetterDefinition> \_letterDefinitions;**

float \_lineHeight;

Font\* \_font;

FontFreeType\* \_fontFreeType;

void\* \_iconv;

// Dynamic GlyphCollection related stuff

int \_currentPage;

unsigned char \*\_currentPageData;

int \_currentPageDataSize;

float \_currentPageOrigX;

float \_currentPageOrigY;

int \_letterPadding;

int \_letterEdgeExtend;

int \_fontAscender;

EventListenerCustom\* \_rendererRecreatedListener;

bool \_antialiasEnabled;

int \_currLineHeight;

friend class Label;

};

这里面最关键的是**\_letterDefinitions**，这里是一个从字符的编码到FontLetterDefinition的映射，其中FontLetterDefinition的结构如下：

struct FontLetterDefinition

{

float U;

float V;

float width;

float height;

float offsetX;

float offsetY;

int textureID;

bool validDefinition;

int xAdvance;

};

如果在FontAtalsCache中没有找到FontAtals，则重新创建一个，在创建的时候是通过FontFNT进行创建的，FontFNT中包含了一个BMFontConfiguration类：

class CC\_DLL BMFontConfiguration : public Ref

{

// FIXME: Creating a public interface so that the bitmapFontArray[] is accessible

public://@public

// BMFont definitions

**tFontDefHashElement \*\_fontDefDictionary;**

//! FNTConfig: Common Height Should be signed (issue #1343)

int \_commonHeight;

//! Padding

BMFontPadding \_padding;

//! atlas name

std::string \_atlasName;

//! values for kerning

**tKerningHashElement \*\_kerningDictionary;**

// Character Set defines the letters that actually exist in the font

**std::set<unsigned int> \*\_characterSet;**

//! Font Size

int \_fontSize;

public:

/\*\*

\* @js ctor

\*/

BMFontConfiguration();

/\*\*

\* @js NA

\* @lua NA

\*/

virtual ~BMFontConfiguration();

/\*\*

\* @js NA

\* @lua NA

\*/

std::string description() const;

/\*\* allocates a BMFontConfiguration with a FNT file \*/

static BMFontConfiguration \* create(const std::string& FNTfile);

/\*\* initializes a BitmapFontConfiguration with a FNT file \*/

bool initWithFNTfile(const std::string& FNTfile);

inline const std::string& getAtlasName(){ return \_atlasName; }

inline void setAtlasName(const std::string& atlasName) { \_atlasName = atlasName; }

std::set<unsigned int>\* getCharacterSet() const;

private:

std::set<unsigned int>\* parseConfigFile(const std::string& controlFile);

std::set<unsigned int>\* parseBinaryConfigFile(unsigned char\* pData, unsigned long size, const std::string& controlFile);

void parseCharacterDefinition(const char\* line, BMFontDef \*characterDefinition);

void parseInfoArguments(const char\* line);

void parseCommonArguments(const char\* line);

void parseImageFileName(const char\* line, const std::string& fntFile);

void parseKerningEntry(const char\* line);

void purgeKerningDictionary();

void purgeFontDefDictionary();

};

这个类中保存的是最原始的fnt中的文件信息，主要包含了两个字典，一个地点是FontDef字典，一个是Kerning字典，

typedef struct \_BMFontDef {

//! ID of the character

unsigned int charID;

//! origin and size of the font

Rect rect;

//! The X amount the image should be offset when drawing the image (in pixels)

short xOffset;

//! The Y amount the image should be offset when drawing the image (in pixels)

short yOffset;

//! The amount to move the current position after drawing the character (in pixels)

short xAdvance;

} BMFontDef;

Label中的关键计算updateContent：

void Label::updateContent()

{

if (\_systemFontDirty)

{

if (\_fontAtlas)

{

\_batchNodes.clear();

FontAtlasCache::releaseFontAtlas(\_fontAtlas);

\_fontAtlas = nullptr;

}

\_systemFontDirty = false;

}

CC\_SAFE\_RELEASE\_NULL(\_textSprite);

CC\_SAFE\_RELEASE\_NULL(\_shadowNode);

bool updateFinished = true;

if (\_fontAtlas)

{

std::u16string utf16String;

if (StringUtils::UTF8ToUTF16(\_utf8Text, utf16String))

{

\_utf16Text = utf16String;

}

// 计算当前字符串的kernings

**computeHorizontalKernings(\_utf16Text);**

**updateFinished = alignText();**

}

else

{

auto fontDef = \_getFontDefinition();

createSpriteForSystemFont(fontDef);

if (\_shadowEnabled)

{

createShadowSpriteForSystemFont(fontDef);

}

}

if(updateFinished){

\_contentDirty = false;

}

#if CC\_LABEL\_DEBUG\_DRAW

\_debugDrawNode->clear();

Vec2 vertices[4] =

{

Vec2::ZERO,

Vec2(\_contentSize.width, 0),

Vec2(\_contentSize.width, \_contentSize.height),

Vec2(0, \_contentSize.height)

};

\_debugDrawNode->drawPoly(vertices, 4, true, Color4F::WHITE);

#endif

}

最终还会调用Sprite的updateTransiom函数，且该函数仅仅在Sprite设置了BatchNode才会调用：

void Sprite::updateTransform(void)

{

**CCASSERT(\_batchNode, "updateTransform is only valid when Sprite is being rendered using an SpriteBatchNode");**

// recalculate matrix only if it is dirty

if( isDirty() ) {

// If it is not visible, or one of its ancestors is not visible, then do nothing:

if( !\_visible || ( \_parent && \_parent != \_batchNode && static\_cast<Sprite\*>(\_parent)->\_shouldBeHidden) )

{

\_quad.br.vertices.setZero();

\_quad.tl.vertices.setZero();

\_quad.tr.vertices.setZero();

\_quad.bl.vertices.setZero();

\_shouldBeHidden = true;

}

else

{

\_shouldBeHidden = false;

if( ! \_parent || \_parent == \_batchNode )

{

\_transformToBatch = getNodeToParentTransform();

}

else

{

CCASSERT( dynamic\_cast<Sprite\*>(\_parent), "Logic error in Sprite. Parent must be a Sprite");

const Mat4 &nodeToParent = getNodeToParentTransform();

Mat4 &parentTransform = static\_cast<Sprite\*>(\_parent)->\_transformToBatch;

\_transformToBatch = parentTransform \* nodeToParent;

}

//

// calculate the Quad based on the Affine Matrix

//

Size &size = \_rect.size;

float x1 = \_offsetPosition.x;

float y1 = \_offsetPosition.y;

float x2 = x1 + size.width;

float y2 = y1 + size.height;

float x = \_transformToBatch.m[12];

float y = \_transformToBatch.m[13];

float cr = \_transformToBatch.m[0];

float sr = \_transformToBatch.m[1];

float cr2 = \_transformToBatch.m[5];

float sr2 = -\_transformToBatch.m[4];

float ax = x1 \* cr - y1 \* sr2 + x;

float ay = x1 \* sr + y1 \* cr2 + y;

float bx = x2 \* cr - y1 \* sr2 + x;

float by = x2 \* sr + y1 \* cr2 + y;

float cx = x2 \* cr - y2 \* sr2 + x;

float cy = x2 \* sr + y2 \* cr2 + y;

float dx = x1 \* cr - y2 \* sr2 + x;

float dy = x1 \* sr + y2 \* cr2 + y;

\_quad.bl.vertices.set(SPRITE\_RENDER\_IN\_SUBPIXEL(ax), SPRITE\_RENDER\_IN\_SUBPIXEL(ay), \_positionZ);

\_quad.br.vertices.set(SPRITE\_RENDER\_IN\_SUBPIXEL(bx), SPRITE\_RENDER\_IN\_SUBPIXEL(by), \_positionZ);

\_quad.tl.vertices.set(SPRITE\_RENDER\_IN\_SUBPIXEL(dx), SPRITE\_RENDER\_IN\_SUBPIXEL(dy), \_positionZ);

\_quad.tr.vertices.set(SPRITE\_RENDER\_IN\_SUBPIXEL(cx), SPRITE\_RENDER\_IN\_SUBPIXEL(cy), \_positionZ);

setTextureCoords(\_rect);

}

// MARMALADE CHANGE: ADDED CHECK FOR nullptr, TO PERMIT SPRITES WITH NO BATCH NODE / TEXTURE ATLAS

if (\_textureAtlas)

{

\_textureAtlas->updateQuad(&\_quad, \_atlasIndex);

}

\_recursiveDirty = false;

setDirty(false);

}

// MARMALADE CHANGED

// recursively iterate over children

/\* if( \_hasChildren )

{

// MARMALADE: CHANGED TO USE Node\*

// NOTE THAT WE HAVE ALSO DEFINED virtual Node::updateTransform()

arrayMakeObjectsPerformSelector(\_children, updateTransform, Sprite\*);

}\*/

Node::updateTransform();

}

**再看看LabelTTF**

首先查看一个结构体TTFConfig

typedef struct \_ttfConfig

{

std::string fontFilePath;

float fontSize;

GlyphCollection glyphs;

const char \*customGlyphs;

bool distanceFieldEnabled;

int outlineSize;

\_ttfConfig(const std::string& filePath = "",float size = 12, const GlyphCollection& glyphCollection = GlyphCollection::DYNAMIC,

const char \*customGlyphCollection = nullptr, bool useDistanceField = false, int outline = 0)

: fontFilePath(filePath)

, fontSize(size)

, glyphs(glyphCollection)

, customGlyphs(customGlyphCollection)

, distanceFieldEnabled(useDistanceField)

, outlineSize(outline)

{

if(outline > 0)

{

distanceFieldEnabled = false;

}

}

} TTFConfig;

在创建的过程中，它将ttf文件读出来放入一个Data（保存了一个char数组以及大小）中，同时将其添加入缓存s\_cacheFontData（它是FontFreeType中的一个静态类型）中，以便可以通过文件名找到对应的Data，通过Data填充FontFreeType（继承于Font）中的数据内容，然后通过FontFreeType创建一个FontAtlas，放入到FontAtlasCache中。

在通过FontFreeType创建FontAtlas的过程中，会自动创建Texture2D

FontAtlas::FontAtlas(Font &theFont)

: \_font(&theFont)

, \_fontFreeType(nullptr)

, \_iconv(nullptr)

, \_currentPageData(nullptr)

, \_fontAscender(0)

, \_rendererRecreatedListener(nullptr)

, \_antialiasEnabled(true)

, \_currLineHeight(0)

{

\_font->retain();

**\_fontFreeType = dynamic\_cast<FontFreeType\*>(\_font);**

**if (\_fontFreeType) // 当前font是FontFreeType**

{

\_lineHeight = \_font->getFontMaxHeight();

\_fontAscender = \_fontFreeType->getFontAscender();

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

\_currentPage = 0;

\_currentPageOrigX = 0;

\_currentPageOrigY = 0;

\_letterEdgeExtend = 2;

\_letterPadding = 0;

if (\_fontFreeType->isDistanceFieldEnabled())

{

\_letterPadding += 2 \* FontFreeType::DistanceMapSpread;

}

\_currentPageDataSize = CacheTextureWidth \* CacheTextureHeight;

auto outlineSize = \_fontFreeType->getOutlineSize();

if(outlineSize > 0)

{

\_lineHeight += 2 \* outlineSize;

\_currentPageDataSize \*= 2;

}

\_currentPageData = new unsigned char[\_currentPageDataSize];

memset(\_currentPageData, 0, \_currentPageDataSize);

auto pixelFormat = outlineSize > 0 ? Texture2D::PixelFormat::AI88 : Texture2D::PixelFormat::A8;

texture->initWithData(\_currentPageData, \_currentPageDataSize,

pixelFormat, CacheTextureWidth, CacheTextureHeight, Size(CacheTextureWidth,CacheTextureHeight) );

addTexture(texture,0);

texture->release();

#if CC\_ENABLE\_CACHE\_TEXTURE\_DATA

auto eventDispatcher = Director::getInstance()->getEventDispatcher();

\_rendererRecreatedListener = EventListenerCustom::create(EVENT\_RENDERER\_RECREATED, CC\_CALLBACK\_1(FontAtlas::listenRendererRecreated, this));

eventDispatcher->addEventListenerWithFixedPriority(\_rendererRecreatedListener, 1);

#endif

}

}

查看Font，实际上它是一个抽象类，继承它的类都需要实现一个createFontAtlas的方法：

class CC\_DLL Font : public Ref

{

public:

**virtual FontAtlas\* createFontAtlas() = 0;**

virtual int\* getHorizontalKerningForTextUTF16(const std::u16string& text, int &outNumLetters) const = 0;

virtual int getFontMaxHeight() const { return 0; }

};

**比较LabelBMFont和LabelTTF可知：**

实际上两者都形成了一个FontAtlas，保存在FontAtlasCache的映射中，其中LabelBMFont是由FontFNT创建，而LabelTTF是由FontFreeType创建，每个FontAtlas包含了下面几个结构：

std::unordered\_map<ssize\_t, Texture2D\*> \_atlasTextures;

std::unordered\_map<char16\_t, FontLetterDefinition> \_letterDefinitions;

FontFreeType\* \_fontFreeType;

1 创建FontFNT的时候借助于FontConfiguration（包含fnt文件信息），然后借助FontFNT中的FontConfiguration创建FontLetterDefinition

2 创建FontFreeType的时候直接创建，但其中会保存一个映射关系（文件名—-Data数据），然后通过FontFreeType创建FontAtlas，在FontAtlas的构造函数中，会判断当前的Font是否为FontFreeType类型，进行各种属性的赋值，同时生成了一个Texture2D（但是此时并没有数据呀？？？），但是它将整个FontFreeType放入到了FontAtlas中。

在visit的时候都会调用到updateContent函数，这里面针对FontFreeType会通过字符串调用prepareLetterDefinitions函数，通过FontFreeType创建FontLetterDefinition，加入到映射表中。**同时在这里通过Texture2D中的updateWithData填充了数据，也就是说如果我们使用TTF，它只是创建了一个纹理，然互不断修改纹理中的数据**

在Label中一张纹理对应一个SpriteBatchNode，将Label::alignText()，在这里面会调用reuseSprite的updateTransform函数

在Label中还保存了std::vector<LetterInfo> \_lettersInfo;

struct LetterInfo

{

char16\_t utf16Char;

bool valid;

**float positionX; // 注意这里的两个值已经考虑了xadvance和offsetX**

**float positionY;**

int atlasIndex;

int lineIndex;

};

对于系统字体而言：

它使用的类型是一个\_textSprite的精灵进行绘制，也就是说该Label的渲染实际上是直接调用的Sprite的渲染函数，直接调用createSpriteForSystemFont函数将文本渲染成一个Texture2D，所以如果改变它的内容会要重新创建一个新的纹理，效率很低

cocos2dx中的简单说明：

\* Label can be created with:

\* - A true type font file.

\* - A bitmap font file.

\* - A char map file.

\* - The built in system font.

首先查看LabelBMFont