# Gnash 在 ARM 嵌入式 Linux 平台的实现①

孟小华　杨福安 (暨南大学 计算机系 广东 广州 510632)

**摘　要：** Flash是Adobe推出的PC平台主流的多媒体格式标准和产品，但Adobe一直没有推出应的嵌入式解决方案。为了解决Flash格式动画文件在各种嵌入式设备上的播放问题，基于开源的GNU Gnash flash解决方案，实现了flash动画文件在ARM嵌入式Linux平台的播放。分析了Gnash系统的组成结构，研究了其各个功能模块的实现技术以及与其他模块和操作系统平台的关系。然后针对嵌入式系统主频低、内存小的特点,对Gnash的功能模块进行裁剪，去掉了ActionScript引擎等动画播放不需要的交互式部分功能，并在基于ARM的Intel XSCALE PXA270嵌入式linux平台实现了Gnash系统。

**关键词：** Gnash；Flash；嵌入式linux；ARM；GNU

## Implementation of Gnash System Based on the ARM and Embedded Linux

MENG Xiao-Hua, YANG Fu-An

(Department of Computer Science, Jinan University, Guangzhou 510632, China)

**Abstract:** Adobe Flash is a multimedia format standard in PC. However, Adobe has not provided a solution in embedded platform. In order to play Flash file in embedded device, this paper uses Gnash which is a GUN open source Flash solution to play Flash file in ARM embedded Linux. It analyses the framework of gansh. It also researches various functional modules and discusses the relations between functional modules and operating system platform. Because embedded environment has low clock speed and limited memory, the paper cuts off some Gnash's modules, e.g. interactive modules, which is not mandatory while ActionScript engine plays movies. And the Gnash is run in the Intel PXA270 Linux platform successfully.

**Keywords:** Gnash; Flash; embedded linux; ARM; GNU

## 1　引言

Adobe Flash是目前PC平台主流的多媒体格式标准，广泛应用在各种因特网网站和RIA应用系统中，其应用包括交互式网站、交互式动画和交互式视频播放等。由于其采用矢量图像和高压缩比的音视频编码文件格式，因此特别适合于网络和嵌入式环境的应用。在数字家庭和数字媒体等嵌入式应用领域更是具有很大的应用需求。由于Flash是Adobe的专利技术，Adobe到目前为止一直没有推出其相应的嵌入式解决方案。为了解决Flash格式动画文件在各种嵌入式设备上的播放问题[1-3]，本文基于GNU开发的flash解决方案——Gnash[4]，将其移植到ARM嵌入式Linux平台，实现了flash动画文件的播放。本文首先分析了Gnash系统的组成结构，研究了其各个功能模块的实现技术以及与其它模块和操作系统平台的关系。然后针对嵌入式应用的需求和嵌入式系统主频低、内存小的特点,对Gnash的功能模块进行裁剪和移植，去掉了ActionScript引擎等动画播放不需要的交互式

① 基金项目:广东省教育部产学研结合项目(2009B090300283)
收稿时间:2009-10-22;收到修改稿时间:2009-12-14

部分功能,并在基于 ARM 的 Intel XSCALE PXA270 linux 平台实现了 Gnash 系统。

## 2 Gnash项目介绍

Gnash 项目[5]的目的是为了创建一个支持 Adobe Flash 文件格式的开源独立播放器和浏览器插件，以此代替占领整个市场的 Adobe Flash Player。Gnash 来源于 GPL Flash 项目，并通过 GNU 的 GPL(通用公共许可证)发布。它的最新发布的版本是 0.8.5，现在已经支持绝大多数SWFV7功能特性以及ActionScript 2类库，并支持 SWF V8 和 V9 版的部份功能、流媒体及 XML 消息服务器等高级特性。支持几乎所有的流媒体格式(FLV, H264, MP3 等),同时在 GUI 方面,加入对 KDE4 / QT4 的支持。Gnash 的编码实现是支持跨平台的，可以在各种软硬件平台上编译和执行，包含 PowerPC, AMD64, ARM, MIPS 等处理器架构,以及基于 BSD 的 UNIX 类操作系统。但针对具体的受限的嵌入式应用环境实现时还必须做一定的移植和裁剪。

## 3 Gnash系统组成及结构分析

因为在移植的过程中我们必须要对 Gnash 进行裁剪，所以在裁剪前我们必须了解其组成结构和各个功能模块，Gnash 播放器在设计中采用了层次化与模块化的结构[6]，其结构如图 1 所示，其中重要的模块说明如下:

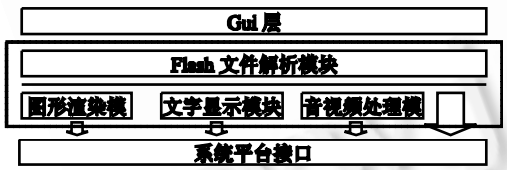


图 1 Gnash 系统层次结构

(1) Gui 层：Gnash 的应用层，接受与用户的交互行为，控制播放器的初始化和控制播放流程。如暂停、播放、回退等。

(2) Flash 文件的解析模块：解析 Flash 文件，将 Flash 文件的组成元素依次解析出来并存储到一个称为字典(Dictionary)的数据结构中去,等待图形,文字,声音处理模块的使用，当然在解析过程中会调用到系统函数。在 Flash 文件中所包含有的 ActionScript 脚本语言也是在此模块中进行解析工作。解析完成后将会形成一个显示类表即 DisplayList，用于控制每帧对 Character 的使用。

(3) 图形渲染模块：在播放器中和矢量图形相关的数据结构，实现了对矢量图形的高效存储。矢量图形是基于贝塞尔曲线的，再结合填充样式与线条样式，将一条条的记录(Record)组织起来形成的形状(Shape)。显示设备只能直接显示 RGB 位图，因此，最后要将图形呈现在用户面前，就必须将这些形状转换为位图。这个转换的过程，称为矢量图形的渲染。这是整个播放器运行过程中运算最复杂的步骤，是实现流畅播放的关键。

图形渲染模块的工作，主要是根据形状中的记录列表进行绘图操作，包括：移动画笔位置、绘制贝塞尔曲线、根据指定的填充样式对某封闭区域进行填充、根据指定的线条样式描线等。对于这些功能，许多流行的图形库都提供了相应或相似的功能。因此，我们可以选用适合的图形库来实现我们需要的渲染引擎。

考虑到不同平台的特点与需要，Gnash 实现了三个版本的渲染引擎，一个基于 Cairo，一个基于 AGG，还有一个基于 OpenGL。这三个渲染引擎模块的类图如图 2 所示：

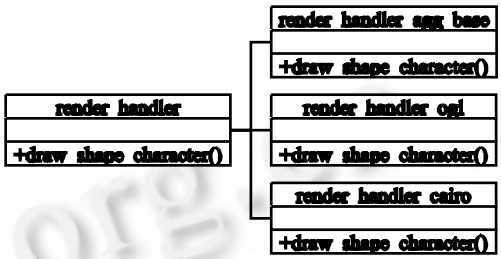


图 2 Gnash 渲染引擎模块类图

从图中我们可以看到,render_handler 类是整个渲染引擎的基类，它使用虚函数的方式定义了若干方法或接口；render_handler_ogl 类,render_ handler_cairo 类还有 render_handler_agg_base 类则是继承自 render_handler 基类，由它们再分别通过调用 Cairo 库，AGG 库与 OpenGL 库的 API，来实现 draw_ shape_character(shape_character_def *def, character *inst) 等一系列的函数，为上层应用提供一个统一的接口。而这三种渲染引擎之间的优缺点会在后文再加以分析。

(4) 文字显示模块[7]：实现 Flash 文件中文字的显示。Flash 文件中的文字有两种形式，一种是在 Flash 文件中使用 DefineFontTag 定义嵌入的字库信息，供

文本显示时使用，另一种则要通过调用外部的字库来显示。Gnash 播放器对这两种的文字显示形式皆能很好地支持。

在嵌入的字库信息中，Glyph Text 的定义由两个标签组成：DefineFont 标签定义字体形状(Glyph)的集合，DefineText 标签定义要显示的文本。Gnash 中的 DefineFontTag.h 为字体设计了如下的数据结构：

```
class DefineFontTag{
   public:
      static void loader(SWFStream& in, TagType tag,
movie_definition& m,const unInfo& r);
   const Font::GlyphInfoRecords& glyphTable() const
     { return _glyphTable; }
   Font::GlyphInfo   Records _glyphTable;// 字体形状集
合
      bool _italic;//斜体标识位
      bool _bold;//粗体标识位
   boost::shared_ptr<const Font::CodeTable>
_codeTable;//字符对应表
      ......//其他方法和变量};
```

DefineText 标签定义了要显示的字符串，同时指定了选用的字体、大小、颜色以及每一个字符的显示位置。Gnash 中的 DefineTextTag.h 为字体设计了如下的数据构：

```
class DefineTextTag {
public:
    static void loader(SWFStream& in, TagType tag,
    movie_definition& m, const unInfo& r);
    /// Draw the string.
    void display(character* inst);
    const rect& get_bound() const {return _rect; }
private:
    /// DefineText2Tag::loader also constructs a
DefineTextTag.
    friend class DefineText2Tag;
    /// Construct a DefineTextTag.
    /// This sho uld only be constructed using the loader()
functions.
    DefineTextTag(SWFStream& in, mo    vie_definition&
m, TagType tag){
       read(in, m, tag);   }
    rect _rect;
    SWFMatrix _matrix;//文本位置变换矩阵
    void read(SWFStream& in, movie_definition& m,
TagType tag);
    std::vector<TextRecord> _textRecords;// 文本记录集
合};
```

(5) 音视频处理模块：实现对 Flash 文件中由 StartSoundTag 标签嵌入的音频(有 ADPCM 与 MP3 两种格式)的解码、控制与播放。

Gnash 对声音的控制是采用标准的 SDL。声音可以分为 event-sound 和 soundstreams 两类。event-sound 被包含在一个单独的 SWF 帧中，但在播放时可以跨越多个帧。Soundstreams 则可以被分成多个 SWF 帧。这意味着我们可以对 soundstreams 进行回放操作。当要播放声音的时候，Gnash 便调用 sound handler 来进行相应操作。由于所有的操作都是由线程来完成的，所以在声音和图像播放时要保持它们的一致性。必须保证声音线程的安全性和互斥性。

在 gnash 源码 libsound/sound_handler.h 中定义了 sound_handler 用来存储操作音频。Sound-Envelope.h 中的 SoundEnvelope 控制 event sound。EmbedSound.h 声明对嵌入式声音的操作。InputStream.h 声明对音频输入的操作。

在 Gnash 源码 libmedia/AudioDecoder.h 中定义的 AudioDecoder 是音频解码的基类，针对依赖的不同软件包对音频的解码操作都要继承 Audio-Decoder。如图 3 所示：

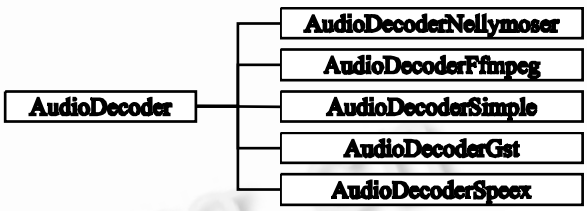


图 3　AudioDecoder 继承关系

在 Gnash 源码 libmedia/VideoDecoder.h 中定义了 VideoDecoder 是视频解码的基类，针对依赖不同软件包对视频的解码操作都要继承 Video-Decoder。如图 4 所示：

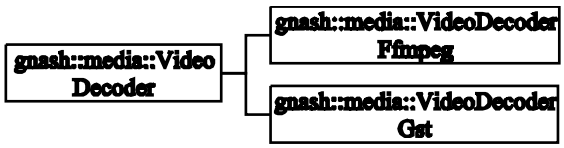


图 4　VideoDecoder 继承关系

在 gnash 源码 libmedia/MediaHandler.h 定义的 MediaHandler 类是提供解析和解码操作的工厂。针对依赖不同软件包对视频的解码操作都要继承 MediaHandler。如图 5 所示：

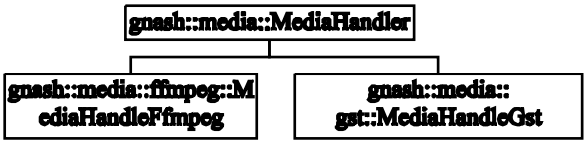


图 5　MediaDecoder 继承关系

(6) 系统平台接口：包括定时器操作、信号量、线程操作等平台相关的操作。

## 4　Gnash的嵌入式实现

### 4.1 Gnash 的依赖库分析

在前文中我们已经了解了 Gnash 的组成结构和各个功能模块的关系，现在我们就要开始根据嵌入式环境的特点来对 Gnash 进行裁剪，首先我们通过对 Gnash 的代码和文檔对所依赖的库详细的分析，我们可以把 Gansh 的库分为必须依赖库和可选库两类，对必须依赖库，我们是一定要进行交叉编译的，否则整个程序都会无法运行；对可选库，我们则可以根据嵌入式环境的要求进行裁剪，使之附合我们的实际需要，这个部份我们会在后文中会作重点探讨。必须依赖的库和说明如表 1 所列：

表 1　Gnash 的必须依赖库

| 名称 | 描述 | Package |
|---|---|---|
| PNG | PNG 相对于 GIF 来说是开源的，用于渲染 PNGs.文件 | 1.2.35 or higher libpng |
| JPG | JPEG 是被用在图像上的损失最小图像格式，用于渲染 JPEG 文件 | libjpeg-6b or higher |
| GIF | 用于渲染 GIF 文件 | 4.1.6 or higher giflib |
| Boost | Boost 是一个可移植的 C++类和模板类库，在 Gnash 中, Boost 类库被广泛使用 | 1.34.1 or higher boost |
| Libxml2 | Libxml2 是 GNOME XML 解析函数库，用来解析在 XML 和 XML Socket ActionScript 类中的消息 | 2.6.30 or higher libxml2 |
| Zlib | 用于资料压缩的函式库，使用 DEFLATE 算法 | 1.2.3 or higher zlib |
| Curl | ibcurl 是多种协议文件的转换工具提供，用来提供 URL 下载功能 | 7.19.4 or higher libcurl |
| libtool | 这是一个支持脚本的通用库函数 | 2.2.6a or higher |
| glib | 提供 C 语言一套高度可移植性、简单易学且通用的工具库 | 2.18.4 or higher |
| freetype | 用 C 语言实现的字型栅格化引擎制作的的一个函式库 | 2.3.6 or higher |

### 4.2 可选库的交叉编译

在编译[8]Gnash 前,我们要先指明交叉编译器，以避免因为环境设定错误而导致编译失败，所以我们先做了以下的环境设定。

```
export CC=arm-linux-gnu-gcc
export CXX=arm-linux-gnu-g++
export AR=arm-linux-gnu-ar
export STRIP=arm-linux-gnu-strip
export RANLIB=arm-linux-gnu-ranlib
export LD=arm-linux-gnu-ld
```

(1) 视频处理器的编译：在视频处理器中，Gnash 预设了两个视频处理器，如下表所示：

表 2　Gnash 的视频处理器

| | | |
|---|---|---|
| FFmpeg | FFMPEG 是视频处理器。当需要对视频回放，就必须安装一个视频处理器，FFMPEG 依赖于 gst-ffmpeg | 0.5 or higher FFmpeg |
| GStreamer | Gstreamer 是视频处理器。当需要对视频回放，就必须安装一个视频处理器 | 0.10 or higher gstreamer |
| gst-ffmpeg | gst-ffmpeg 允许你通过 Gstreamer 使用 FFMPEG 解码器。如果您想使用 Gstreamer 作为视频处理程序，这个软件包是需要 | gstreamer0.8-ffmpeg-dev |

由于视频处理器 GStreamer 和 FFMPEG 不能同时存在，我们只能选择其一个作为我们的视频处理器，在这里我们选用了 Gstreamer(须先安装 glib)作为我们的视频处理器，编译过程如下：

```
./configure
--prefix=/opt/crosstool/gcc-3.4.2-glibc-2.3.3/arm-linux-gnu/arm-linux-gnu/
--host=arm-linux-gnu
make;make install;
```

因为我们选用了 GStreamer 作为我们的视频处理器，为了保证它能正确运行，我们还必须加入与其相关的依赖包 gstreamer0.8-ffmpeg-dev。

(2) 音频的处理器编译：在 Gnash 中，我们使用 SDL 作为音频的处理器，我们所需编译的库和说明如下表所示：

表 3　Gnash 的音频处理器

| | | |
|---|---|---|
| SDL | 跨平台的多媒体开发函数库，为音频，图形，声音和输入接口提供抽象对象。Gnash 需要安装至少有一个图形用户界面库，SDL 被认为是对 Gnash 支持最差的 GUI 库，但是对于 Gnash 来说 SDL 中声音的处理是最好的 | 1.2.13 or higher sdl |

在上表中，我们看到其实 SDL 是一个多媒体开发

函数库，它包含了众多接口，特别是声音处理器部份，它对 Gnash 的支持是最好的，我们取优汰劣，保留 SDL 的声音处理器部份，SDL 的编译过程如下：

./configure
--prefix=/opt/crosstool/gcc-3.4.2-glibc-2.3.3/arm-linux-gnu/arm-linux-gnu/
--host=arm-linux-gnu?--enable-shared
--enable-video-fbcon
--disable-video-x11 --disable-video-photon
--disable-video-directfb
--disable-video-ggi--disable-video-svga
--disable-video-aalib --disable-video-qtopia
--disable-video-dummy --disable-joystick
--disable-alsa --disable-esd --disable-arts
--disable-nasm --disable-dga

make;make install;

(3) 渲染引擎的编译：我们再来看一下渲染引擎，在 Gnash 中有三种渲染引擎可供我们选择，如下表所示：

表 4　Gnash 的渲染引擎

| | | |
|---|---|---|
| AGG | AGG是AntiGrain低级别的2D图形库。而AGG被认为对Gnash支持性最好的Render | 2.5 or higher agg |
| OpenGL | OpenGL是一个标准的规范，定义了一个跨语言跨平台的API编写应用程序，产生3D和2D图形。它支持硬件加速。 | libgl1-mesa-dev |
| Cairo | Cairo是一个2D图形库，支持多种输出设备。它会自动使用显卡加速，并有一个实验的OpenGL backend 接口。而Cairo对Gnash的支持性最差的 | libcairo2-dev |

在这 3 种渲染引擎中，OpenGL 和 Cairo 虽然功能强大但需要图形加速卡的支持，这并不太符合嵌入式的工作环境的需求，根据大部份嵌入式设备配置低的特点，所以我们只能选择 AGG 作为我们在嵌入式环境上工作的渲染引擎，在这我们可以把对 OpenGL 和 Cairo 的支持从 Gnash 中裁剪掉，只保留 AGG。AGG(须先安装 SDL)的编译过程如下：

修改 Makefile.in.linux.SDL

AGGLIBS= -lagg -lSDL

AGGCXXFLAGS = -O3 -I/opt/crosstool/gcc-3.4.2-glibc-2.3.3/arm-linux-gnu/arm-linux-gnu/include/SDL

-L/opt/crosstool/gcc-3.4.2-glibc-2.3.3/arm-linux-gnu/arm-linux-gnu/lib

CXX = arm-linux-gnu-g++

C = arm-linux-gnu-gcc

#CXX = icc

LIB = arm-linux-gnu-ar cr

.PHONY : clean

Make

copy agg-2.5/include

copy agg-2.5/src/libagg.a

## 5　总结

本文对 Gnash 系统的组成及结构进行了分析，给出了其功能模块及其依赖关系，为嵌入式移植时的模块裁剪和重组打下了基础。重点讨论了将 Gnash 移植到基于 ARM 的嵌入式 Linux 平台时的交叉编译过程，由于篇幅所限没有深入讨论有关系统裁剪的技术细节。作者已在 64M 内存，版本为 2.4.20 linux 内核的 Intel XSCALE PXA270 平台上基于 Gnash 实现了 flash 动画和视频文件的播放，取得了满意的效果。

### 参考文献


1 孙晓辉.一种嵌入式 Flash 播放器的设计与实现.计算机应用, 2008,28(1):248－249.

2 杨宗凯,梁志聪.嵌入式 Flash 播放器的设计与实现.计算机工程与科学, 2007,29(6):128－131.

3 杨宗凯.嵌入式 Flash 播放器的设计与实现.计算机工程与科学, 2007,29(6):128－131.

4 Gnash Free Software Foundation (FSF). [2007- 07-02]. http://www.gnu.org/ software/gnash

5 Gnash Project. [2008-10-28]. http://www.gna- shdev.org/

6 GNU Libtool.[2008-02-19]. http://developer. apple.com/documentation/DeveloperTools/glibtool/libtool_15.html#SEC86

7 Gnash Documentation.[2008-10-28]. http:// www. gnashdev. org/ doc/apidoc /index.html

8 Gnash Building. [2008-10-28]. http://wiki. gnashdev.org/Gnash#Building