网络播放器需求说明

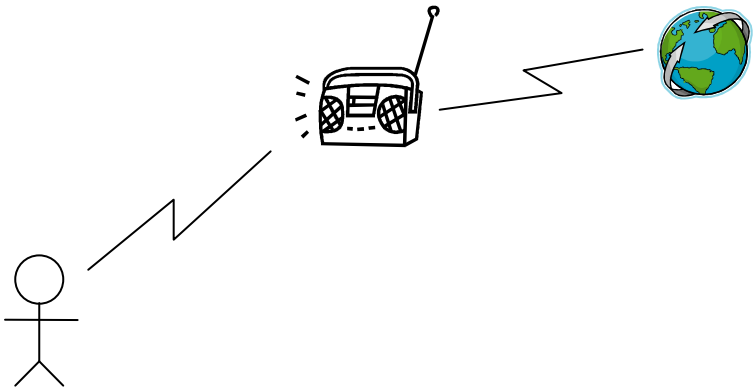
说明：*对于一个软件系统的设计，最开始的步骤就是需求说明，所以网络播放器的开篇还是从最正规的设计方法开始。另外，所有的设计文档都是从实用角度出发，所以在文档格式上不做过多规范，或许最后会有一个总结格式。*

嵌入式系统是一个完整的系统，包含了软件、硬件两个方面。嵌入式系统之软件系统需求说明是从完整的系统需求中截取和软件相关的部分。当然有的时候也会存在，需要的部分功能应该采用软件还是硬件方式来实现的抉择。例如，STM32 网络播放器中，mp3 部分的解码部分，到底是采用硬解码还是软解码。

在通常的实施过程中，系统的需求说明还会先包括一部分系统可行性研究工作，以避免设计虽然做出来了，但到最后却不可能实现（也就意味着产品失败），例如这款网络播放器项目在前期就做了 STM32 的 mp3 软解压可行性分析。

网络播放器的上下文环境

说明：*一个产品的设计离不开产品所处的环境，只有目的明确了，才能保证后续产品的实现能够满足一些先决条件。*

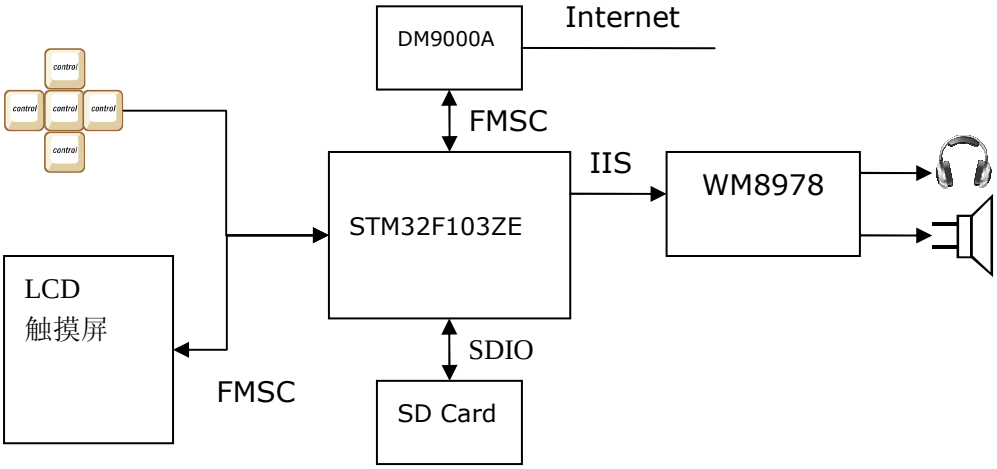


如上图所示的网络播放器操作环境，

1、 网络播放器通过网线连接到路由器，通过路由器连接到互联网中。
2、 网络播放器采用 5V 电源供电（能够兼容从电脑 USB 取电）。
3、 用户可通过网络播放器收听网络上的音频流（当前互联网音频流格式主要包括三种：主要传递 wma 音频的 mms 协议；主要传递 mp3、ogg 音频的 shoutcast 协议；主要传递 real audio 的 RSTP 协议）。
4、 用户可通过网络播放器播放用户提供的 SD 卡上的数据。
5、 用户可通过网络播放器上的按键和触摸屏进行网络播放器的操作。
6、 用户可通过网络播放器自带的喇叭或耳机收听网络电台；

系统硬件框图

软件的设计离不开硬件的实现，所以在设计时很有必要知道硬件框图是什么样的。



软件相关的需求说明

1、 网络播放器通过网线连接到路由器，通过路由器连接到互联网中。
网络播放器支持 DHCP 方式从路由器获取 IP v4 地址。

2、网络播放器采用 5V 电源供电（能够兼容从电脑 USB 取电）。
3、用户可通过网络播放器收听网络上的音频流：
用户可播放网络上的 mp3、wma 音频，能够支持 http 音频流和 mms 音频流协议。为了更好地支持网络音频流的播放，网络播放器支持音频流的缓冲播放。

4、用户可通过网络播放器播放用户提供的 SD 卡上的数据。
用户可播放 SD 卡上的 mp3、wav、wma 歌曲。当用户插上 USB 线后，用户通过电脑操作能够操作 SD 卡上的文件（做为 U 盘操作文件）。

5、用户可通过网络播放器上的按钮和触摸屏进行网络播放器的操作。
网络播放器能够接收按键进行播放器的操作，进行下一电台，上一电台，播放开始，播放停止，声音增大，声音减小的操作。

网络播放器能够通过自带的液晶显示屏给出播放的状态。用户能够使用自带触摸屏对网络播放进行操作：选择相应的功能、播放 SD 卡上的音频文件，播放互联网上的音频流。

用户可从网络中更新网络电台的列表并存放到播放器中。

6、 用户可通过网络播放器自带的喇叭或耳机收听网络电台；

【注：当前版本未包括 wma 软解码实现】

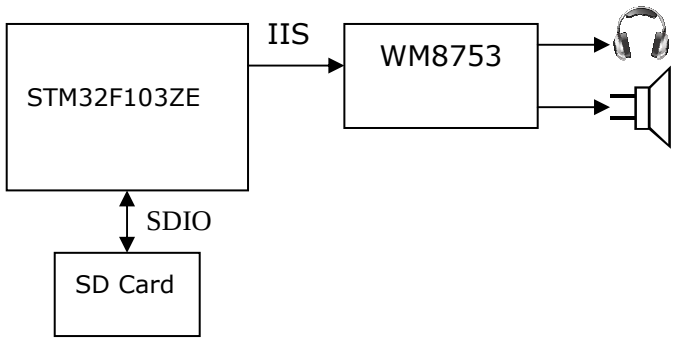
可行性分析

在实施项目前，必要的可行性分析是不可缺少的，它将决定着一个项目的成败，也是项目（产品）开展的前期步骤。

网络收音机项目因为采用的 mp3 软解码，并且后续还需要加入网络音频流播放的功能，图形用户界面的功能等，其中最为关键的就是 mp3 软解码。Mp3 软解码有几个影响非常大的地方：

- 软解码的性能，当进行软解压时，还剩余多少空闲时间给其他线程。
- 软解码的内存占用，还得计算进读取文件时的缓冲，解码出来的 PCM 数据存放用的缓冲等。
- 因为使用的主控芯片是 STM32，还很有必要了解采用板载外扩 SRAM 访问时速度情况如何，如果速度和片内 SRAM 相差不大，那么内存方面的制约将相差不大。

前期的可行性分析是基于一块已有 STM32 开发板进行的，仅自行搭建了一个 WM8753 的 codec 开发板，即仅包括如下的框图：



⌦ 经过实际测试发现，当进行 mp3 软解码时，mp3 码率是 64kpbs 时，系统的占用大约是 50%，mp3 码率是 320kbps 时，系统的占用大约是 70%。在网络中，大多数网络音频流的码率是 64kbps – 128kbps，基本上 CPU 的运算性能满足条件。

⌦ 内存占用上，如果采用 libmad 整数软解码器，它的内存占用量会需求很多，远远超过片内的 64k SRAM，必然将使用板载外扩 SRAM。如果采用 helix 的 mp3 整数软解码器，内存占用是 23kB，再加上解码后的 PCM 数据缓冲，大约在 30kB 左右。

⌦ 板载 SRAM 和片内 SRAM 的测试数据如下：

当把数据段放在片内 SRAM 时的测试结果如下：（分别对内存块做 8 位、16 位、32 位访问及采用库函数的 memset 访问结果）

```
finsh>>benchmark()
internal sram[8bit]: 326
external sram[8bit]: 466
internal sram[16bit]: 210
external sram[16bit]: 233
internal sram[32bit]: 105
```

```
external sram[32bit]: 163
internal sram[memset]: 27
external sram[memset]: 163
```

从上面的数据可以看得出来，如果是单独的 8 位、16 位、32 位数据访问，性能有些差别，基本上板载的外扩 SRAM 要比片内的 SRAM 慢：1.42、1.10、1.55 倍，但是当使用系统的 memset 时，访问速度要慢：6 倍。

实际测试下来，如果把 Helix 的 MP3 软解码程序使用板载外扩 SRAM，速度会明显变慢，声音不连续。

网络播放器体系结构设计

说明：*体系结构设计是需求说明后，软件系统划分的第一步。它会把一个潘多拉黑盒打开，当然一般是大黑盒套小黑盒，也即把系统这个最大的潘多拉黑盒打开，按照它的功能特点进行细一些的模块划分，每个模块在这个阶段依然是一个小的潘多拉黑盒。*

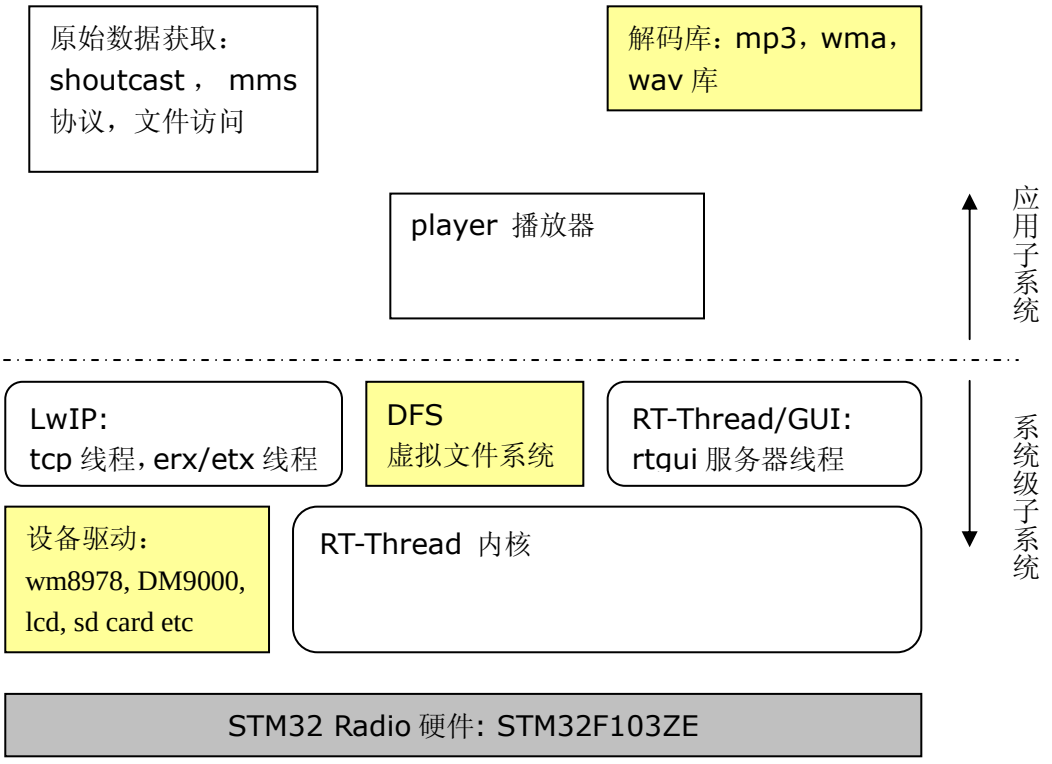
如何确定模块？一般的原则是：

- ✓ *由一些用例直接派生出子系统，例如本例中的音频播放，派生音频解码库（包括 mp3 软解码、wma 软解码等）。*
- ✓ *在功能上或问题域上把一些相对独立的、大粒度的功能模块划分为模块，例如本例中的网络协议栈。*
- ✓ *体系结构设计是为了在团队之间合作开发，所以在定义模块时也需要考虑到模块的规模，避免定义得过大，导致一个团队不能够完成，或完成的时间比较慢，最后导致产品的延时。*

在确定了模块的基础上再明确各模块的职责，定义相关职责接口，需要保证模块与模块间功能不相重叠。另外，模块与模块间的依赖关系也非常重要，需要对每一个用例实例化，做到不会丢失前期的需求。

系统分解

网络播放器组件分解如下图所示（一些小型组件，例如键盘未给出）



系统级模块：
系统模块部分尽量能够采用一些成熟的平台，以避免当问题出现时，不知道到底是上层的问题还是应用的问题。RT-Thread，一套国内主导开发的开源实时系统，它已被数家国内公司所采用，并且还提供相配套的附属组件（而这些也是这个网络播放器需要用到的），选择 RT-Thread 组件有一定风险，也有一些非常有利的地方。

RTOS Kernel：使用国内开源实时操作系统 RT-Thread。RT-Thread 做为一套完善、稳定的实时操作系统，对于这个应用完全能够适用。通过使用 RT-Thread 来作为网络收音机的操作系统，也能够了解、学习 RT-Thread 实时操作系统。

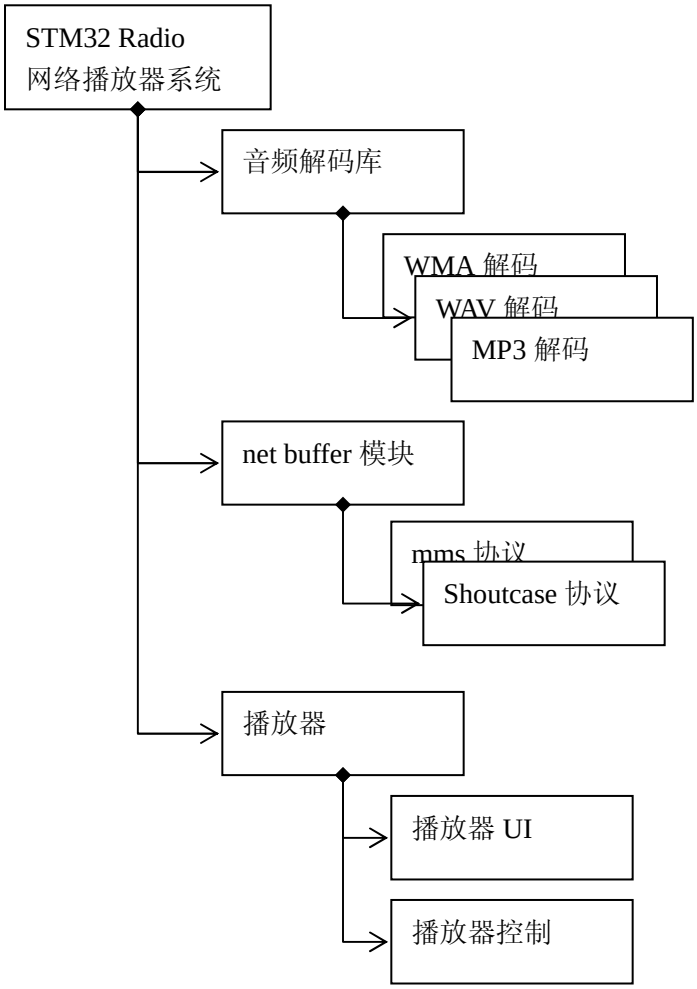
文件系统：使用 RT-Thread 内置的 DFS 虚拟文件系统，目前这个 DFS 虚拟文件系统包装的是 ELM FatFS。采用虚拟文件系统的好就是，底层具体的文件系统实现替换掉，但上层的应用可以保持不变（在 STM32 网络收音机的开发过程中就出现过用 ELM FatFS 替换 EFSL 文件系统的例子，而上层应用代码没动过任何一行），而且 DFS 向上提供的接口是 POSIX 兼容的，这对代码的可移植性更有好处。

TCP/IP 协议栈：LwIP 轻型 TCP/IP 协议栈，与 RT-Thread 的整合也几乎是无缝的，当前的 RT-Thread/STM32 也已经支持 ENC28J60 接口的驱动，只需要把它改成 DM9000A 即可。LwIP 协议栈对应用层提供了标准的 socket 接口，并且 LwIP 自身支持多种协议，例如 ICMP，IGMP，DHCP，DNS，PPP 等，有 DNS、DHCP 协议也能够直接应用到这个网络收音机中，这样可以不用手动的设置 IP 地址。

图形用户界面：RT-Thread/GUI，这个是 RT-Thread 开发的多窗口多线程图形用户界面，应用于 STM32 网络收音机也是非常好的。网络收音机项目包含一个 320x240 的 TFT 屏，5 向导航键，触摸屏，采用键盘与触摸相结合的操作方式。

USB Mass Storage：U 盘，采用 STM32 USB 固件库来实现一个完整的 U 盘，使得能够直接从电脑上操作播放器上的文件。在通过 U 盘操作文件时，系统自动停止播放，关闭打开的文件，不再操作文件，当 USB 连接断开时，系统自动复位以恢复到初始状态。

应用部分模块
其中应用部分的模块可进一部分分解成如下表示的组成图：



音频解码库（codec)

从输入的数据中进行软件解码，给出相应的 PCM 数据；另外音频数据一般会携带 TAG 信息，所以音频解码库也需要提供相应的 TAG 信息，播放进度等信息。

音频解码库主要包括三个解码：MP3 解码，WAV 解码（除掉文件头以后也就是 PCM 数据），WMA 解码。

网络缓冲区管理（net buffer)

这个子系统用于取得原始的音频数据，即未解码前的原始音频数据。这部分包括网络方式获取数据（包括 ShoutCase 协议方式和 MMS 协议方式）。

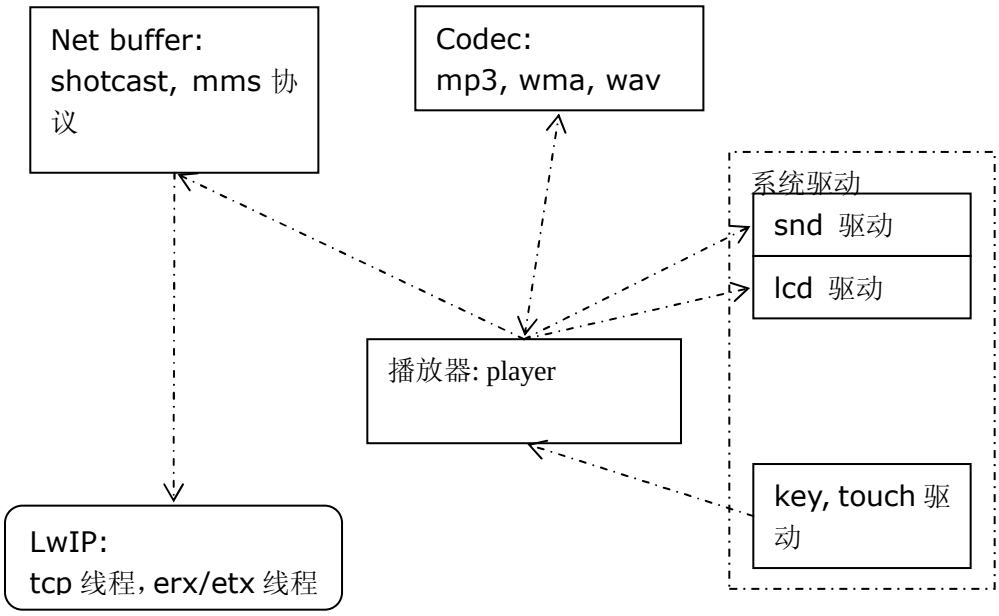
播放器（player)

播放器部分用于控制播放的过程，例如开始播放，停止播放，以及响应用户的操作请求。这部分按照它的功能，又细分为两个部分：

播放器 UI，即播放器的界面，用于获取用户的输入请求。为了满足一些操作上的便利，它还将提供一些额外的功能。

播放器控制，这个是音频播放的主控部分。

各模块间的依赖关系



此处主要给出了播放器核心模块间的依赖关系，而对于 RTOS 底层服务间的关系没给出（因为基本都会有关联）。

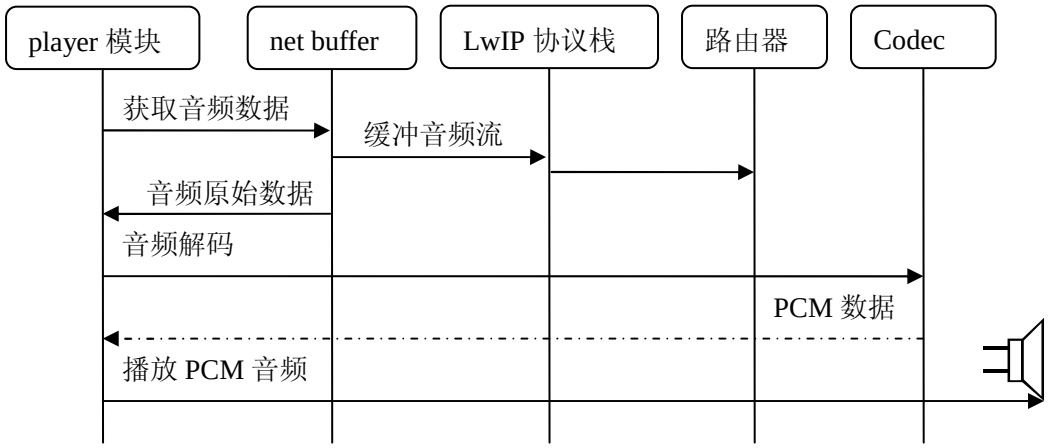
播放器模块需通过 net buffer 模块或文件系统模块获得音频的原始数据；
播放器模块需要传递数据报文给 Codec 模块，Codec 模块转换成 PCM 数据后再发回数据给播放器模块。
播放器模块在系统中需要使用 snd 驱动进行音频的播放，lcd 驱动显示
net buffer 模块在获取网络音频流时需要使用 LwIP 协议栈获得网络报文；

用例需求在设计中的反映

1、网络播放器通过网线连接到路由器，通过路由器连接到互联网中。
网络播放器支持 DHCP 方式从路由器获取 IP v4 地址。

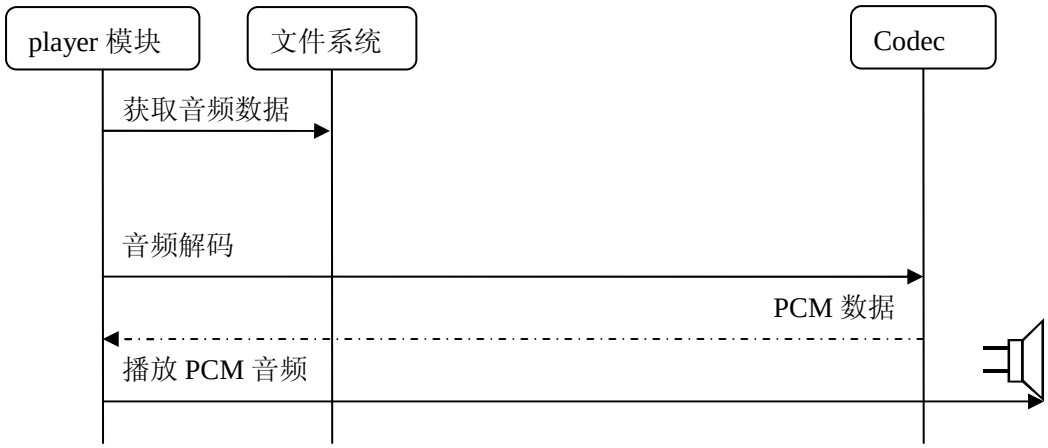
2、网络播放器采用 5V 电源供电（能够兼容从电脑 USB 取电）。
3、用户可通过网络播放器收听网络上的音频流：

用户可播放网络上的 mp3、wma 音频，能够支持 http 音频流和 mms 音频流协议。为了更好地支持网络音频流的播放，网络播放器支持音频流的缓冲播放。



4、用户可通过网络播放器播放用户提供的 SD 卡上的数据。

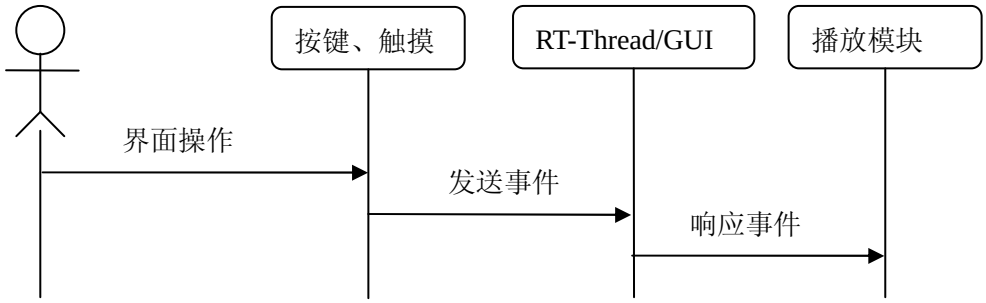
用户可播放 SD 卡上的 mp3、wav、wma 歌曲。当用户插上 USB 线后，用户通过电脑操作能够操作 SD 卡上的文件（做为 U 盘操作文件）。



5、用户可通过网络播放器上的按钮和触摸屏进行网络播放器的操作。

网络播放器能够接收按键进行播放器的操作，进行下一电台，上一电台，播放开始，播放停止，声音增大，声音减小的操作。

网络播放器能够通过自带的液晶显示屏给出播放的状态。用户能够使用自带触摸屏对网络播放进行操作：选择相应的功能、播放 SD 卡上的音频文件，播放互联网上的音频流。



7、 用户可通过网络播放器自带的喇叭或耳机收听网络电台；

网络播放器模块设计

说明:

模块设计在整个设计过程中相当于把一个个小的潘多拉黑盒打开，清晰化。这个过程注重于模块相互之间的接口，模块内部的清晰条例性，能够真正的达到相互之间的并行开发。

但是嵌入式软件系统的设计和通常意义的软件系统设计是不相同的。在通用系统中，通常考虑的是如何顺序的完成一个任务，所以大多数关注点在于如何完成这个任务，如何把一个复杂的任务进行层层分解，由繁到简的划分模块，实现模块。

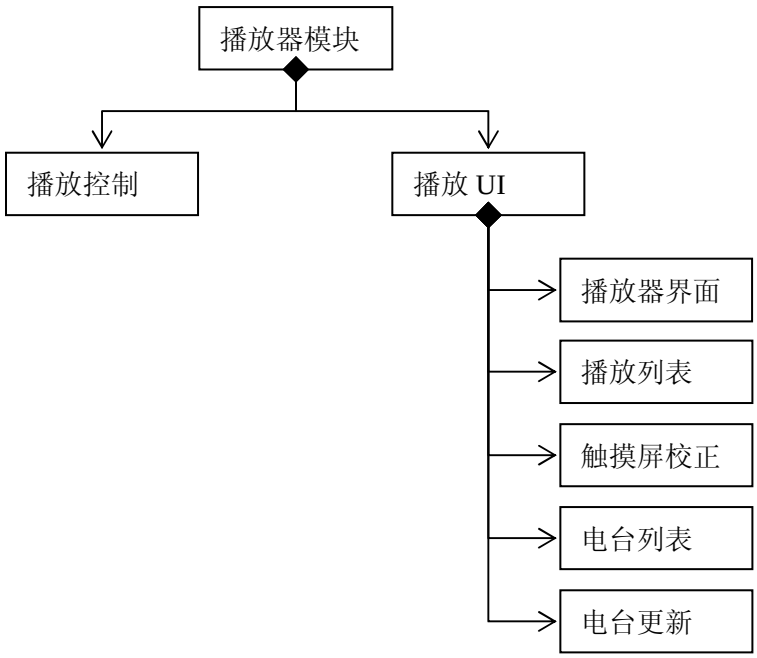
在嵌入式系统上，当引入了实时操作系统后，编程模式需要转换成并行任务的方式解决问题，即包括了线程、任务的模式。所以在进行嵌入式软件系统设计时，不仅需要考虑通常意义上的对象划分，对象与对象之间的关系，更需要考虑系统的动态行为：线程的划分，线程的状态图跃迁（如果复杂的话），线程间的消息序列等。

在下面的设计中，按照上面的原则，将从静态和动态的角度来分析系统模块的实现。

Player 模块设计

播放器模块是网络收音机中的核心模块，相对来说，这个模块功能涉及比较多，所以在目前的基础上把它进行拆分形成：播放控制模块和播放 UI 模块两个模块。

静态视图



播放 UI 模块主要涉及到界面相关的操作，主要又分为：

- 播放器界面
  这个界面中将实现歌曲的播放、停止；下一首，上一首；音量调节

- 播放列表
  用于管理当前播放歌曲的列表

- 触摸屏校正
  用于对触碰屏进行校正

- 电台列表
  从文件中读取存储的电台列表，提供给用户进行选择。

- 电台更新
  从互联网更新电台列表。（网址是：http://radio.rt-thread.org）

播放控制模块进行实际的播放操作，在播放的过程中由它调用 Codec 库进行相应的音频软解码。

这个模块提供的接口如下
播放控制模块

名称	描述
player_play_req	请求播放指定的文件名。
player_radio_req	请求播放指定服务地址的电台。
player_stop_req	请求停止播放。
player_is_playing	返回当前处于停止或播放状态。

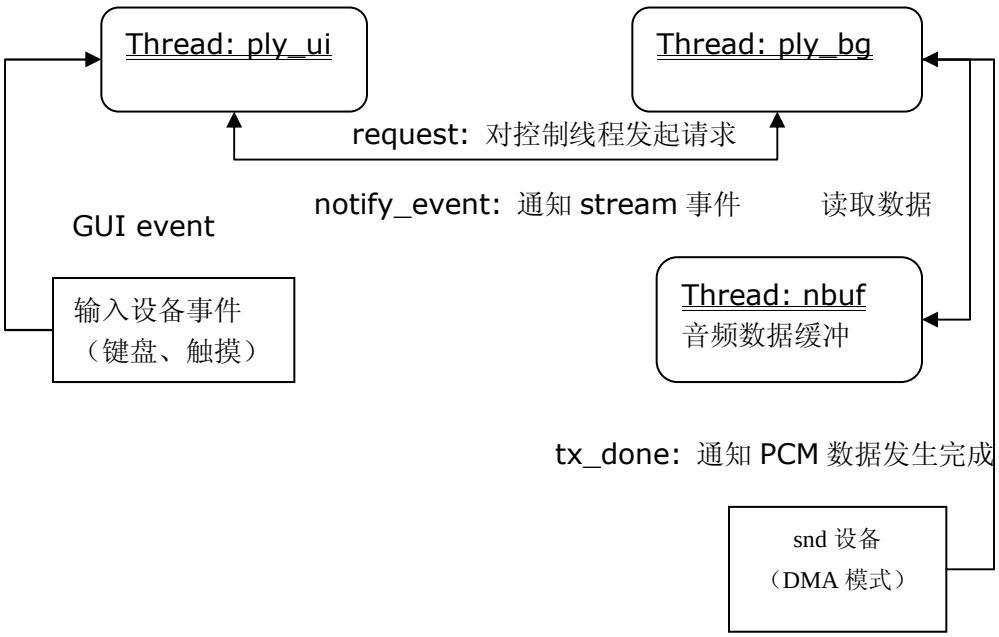
播放 UI 模块接口

名称	描述
player_ui_freeze	冻结播放界面<用于 USB 联机时>
player_notify_play	指示 UI 已经开始播放
player_notify_stop	指示 UI 播放已经停止
player_notify_info	指示 UI 当前正在播放歌曲的信息
player_notify_functionview	请求 UI 进入功能菜单
player_set_position	设置当前播放歌曲的正在播放的位置
player_set_title	设置当前播放歌曲的标题
player_set_buffer_status	设置播放器是否处于缓冲状态

线程视图

播放器控制线程：线程名 ply_bg (player background)
播放器控制线程接收来自用户的命令，进行播放文件或播放音频流操作。播放器在接收到播放命令时，它将先识别出是文件或流，如果是文件，将直接调用文件系统的接口进行文件访问（不再做专门的音频缓冲）。如果是网络音频流将从 net buffer 模块获得内容。

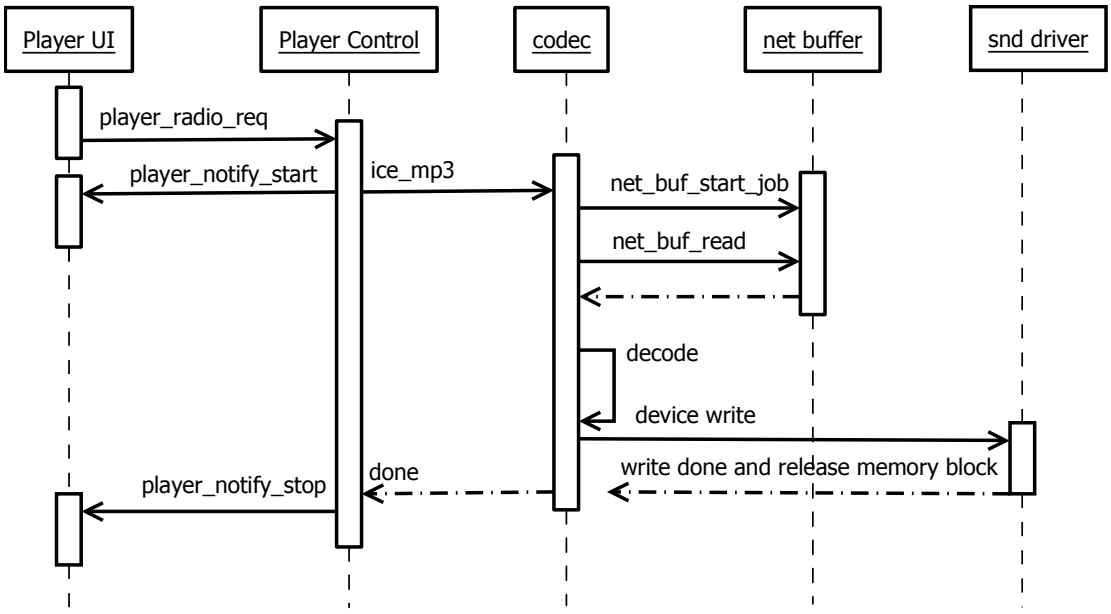
播放器 UI 线程：线程名 ply_ui (player UI)
播放器 UI 线程是根据 RT-Thread/GUI 特点来设计的，因为 RT-Thread/GUI 的一个应用需要一个独立的上下文环境。从通常的 UI、数据分离的角度出发，也需要把播放器 UI 和播放器控制分离开。播放器 UI 线程将实现所有绘图显示操作，并且能够接收 UI 上的人机交互事件，例如按键和触摸屏事件。



线程交互的主要是 ply_ui 和 ply_bg 相互间的请求和通知。而值得注意的是 ply_bg 与 nbuf 线程间的行为。

用例实现视图

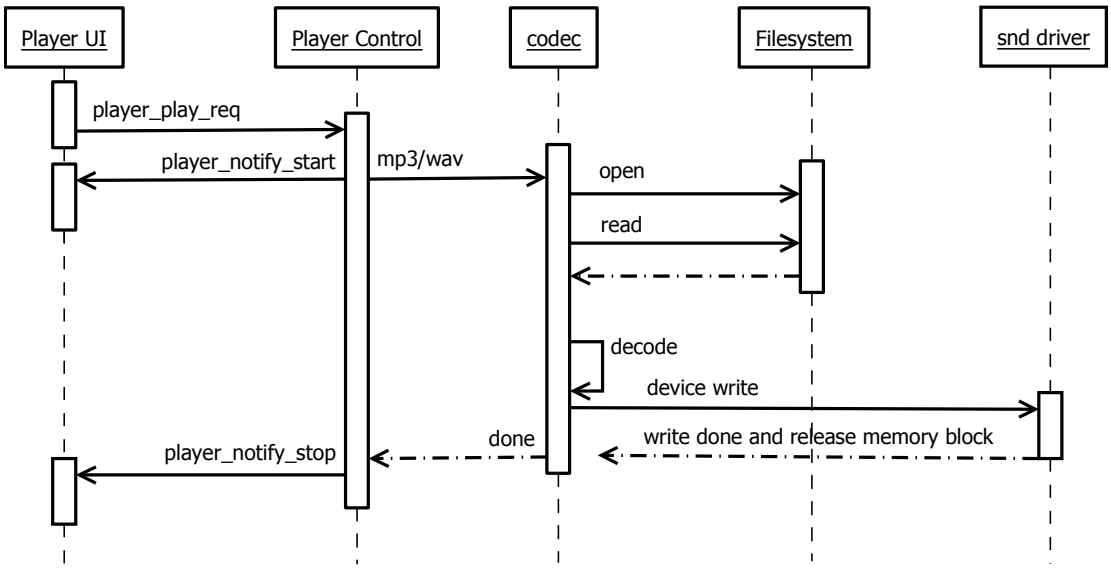
3、用户可通过网络播放器收听网络上的音频流：



播放电台时的模块间消息序列图

Player UI 通过 Player Control 提供的接口：player_radio_req 来发起一条播放请求，其中包含了电台网址和电台名称；Player Control 模块在验证请求无误后，先调用 Player UI 模块提供的接口 player_notify_start 通知 Player UI 播放已经开始。接下来，Player Control 模块根据请求的情况调用 Codec 中的 ice_mp3 接口启动播放。Codec 模块先采用 shoutcast 协议打开相应的 TCP 连接，然后向 net buffer 模块请求开始一项缓冲任务，同时发起一条读取数据请求。当 net buffer 缓冲数据达到一定程度时，它将返回 Codec 预读取的数据。Codec 拿到相应的数据后开始解码获得 PCM 数据写入到 snd 驱动中进行播放。

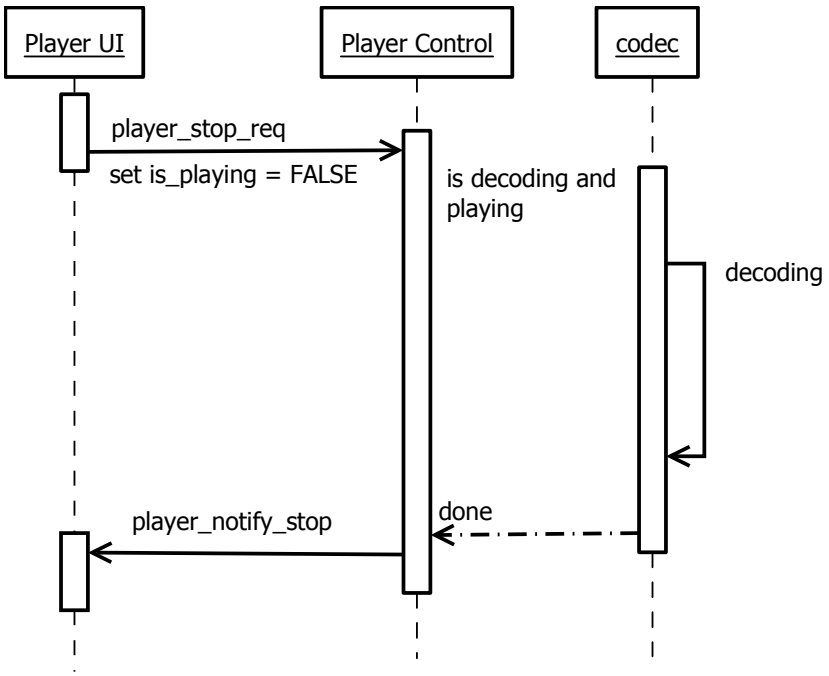
4、用户可通过网络播放器播放用户提供的 SD 卡上的数据。



播放 MP3 或 WAV 文件时模块与模块间的消息序列图

Player UI 通过 Player Control 提供的 player_play_req 接口来发起一条播放请求；Player Control 模块在验证请求无误后，先调用 Player UI 模块提供的接口 player_notify_start 通知 Player UI 播放已经开始。接下来，Player Control 模块根据请求的情况调用 Codec 中的 MP3 播放或 WAV 播放。Codec 模块则根据请求的文件，打开文件并读取文件，然后解码，最终获得 PCM 数据写入到 snd 驱动中。

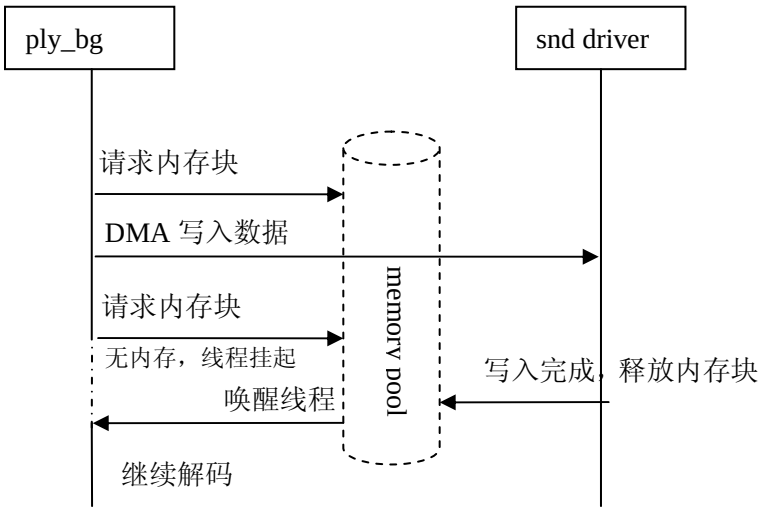
补充的停止播放用例（3、4 用例都包含这个过程）



停止播放的模块间消息序列图

Player UI 通过调用 Player Control 提供的 player_stop_req 接口通知一次播放过程结束。Player Control 此时应该是一直解码播放的过程，当它得到播放停止的状态时，将停止余下的软解码播放过程,并调用 Player UI 提供的 player_notify_stop 接口通知 Player UI 模块，这次播放过程已经结束。当播放结束时，Player UI 可以抉择是否进行下一步行动，例如播放下一首歌曲。

播放时的用例实例化补充



ply_bg 线程当要解码 PCM 数据时，将向 memory pool 请求内存块，如果内存块已经用完，将被挂起。否则，它将获得内存块以放置 PCM 数据。

而后 ply_bg 线程把包含 PCM 数据的内存块写入到 snd 抽象设备中。(线程会立刻返回，

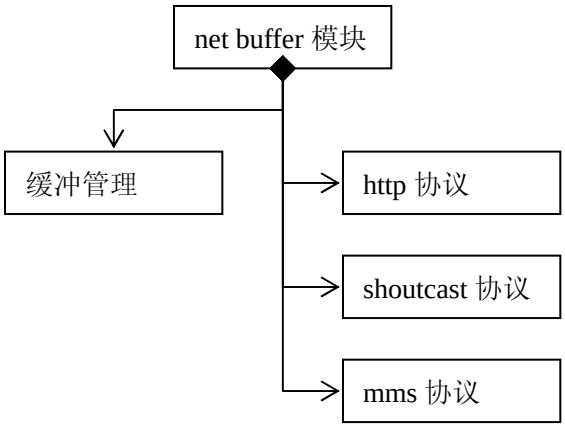
继续制造解码出更多的 PCM 数据)

当 snd 抽象设备写入完成后，将调用 ply_bg 设置的回调函数，释放写入的内存块。如果此时，播放线程因为把 memory pool 里的内存块都用光而挂起时，将唤醒播放线程，让播放线程继续软解码更多的 PCM。

Net buffer 模块设计

网络缓冲管理模块主要用于获取网络上的音频流数据并把它放置到自己的缓冲区中进行管理，网络电台多数采用 64kbps- 128kbps 的码率进行音频流传输。按照 128kbps，缓存 20 秒需要用到 320kB 的内存量。

静态视图



网络缓冲管理模块的分解图

这个模块主要包括两个部分，一个是缓冲区管理，用于把数据进行缓冲。
缓冲区管理的接口如下：

名称	描述
net_buf_read	从 net buffer 中读取指定长度的数据。
net_buf_start_job	请求 net buffer 开始一项工作。
net_buf_stop_job	请求 net buffer 停止一项工作。
net_buf_get_usage	获得 net buffer 中已获得的数据缓冲长度；

一个是一些网络协议的实现，包括 http 协议，shoutcast 协议和 mms 协议。shoutcast 协议是基于 http 协议添加自己固定的信息。http 协议并不是网络收音机需要支持的格式。为了便于管理几种协议实例，定义如下接口：
http 协议

名称	描述
http_session_open	根据 URL 打开相应的协议会话，成功返回相应句柄，否则 NULL
http_session_read	从协议句柄中读取指定的数据
http_session_seek	对本次会话向前跳过（忽略）指定的数据字节长度

http_session_close	关闭本次协议会话。

shoutcast 协议

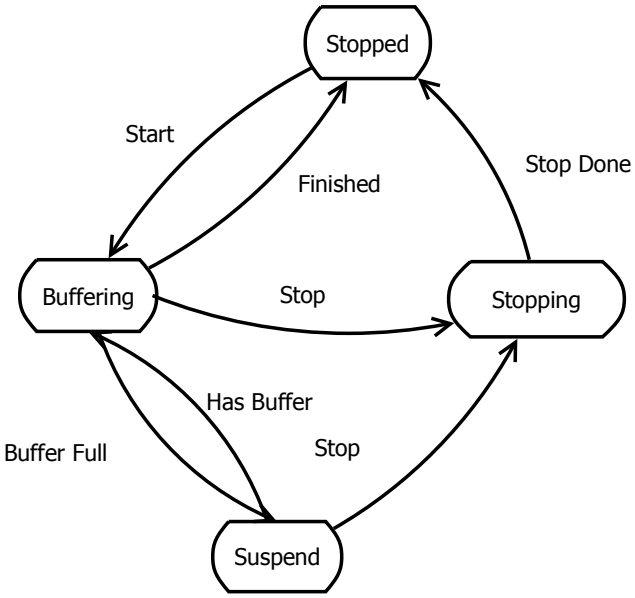
名称	描述
shoutcast_session_open	根据 URL 打开相应的协议会话，成功返回相应句柄，否则 NULL
shoutcast_session_read	从协议句柄中读取指定的数据
shoutcast_session_seek	对本次会话向前跳过（忽略）指定的数据字节长度
shoutcast_session_close	关闭本次协议会话。

mms 协议

名称	描述
mms_session_open	根据 URL 打开相应的协议会话，成功返回相应句柄，否则 NULL
mms_session_read	从协议句柄中读取指定的数据
mms_session_seek	对本次会话向前跳过（忽略）指定的数据字节长度
mms_session_close	关闭本次协议会话。

线程视图

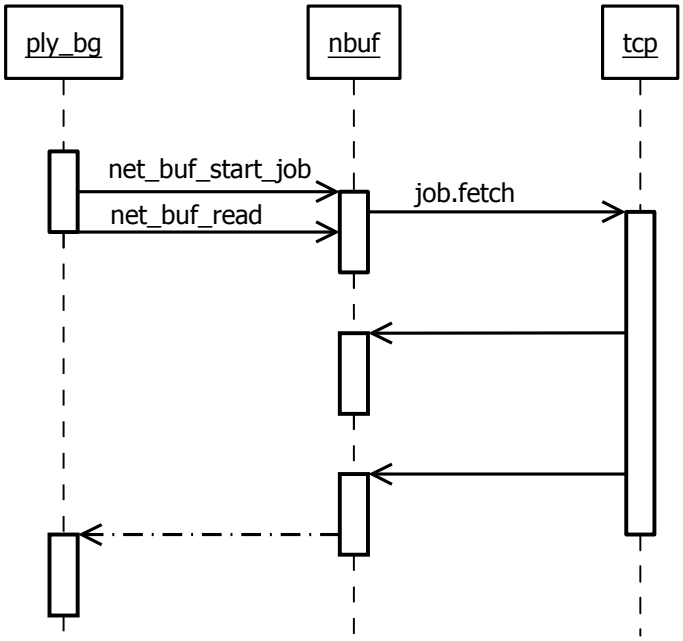
这个模块提供一个线程用于获取网络数据：线程名 nbuf。线程的状态转换图如下图



状态描述

名称	描述
stopped	停止状态，nbuf 线程等待在自己的消息队列中以接收新的命令。
buffering	缓冲状态，正在从网络中读取数据进行缓冲。
suspend	挂起状态，保存数据达到缓存的最大空间，线程处于挂起状态等待空间释放。
stopping	预停止状态，停止请求已经发送，但线程还未达到停止状态。

这个线程与其他模块间的消息序列图

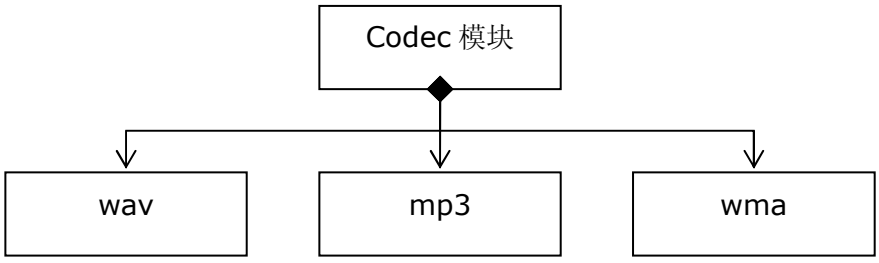


ply_bg 线程调用 net buffer 提供的 net_buf_start_job 接口以开始一个缓冲工作，此时 nbuf 线程将唤醒（假设它先处于 stopped 状态）。nbuf 线程唤醒后开始调用 job 的 fetch 接口向 tcp 线程获取数据报文。ply_bg 线程在发送完开始工作后，它将紧接着调用 net_buf_read 接口读取网络数据，但此时 net buffer 还是 buffering 状态，ply_bg 线程将被挂起。当 nbuf 线程获得足够的网络数据时，它将唤醒 ply_bg 线程并给它足够的数据用于播放。当 nbuf 线程读取到的网络数据把缓冲区都填满时，它将挂起以等待缓冲区的数据读取掉释放出空间出来。当 ply_bg 线程读出数据，缓冲去遗留的剩余数据小于一个数值时，它将唤醒 nbuf 线程继续获取接下来的网络数据。

Codec 模块设计

codec 模块提供对 wav 数据，mp3 数据和 wma 数据的解码操作。除了解码外，它也需要包括提供一些文件格式信息的分析。

静态视图



- wav 格式，未压缩的 PCM 数据，但有文件头信息；
- mp3 格式，压缩格式，包含 ID3Tag 信息；
- wma 格式，压缩格式

相应的接口包括：

名称	描述
wav	根据提供的文件名播放 wav 文件。
mp3	根据提供的文件名播放 mp3 文件。
ice_mp3	根据提供的电台地址和电台名播放 shoutcast 电台。
http_mp3	根据提供的地址播放 http 方式的 mp3 流。
wma	根据提供的文件名播放 wma 文件。
mms_wma	更加提供的电台地址和电台名播放 mms 电台。

线程视图

这个模块不提供独立的线程。

网络播放器详细设计

说明:

详细设计是一个非常底层的环节，通常来说，这也是一个“各显神通”的环节：和实现人员素质密切相关的环节：

在实现中，或许由于实现人员水平参差不齐，里面隐藏的 BUG 数目也不尽相同，当完成后还能够依据一些其他手段进行修补，例如开发人员间的代码评审，单元测试，集成测试等。

模块详细说明

播放控制模块：player_bg.c

接口详细说明

名称	描述
player_play_req	void player_play_req(const char* fn); 请求播放指定的文件名，参数 fn 为播放的文件名，给出的应该是绝对路径。 在这个函数中，将发送类型为： PLAYER_REQUEST_PLAY_SINGLE_FILE 的消息给 ply_bg 线程。
player_radio_req	void player_radio_req(const char* fn, const char* station); 请求播放指定服务地址的电台，参数 fn 为播放电台的服务器地址，参数 station 为电台名称，用于 UI 后续的 UI 显示。 在这个函数中，将发送类型为： PLAYER_REQUEST_PLAY_SINGLE_FILE 的消息给 ply_bg 线程。 这个消息和 player_play_req 不同的是，它也会填充 station 字段。
player_stop_req	void player_stop_req(); 请求停止播放。 在这个函数中，会设置变量 is_playing 为 RT_FALSE；这个变量将在 ply_bg 线程解码一帧后检查一次。
player_is_playing	rt_bool_t player_is_playing(); 返回当前处于停止或播放状态，如果是播放状态返回 RT_TRUE，如果是停止状态返回 RT_FALSE;

其他说明：

void player_init(void);

播放器模块的初始化函数，会创建 ply_bg 线程及初始化 ply_ui。

void player_thread(void* parameter);
ply_bg 线程入口函数。ply_bg 线程启动后主要等待在消息队列(player_thread_mq)上，当收到消息时（当前仅对 PLAYER_REQUEST_PLAY_SINGLE_FILE 消息进行处理），将根据消息中的 fn 成员判断是 wav、mp3 还是电台（http 开头）。

播放 UI 模块：player_ui.c, calibration.c, play_list.c, station_list.c, radio_list_update.c

player_ui 包含了播放器的主界面
通常 player_ui，即播放器 workbench，包括了两个视图：
home 视图，用于显示播放器主界面

function 视图，用于显示功能菜单（可以通过在播放器主界面点击触摸屏的 RT-Thread LOGO 部位或不播放时按右方向键盘进入功能视图）
功能列表定义在 function_list 数组变量中，采用的是一个 rtgui_list_view 的形式，当前没添加上图标的支持，能做的扩展是加入图标的支持，并变换显示格式为图标显示，将变成手机上常见的九宫格方式显示。

接口的详细描述：

名称	描述
player_ui_freeze	void player_ui_freeze(void); 冻结播放界面<用于 USB 联机时>，这个函数会给发送 id=PLAYER_REQUEST_FREEZE 的命令。
player_notify_play	void player_notify_play(void); 指示 UI 已经开始播放，这个函数会向 home 视图发送 PLAYER_REQUEST_PLAY_SINGLE_FILE 的命令以标识一首歌曲播放开始。
player_notify_stop	void player_notify_stop(void); 指示 UI 播放已经停止，这个函数会向 home 视图发送 PLAYER_REQUEST_STOP 的命令以标识一首歌曲播放停止。
player_notify_info	void player_notify_info(const char* information); 指示 UI 当前正在播放歌曲的信息，这个函数会更新当前播放信息，并向 home 视图发送 PLAYER_REQUEST_UPDATE_INFO 的命令以请求重新绘制播放信息。
player_notify_functionview	void player_notfiy_functionview(void); 请求 UI 进入功能菜单，这个函数会向 home 视图发送 PLAYER_REQUEST_FUNCTION_VIEW 的命令以请求进入功能视图<这个函数被 info workbench 调用>。

player_set_position	void player_set_position(rt_uint32_t position); 设置当前播放歌曲的正在播放的位置，这个函数会更新当前播放歌曲正在播放的位置。
player_set_title	void player_set_title(const char* title); 设置当前播放歌曲的标题，这个函数会更新当前播放信息，并向 home 视图发送 PLAYER_REQUEST_UPDATE_INFO 的命令以请求重新绘制播放信息。
player_set_buffer_status	void player_set_buffer_status(rt_bool_t buffering); 设置播放器是否处于缓冲状态，这个函数会更新当前播放缓冲状态,并向 home 视图发送 PLAYER_REQUEST_UPDATE_INFO 的命令以请求重新绘制播放信息。

play_list 提供的是播放列表的管理

每个播放项由如下结构描述

```
struct play_item
{
    char title[40]; /* 播放项标题或名称 */
    char *fn; /* 播放项指向的文件名或 URL */
    rt_uint32_t duration; /* 歌曲的总长度，单位是秒 */
};
```

名称	描述
play_list_start	struct play_item *play_list_start(void); 开始一个播放列表，播放列表将从 0 的位置开始给出播放项，并返回第 0 个播放项。
play_list_items	rt_uint32_t play_list_items(void); 获得播放列表中播放项的个数。
play_list_item	struct play_item* play_list_item(rt_uint32_t n); 获得第 n 个播放项，N 是从 0 开始的正整数。
play_list_current	struct play_item* play_list_current(void); 获得当前的播放项。
play_list_set_current	void play_list_set_current(rt_uint16_t n); 设置当前的播放项为第 n 个播放项。
play_list_get_current	rt_uint16_t play_list_get_current(void); 获得当前的播放项。
play_list_next	struct play_item* play_list_next(int mode); 获得下一个播放项。
play_list_prev	struct play_item* play_list_prev(int mode); 获得上一个播放项。

play_list_get_mode	int play_list_get_mode(void); 获得播放列表的模式，分为 PLAY_LIST_SINGLE – 仅播放单一歌曲 PLAY_LIST_REPEAT – 重复顺序播放 PLAY_LIST_RANDOM – 随机播放
play_list_append	void play_list_append(char* fn); 添加文件到播放列表中。
play_list_append_radio	void play_list_append_radio(const char* url, const char* station); 添加电台到播放列表中。
play_list_append_directory	void play_list_append_directory(const char* path); 添加目录中所有支持的媒体文件到播放列表中。
play_list_append_m3u	void play_list_append_m3u(const char* file); 添加 m3u 列表中的歌曲到播放列表中。

station_list 提供的是电台列表的选择
电台列表及其中的项定义为

```
struct station_item
{
    char title[40]; /* 电台标题 */
    char url[128]; /* 电台网址 */
};

struct station_list
{
    rt_uint32_t count; /* 列表中的电台项数目 */
    struct station_item* items; /* 电台项列表指针 */
};
```

接口详细说明

名称	描述
station_list_create	struct station_list* station_list_create(const char* fn); 这个函数根据指定的电台列表文件创建相应的电台列表。
station_list_destroy	void station_list_destroy(struct station_list* list); 这个函数删除 station_list_create 创建出来的电台列表。
station_list_select	struct station_item* station_list_select(struct station_list* list, struct rtgui_workbench* workbench); 这个函数在指定的 workbench 上提供了电台选择的视图，并且是以模态方式显示的视图。参数 list 为 station_list_create 创建的电台列表，参数 workbench 为父 workbench。 当用户选择了电台时，将返回电台项，否则返回 NULL。

net buffer 模块: netbuffer.c, wav.c, mp3.c
接口详细说明如下：

名称	描述
net_buf_init	void net_buf_init(rt_size_t size); 初始化 net buffer，参数 size 为缓冲区的大小。
net_buf_read	rt_size_t net_buf_read(rt_uint8_t* buffer, rt_size_t length); 从 net buffer 中读取指定长度的数据。 参数 buffer 用于保存读取到的数据，参数 length 指示出最大能够保存的长度。 返回实际读取到的长度。
net_buf_start_job	int net_buf_start_job(rt_size_t (*fetch)(rt_uint8_t* ptr, rt_size_t len, void* parameter), void (*close)(void* parameter), void* parameter); 请求 net buffer 开始一项工作，它会更改 net buffer 的状态为 buffering。 参数 fetch 给出获取数据的函数指针，参数 close 给出关闭连接的函数指针，参数 parameter 用于提供给 fetch、close 函数参数。 成功返回 0，失败返回-1（例如当前状态并不是 stopped 状态）。
net_buf_stop_job	void net_buf_stop_job(void); 请求 net buffer 停止一项工作。如果 net buffer 当前状态是 suspend 状态，说明 nbuf 线程处于挂起状态，会先唤醒 nbuf 线程，然后更改 net buffer 的状态为 stopping。
net_buf_get_usage	int net_buf_get_usage(void); 获得 net buffer 中已获得的数据缓冲长度；

其他说明：

static void net_buf_thread_entry(void* parameter);

这个函数是 nbuf 线程的入口，它运行后会等待_netbuf_mq 上的消息。当有请求达到时，将开始设定的工作（net_buf_do_job 函数）。

STM32 上有板载的外扩 SRAM，但存在两个限制：

- 速度比较慢，特别当有芯片的读取预测时，速度会慢很多。
- 外扩的 SRAM 做 DMA 有限制，当进行 DMA 传送时，FSMC 外设将不能够被主机再行访问，否则出 fault 错误。

基于这个原因，在 net buffer 模块中添加了片内的一个 memoy pool 支持：

void sbuf_init(void);

初始化片内 SRAM 内存块区；总计大小定义为

```
#define MP3_DECODE_MP_SZ      2560
static rt_uint8_t mempool[(MP3_DECODE_MP_SZ * 2 + 4)* 2]; // 5k x 2
```

共两块，每块的长度是 MP3_DECODE_MP_SIZE * 2 的大小，4 字节用于 memory pool 内部管理使用。

rt_size_t sbuf_get_size(void);
获得一块片内内存块的大小（返回 MP3_DECODE_MP_SIZE * 2）。

void* sbuf_alloc(void);
分配一块片内内存块，如果当前无内存块可用，调用线程将被挂起。

void sbuf_release(void* ptr);
释放一块片内内存块。

codec 模块：wav.c，mp3.c，wma.c
详细接口说明

名称	描述
wav	void wav(char* filename); 根据提供的文件名播放 wav 文件。
mp3	void mp3(char* filename); 根据提供的文件名播放 mp3 文件。针对于 mp3，调用解码函数时需要给出获取 mp3 数据的函数指针（decoder 中的 fetch_data 和 fetch_parameter）： rt_size_t fd_fetch(void* parameter, rt_uint8_t *buffer, rt_size_t length) 这个函数用于获取以文件为句柄的 mp3 数据。

ice_mp3	void ice_mp3(const char* url, const char* station);
	
	根据提供的电台地址和电台名播放 shoutcast 电台。这个函数将先根据给定的 URL 地址打开一个 shoutcast_session，如果成功将向 net buffer 请求开始一个 job，然后开始解码的过程。
	
	针对 shoutcast，生成了一个 ice job:
	static rt_size_t ice_fetch(rt_uint8_t* ptr, rt_size_t len, void* parameter)
	static void ice_close(void* parameter);
	这两个函数分别针对于一项 job 用到的 fetch 和 close 函数，这些函数指针会传递给 net buffer 做为一项 job 的方法。
	
	针对于 mp3，调用解码函数时需要给出获取 mp3 数据的函数指针（decoder 中的 fetch_data 和 fetch_parameter）：
	rt_size_t ice_data_fetch(void* parameter, rt_uint8_t *buffer, rt_size_t length);
	这个函数用于从 net buffer 缓冲中获取 mp3 数据。

http_mp3	void http_mp3(char* url);

	根据提供的地址播放 http 方式的 mp3 流。这个函数将先根据给定的 URL 地址打开一个 http_session，如果成功将向 net buffer 请求开始一个 job，然后开始解码的过程。
	
	针对 http session，生成了一个 http job:
	static rt_size_t http_fetch(rt_uint8_t* ptr, rt_size_t len, void* parameter);
	static void http_close(void* parameter);
	这两个函数分别针对于一项 job 用到的 fetch 和 close 函数，这些函数指针会传递给 net buffer 做为一项 job 的方法。
	
	针对于 mp3，调用解码函数时需要给出获取 mp3 数据的函数指针（decoder 中的 fetch_data 和 fetch_parameter）：
	rt_size_t http_data_fetch(void* parameter, rt_uint8_t *buffer, rt_size_t length);
	这个函数用于从 net buffer 缓冲中获取 mp3 数据。

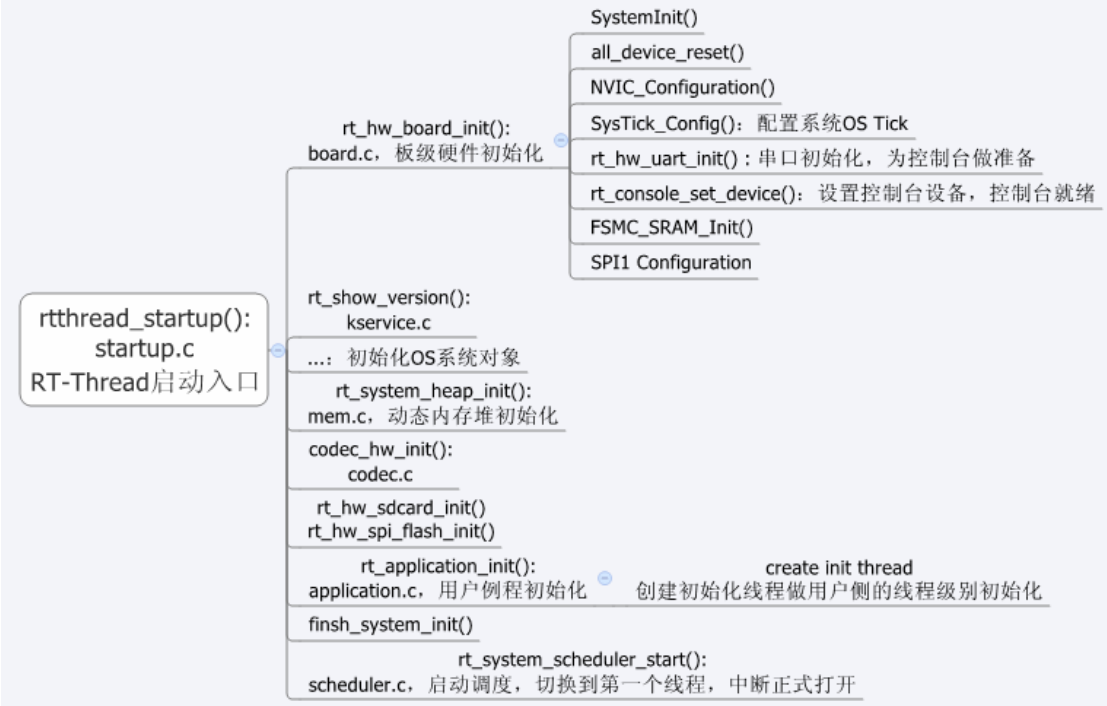
wma	void wma(const char* filename); **[未完成]** 根据提供的文件名播放 wma 文件。
mms_wma	**[未实现]** 更加提供的电台地址和电台名播放 mms 电台。

线程优先级设计

名称	描述
ply_bg	优先级 20 音频解码播放的任务分配在这个线程上运行，如果播放优先（如不出现声音咔咔的现象），这个线程的优先级应该设置得比较高。
ply_ui	RT-Thread/GUI 应用线程默认优先级 25，不能高于服务端线程，采用 25（如果设置高于 ply_bg，那么在进行 UI 绘图操作时将可能影响音频播放）。
nbuf	网络应用的优先级设计原则是，不应该高于 tcp 等几个线程，即不应高于 12。它用于缓冲网络音频数据，它的优先级应高于 UI 类线程的优先级（ply_ui 是 25）。当前优先级设置为 22。
key	扫描按键线程，被设计成周期性运行线程，每次运行 55us。这段时间对音频播放并无太大影响，所以优先级设得更高一些没什么关系，当前优先级设置为 30
rtgui	RT-Thread/GUI 线程，默认优先级 15
tcp	LwIP 主线程，默认优先级 10

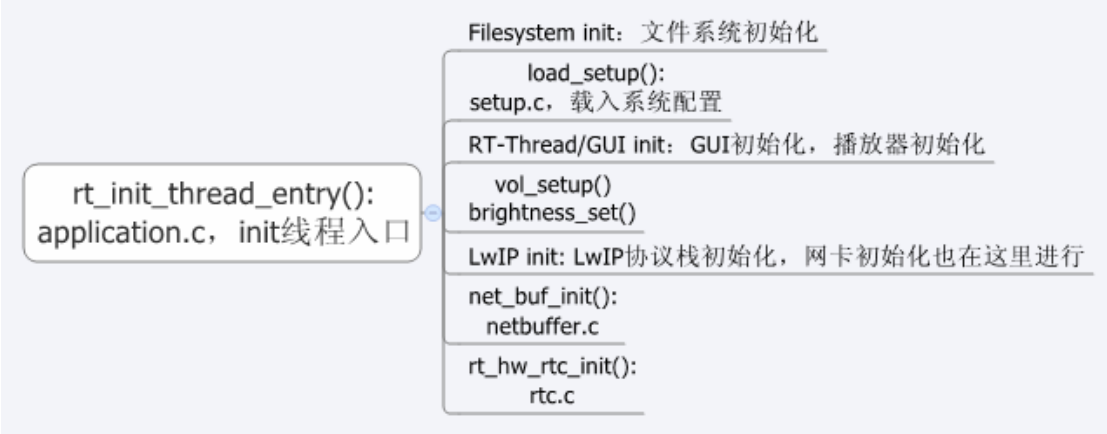
erx	以太网接收线程，默认优先级 12
etx	以太网发送线程，默认优先级 12

系统初始化过程



RT-Thread RTOS 的入口是 rtthread_startup 函数，位于 startup.c 中。由于使用的是 Keil MDK 做为编译器，rtthread_startup 函数将被 Keil MDK 的入口函数 main 调用。

在 rtthread_startup 函数中，将顺序调用 rt_hw_board_init()、rt_show_version、... rt_system_scheduler_start()。其中，rt_hw_board_init()用于系统的板级初始化，需要初始一些最基本的硬件，但这个时候 OS 还未启动起来。rt_application_init()函数在 RT-Thread RTOS 中用于初始化用户应用，这里启动了一个初始化线程（相当于把一部分工作放到初始化线程中）。



rt_init_thread_entry()是初始化线程的入口，在这里会初始化一些系统组件以及播放器

模块，网络缓冲管理器模块。把一些初始化放在这里的好处：对于一些初始化需要做等待的地方，可以使用 OS 提供的线程延时函数，把这段时间切换给其它线程（例如网卡初始化，它会进行 PHY 的自动协商）。