

IoT-Camera 学习笔记之

初识 IoT-Camera

--作者: 燕十三(flyingcys)

-- blog:<http://blog.csdn.net/flyingcys>

--QQ:294102238

1. 开发板介绍

1.1. 概述

IoT-Camera 是一款由开发 RT-Thread 操作系统的[上海睿赛德电子科技有限公司](#)推出的开源物联网摄像头开发板，采用 RT-Thread 开源实时操作系统，支持 C/C++/lua 编程开发，兼容 Arduino, 可通过 Wi-fi 将视频数据传输至手机、平板等设备。

IoT-Camera 购买地址:

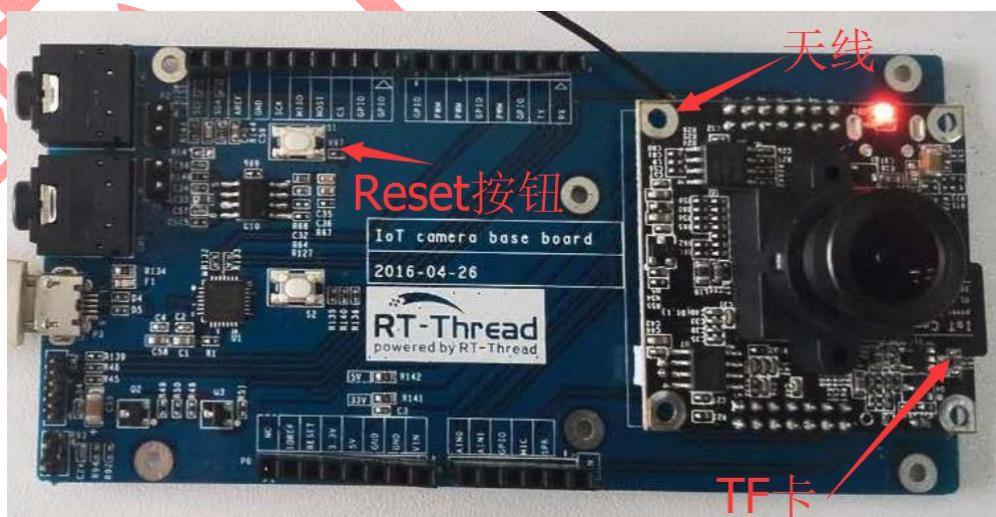
<https://item.taobao.com/item.htm?u=o1m5ek5ce617&id=538501985648>

RT-Thread 是一款嵌入式实时操作系统，包含实时嵌入式操作系统内核及 TCP/IP 协议栈、文件系统、无线网络栈、设备抽象层、存储机制、算法、图形库、libc 接口、POSIX 接口等。RT-Thread 源码下载地址: <https://github.com/RT-Thread/rt-thread>。

不熟悉 RT-Thread 开发的可先登录官网 <http://www.rt-thread.org/> 了解详情。

RT-Thread 最新版本的编程手册下载地址:

http://www.rt-thread.org/download/manual/rtthread_manual.zh.pdf



请注意核心板的连接，正对方向看 IoT-Camera 开发板，天线朝上，TF 卡由右边插入；位于上半部分的是按钮是 Reset 按钮

1.2. 硬件配置

- CPU: 采用富瀚微电子的 FH8620。ARM1176JS 内核, 300MHZ 主频, 集成 128Mbits DDR
- Wi-Fi: 采用 AP6181 模组, 支持 802.11b/g/n , 支持 station 或软 AP 模式
- 视频: 支持 H.264 编码和支持 JPEG/MJPEG 编码
- 音频: 内嵌 Audio Codec (单声道输入和输出), 支持麦克风输入
- 硬件接口:
 - ◆ 一个 iic 接口
 - ◆ 一组 SPI 接口
 - ◆ 一个串口
 - ◆ 六个 gpio 接口
 - ◆ 三路 pwm 接口
 - ◆ 一个 micro SD 卡插槽
 - ◆ 两路 ADC 输入
 - ◆ LINE IN 与 LINE OUT

1.3. 软件功能

- RT-Thread 基本系统平台, 涵盖: RT-Thread 内核, SD 卡上文件系统, TCP/IP 协议栈
- TFTP、web server 方式更新固件;
- Wi-Fi Station 模式/AP 模式 (支持 WEP、WPA/WPA2 等加密方式) ;
- 摄像头以 720P 方式录像成 H.264 视频并存储到 SD 卡中;
- 摄像头以 720P 方式录像成 H.264 格式, 并通过 RTSP TCP 或 UDP 方式通过 wifi 传输出去;
- 摄像头以 720P 方式录像成 mjpeg 视频流, 提供给浏览器查看;
- 摄像头以 720P 方式录像存储到 SD 卡中 (H.264 格式), 并提供一路 720P 子帧用于 mjpeg 方式供浏览器浏览
- 支持 POSIX C/C++开发, 支持 LUA 脚本

IoT-Camera 软件功能持续更新中, 请关注官网发布情况

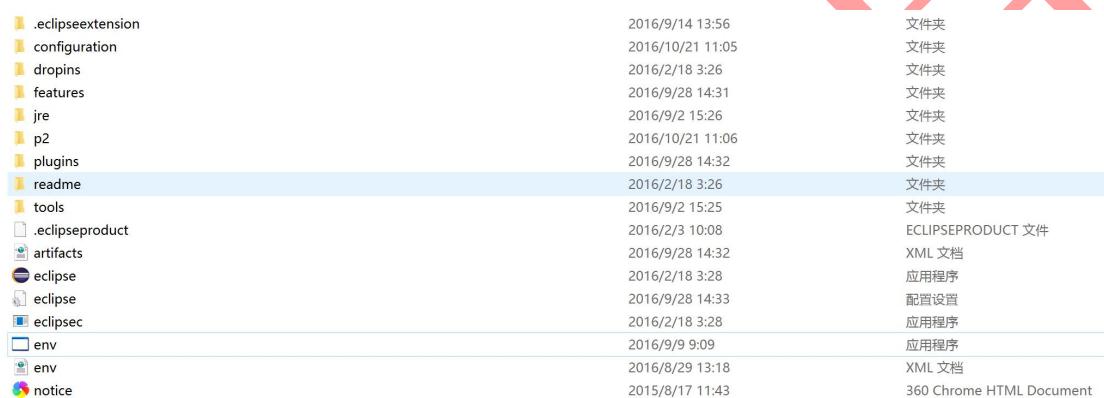
2. 开发环境配置

IoT-Camera 采用 camera studio 集成开发环境。camera studio 是由睿赛德为 IoT-Camera 开发板专门开发的集成开发环境，基于 eclipse 定制，当前支持 windows 32bit/64bit 开发环境，请选择对应版本下载。

2.1. 下载

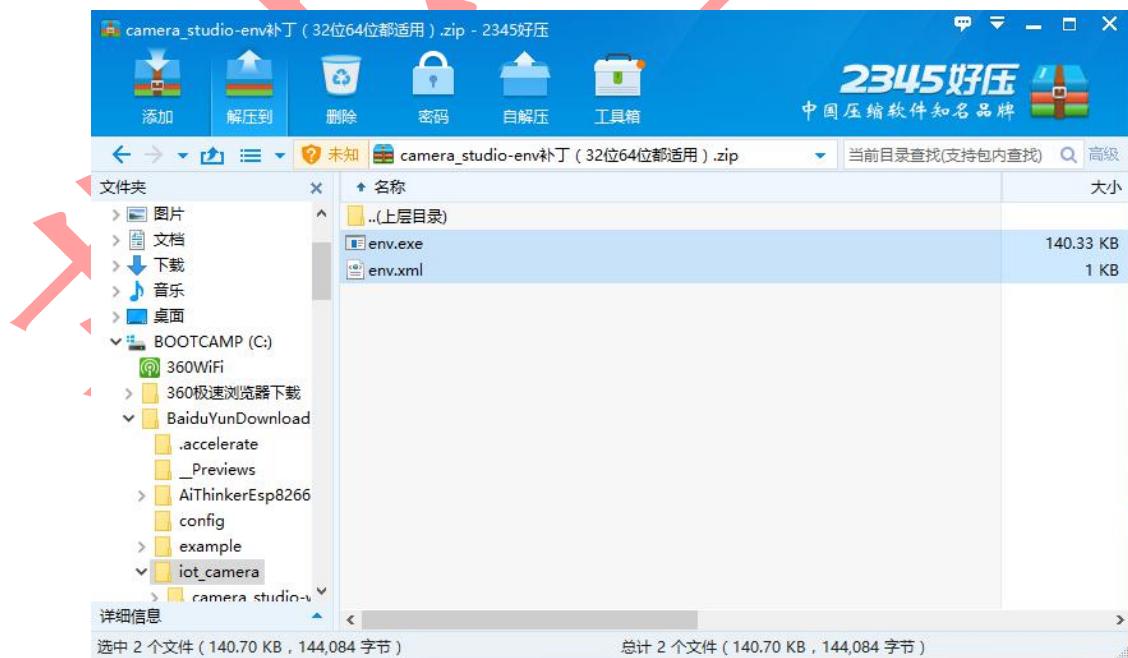
下载地址为：<http://pan.baidu.com/s/1pLguWt1> 密码: 6c33

下载后，请解压至开发主机，特别注意，解压后的目录不能含中文、空格等目录。



.eclipseextension	2016/9/14 13:56	文件夹
configuration	2016/10/21 11:05	文件夹
dropins	2016/2/18 3:26	文件夹
features	2016/9/28 14:31	文件夹
jre	2016/9/2 15:26	文件夹
p2	2016/10/21 11:06	文件夹
plugins	2016/9/28 14:32	文件夹
readme	2016/2/18 3:26	文件夹
tools	2016/9/2 15:25	文件夹
.eclipseproduct	2016/2/3 10:08	ECLIPSEPRODUCT 文件
artifacts	2016/9/28 14:32	XML 文档
eclipse	2016/2/18 3:28	应用程序
eclipse	2016/9/28 14:33	配置设置
eclipsec	2016/2/18 3:28	应用程序
env	2016/9/9 0:09	应用程序
env	2016/8/29 13:18	XML 文档
notice	2015/8/17 11:43	360 Chrome HTML Document

下载、解压完成后，请先打开网盘内有一个“camera_studio-env 补丁（32位 64位都适用）”完成 camera studio env 补丁

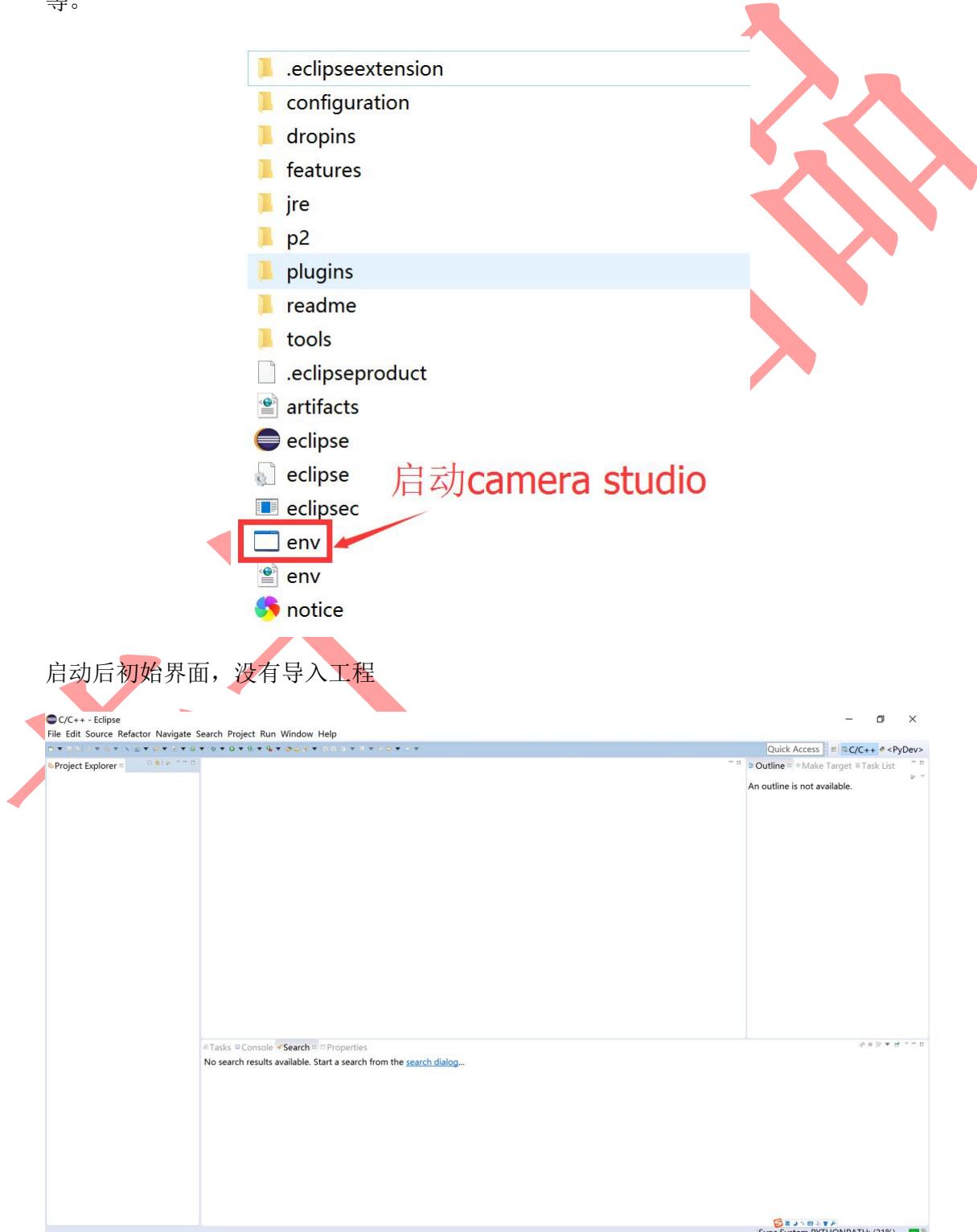


2.2. 启动 camera studio

点击“env.exe”文件启动 camera studio，因为 camera studio 需要做一些环境变量的设置（环境变量就在那个 env.xml 里面吧）。

注意：不是点击 eclipse.exe 启动的哦！

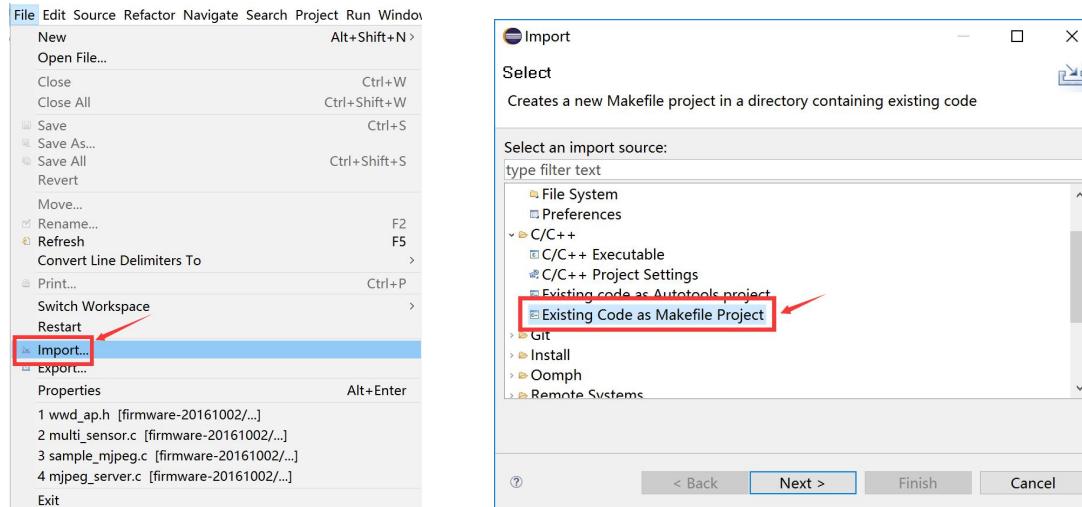
camera studio 内含整套开发所需要的工具，如交叉编译器、make、python、scons 等。



2.3. SDK 源码

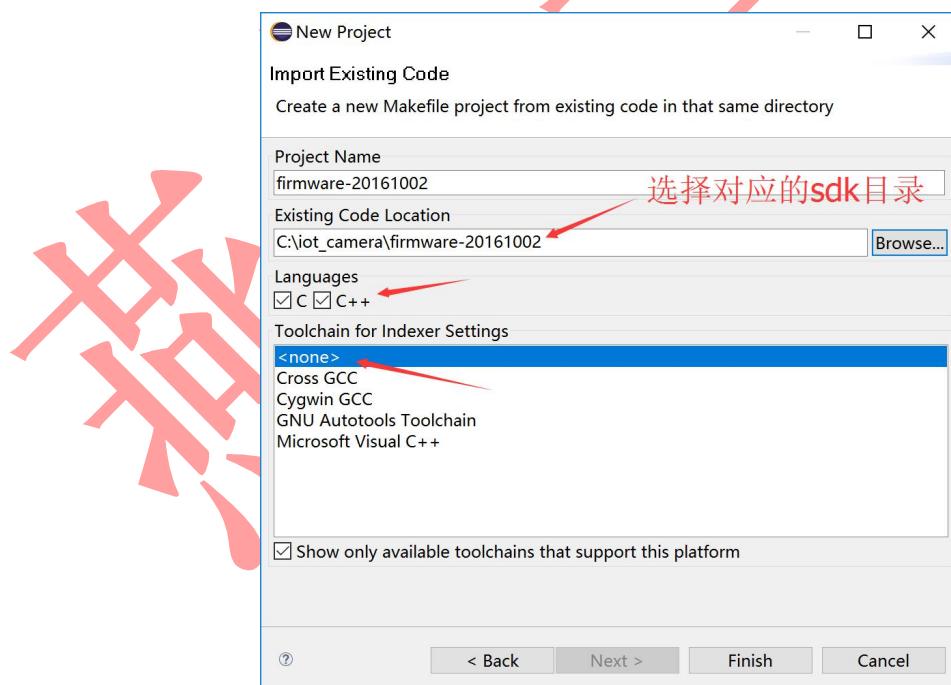
2.3.1. 导入

通过选择“File”->“Import”->“Existing Code as Makefile Project”导入

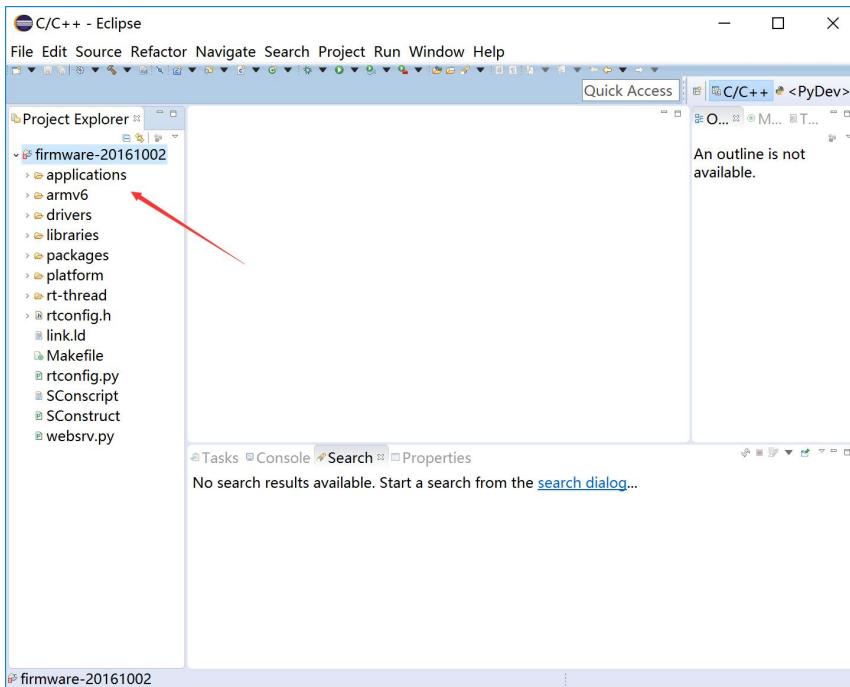


选择对应的 sdk 目录，

- Languages: 默认选择为 c/c++
- Toolchain: 默认选择为 none

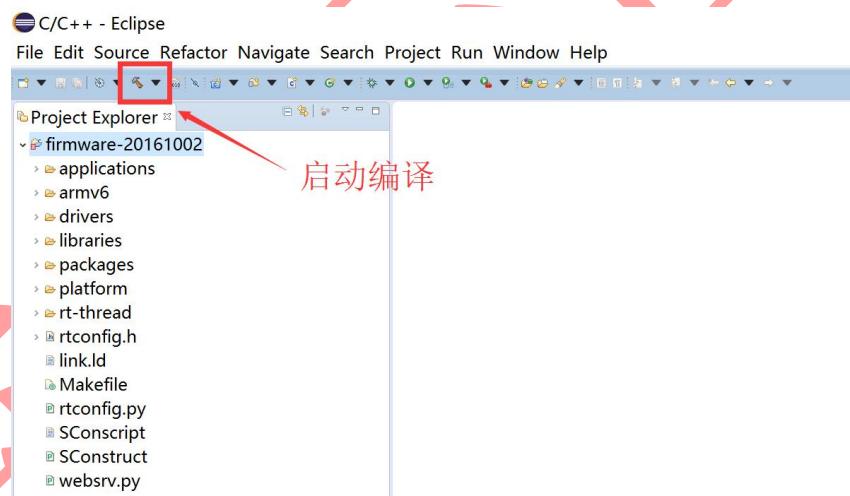


目前官方对 IOT-Camera 的 sdk 在持续更新中，请关注官网发布情况。



2.3.2. 编译

点击工具栏的小锤子或 **CTRL+B** 编辑 SDK



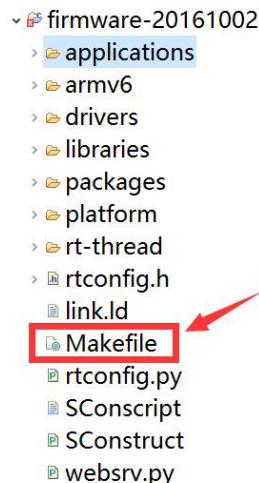
编译完成后，可在 **Console** 下查看编译情况。rtthread.bin 就是最终 bin 文件

```
CDT Build Console [firmware-20161002]
CC build\kernel\src\object.o
CC build\kernel\src\scheduler.o
CC build\kernel\src\thread.o
CC build\kernel\src\timer.o
LINK rtthread.elf
arm-none-eabi-objcopy -O binary rtthread.elf rtthread.bin
arm-none-eabi-size rtthread.elf
    text      data      bss      dec      hex filename
1488344   352264  2392812  4233420   4098cc rtthread.elf
scons: done building targets.

12:00:28 Build Finished (took 1m:57s.825ms)
```

3. SDK 目录结构

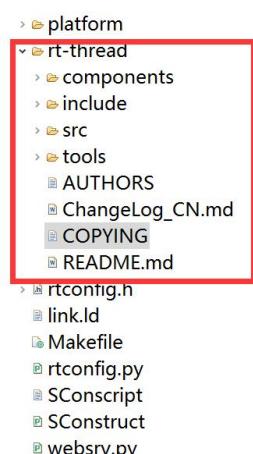
- 1) Makefile: eclipse 调用 make 的入口文件, IoT-Camera 采用 scons 完成编译; rtthread 源码都采用 scons 完成 build



- 2) armv6 目录:针对 FH8620 内核的支持, 包括栈初始化、上下文切换、MMU 配置等

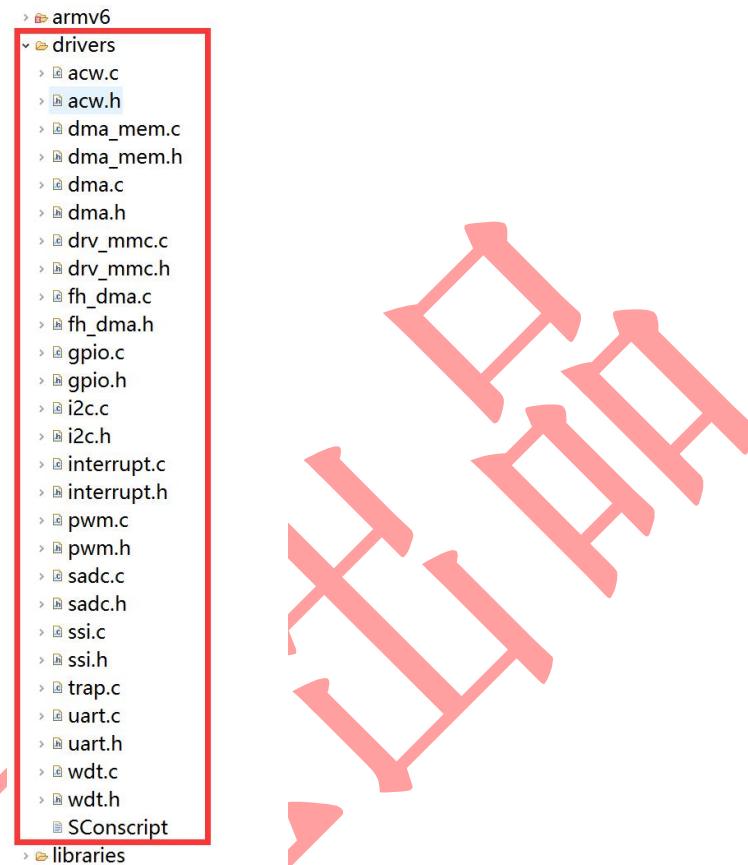


- 3) rt-thread 目录:包括 rt-thread 内核、文件系统 dfs、网络协议栈 lwIP、shell 命令行、c++ 支持等。目前采用的是比较新的 v2.5.0 版本代码



IoT-Camera 学习笔记

- 4) drivers 目录: FH8620 外设驱动, 包括 gpio、i2c、ssi、dma、sdio 等; 和 rt-thread 发行版本的 bsp 目录功能一致



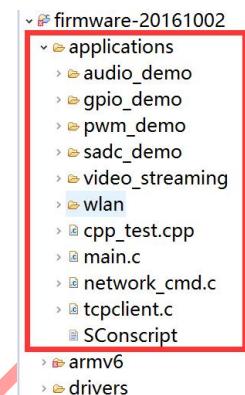
- 5) libraries 目录: 包含系统的启动汇编代码; 同时也包括一些底层库 (AP6181 wifi 驱动库、extlib 芯片 ISP 库等), 一般这部分代码不需要做相应的修改



- 6) platform 目录: FH8620 板级代码, 和 rt-thread 发行版本的 board 目录功能一致。



- 7) applications 目录: 应用目录, 目前有的 demo 有 audio、gpio、pwm、adc、video 等, 该目录下还包括 main 主函数



- 8) packages 目录: 软件包目录, 包括 cJSON、ezxml、telnet_server、wget 等, 该目录下软件包会越来越多



4. SDK 运行流程

4.1. 启动入口

1) 汇编级入口，异常向量表中的 reset 也指向这里，启动后程序跳转到这里运行。

```

Project Explorer
firmware-20161002
  applications
  armv6
  drivers
  libraries
    AP6181
    driverlib
    extlib
    inc
    startup
      gcc
        start_gcc.S
        SConscript
        SConscript
  packages
  platform

start_gcc.S
138
139 /* - - - - - entry - - - - -
140 reset:
141     /* set the cpu to SVC32 mode */
142     mrs r0,cpsr
143     bic r0,r0,#MODEMASK
144     orr r0,r0,#SVC MODE
145     msr cpsr,r0
146
147     @vector 0x0
148     @enable icaches
149     @little endian
150     @disable dcaches, mmu
151     //ldr r0, =0x00400078
152     @ldr r0, =0x0000107a
153     //mcr p15, 0, r0, c1, c0, 0
154
155

```

2) 依次完成各种模式下栈的配置情况。

3) 进入 C 的入口：_rtthread_startup，该函数跳转至 rt-thread 内核

```

start_gcc.S
195     and r0, r0, #(1 << 22) /*Set the U bit, bit 22*/
196     mcr p15, 0, r0, c1, c0, 0
197
198 #ifdef RT_USING_VFP
199     bl vfp_init
200#endif
201
202     /* start RT-Thread Kernel */
203     ldr pc, _rtthread_startup
204
205 _rtthread_startup:
206     .word rtthread_startup
207

```

rtthread_startup 函数位于 rt-thread->src 目录下的 components.c 文件内

开始进入rt-thread的世界

```

Project Explorer
firmware-20161002
  applications
  armv6
  drivers
  libraries
  packages
  platform
  rt-thread
    components
    include
    src
      clock.c
      components.c
      device.c
      idle.c
      ipc.c
      irq.c

start_gcc.S components.c
210     RT_ASSERT(result != RT_EOK);
211 #endif
212
213     rt_thread_startup(tid);
214 }
215
216 int rtthread_startup(void)
217 {
218     rt_hw_interrupt_disable();
219
220     /* board level initialization
221      * NOTE: please initialize heap inside board initialization.
222      */
223     rt_hw_board_init();
224
225     /* show RT-Thread version */
226     rt_show_version();
227
228     /* timer custom initialization */
229

```

4.2. RT-Thread 的 C 入口 rtthread_startup:

完成各种板级初始化、RTT_Log 显示、用户代码初始化、调度器启动等工作

```

216 int rtthread_startup(void)
217 {
218     rt_hw_interrupt_disable();
219
220     /* board level initialization
221      * NOTE: please initialize heap inside board initialization.
222      */
223     rt_hw_board_init();
224
225     /* show RT-Thread version */
226     rt_show_version(); ← RTT log 显示
227
228     /* timer system initialization */
229     rt_system_timer_init();
230
231     /* scheduler system initialization */
232     rt_system_scheduler_init();
233
234     /* create init_thread */
235     rt_application_init(); ← 用户代码初始化
236
237     /* timer thread initialization */
238     rt_system_timer_thread_init();
239
240     /* idle thread initialization */
241     rt_thread_idle_init();
242
243     /* start scheduler */
244     rt_system_scheduler_start(); ← 开始系统调度
245
246     /* never reach here */
247     return 0;
248 }

```

- 1) 硬件板级初始化函数 `rt_hw_board_init`: 主要完成了中断初始化、MMU 初始化等工作

```

1 void rt_hw_board_init() ← 硬件板级初始化
2 {
3     /* disable interrupt first */
4     rt_hw_interrupt_disable();
5     /* initialize hardware interrupt */
6     rt_hw_interrupt_init();
7
8     /* initialize mmu */
9     rt_hw_mmu_init(&fh_mem_desc, sizeof(fh_mem_desc) / sizeof(fh_mem_desc[0]));
0
1     rt_system_heap_init((void*) &_bss_end, (void*) FH_RTT_OS_MEM_END); ← heap 初始化
2
3 #ifdef RT_USING_DMA_MEM
4     /* just use the last 100KB */
5     fh_dma_mem_init(rt_uint32_t *) FH_RTT_OS_MEM_END, FH_DMA_MEM_SIZE);
6 #endif
7
8     /* initialize the system clock */
9     rt_hw_clock_init();
0     //add_iomux_init 2015-3-11 by yu.zhang for fh81(fullhan)
1     //iomux_init();
2     fh_iomux_init(PMU_REG_BASE + 0x5c);
3     //add_clk_init 2015-3-11 by yu.zhang for fh81(fullhan)
4     clock_init();

```

a. MMU 初始化: 通过内存描述数组, 完成内存段映射关系

```

452
453 static struct mem_desc fh_mem_desc[] = ← 内存描述数组
454 {
455     { 0xA0000000, FH_RTT_OS_MEM_END-1, 0xA0000000, SECT_RWX_CB, 0, SECT_MAPPED },
456     { FH_RTT_OS_MEM_END, FH_DDR_END-1, FH_RTT_OS_MEM_END, SECT_RWNX_NCNB, 0, SECT_MAPPED }
457     { 0xFFFFF1000-1, 0xA0000000, SECT_TO_PAGE, PAGE_ROX_CB, PAGE_MAPPED }, /* io
458     { 0xE0000000, 0xF1300000-1, 0xE0000000, SECT_RWNX_NCNB, 0, SECT_MAPPED }, /* io
459     { 0xF4000000, 0xF4100000-1, 0xF4000000, SECT_RWNX_NCNB, 0, SECT_MAPPED }, /* GP
460 };
461

```

b. heap 空间定义

```

48 #ifndef FH_DDR_START
49 #define FH_DDR_START          0xA0000000
50 #define FH_DDR_END            0xA1000000
51
52 #define FH_RTT_OS_MEM_SIZE    0x00600000
53 #define FH_DMA_MEM_SIZE       0x20000 /* 128k */
54
55 #define FH_RTT_OS_MEM_END    (FH_DDR_START + FH_RTT_OS_MEM_SIZE)
56 #define FH_SDK_MEM_START     (FH_RTT_OS_MEM_END + FH_DMA_MEM_SIZE)
57 #define FH_RTT_OS_HEAP_END   FH_SDK_MEM_START
58 #define FH_SDK_MEM_SIZE      (FH_DDR_END - FH_SDK_MEM_START)
59 #endif /* end of FH_DDR_START */

```

2) 用户初始化函数: rt_application_init

```

196 void rt_application_init(void)
197 {
198     rt_thread_t tid;
199
200 #ifdef RT_USING_HEAP
201     tid = rt_thread_create("main", main_thread_entry, RT_NULL,
202                           2048, RT_THREAD_PRIORITY_MAX / 3, 20);
203     RT_ASSERT(tid != RT_NULL);
204 #else
205     rt_err_t result;
206
207     tid = &main_thread;
208     result = rt_thread_init(tid, "main", main_thread_entry, RT_NULL,
209                           2048, RT_THREAD_PRIORITY_MAX / 3, 20);
210     RT_ASSERT(result != RT_EOK);
211 #endif
212

```

创建main任务

在 main_thread_entry 里面完成:

- ◆ 组件初始化
- ◆ 调用 main

```

179 /* the system main thread */
180 void main_thread_entry(void *parameter)
181 {
182     extern int main(void);
183     extern int $Super$$main(void);
184
185     /* RT-Thread components initialization */
186     rt_components_init(); → 完成组件初始化相关工作
187
188     /* invoke system main function */
189 #if defined (__CC_ARM)
190     $Super$$main(); /* for ARMCC. */
191 #elif defined(__ICCARM__) || defined(__GNUC__)
192     main(); → 调用主函数
193 #endif
194 }

```

所以 main 函数只是 rt-thread 里面一个优先级为 RT_THREAD_PRIORITY_MAX / 3 的任务; main 函数位于 applications 目录下 main.c 文件内



```

Project Explorer
firmware-20161002
  applications
    audio_demo
    gpio_demo
    pwm_demo
    sadc_demo
    video_streaming
    wlan
    cpp_test.cpp
    main.c →
    network_cmd.c
    tcpclient.c
    SConscript
  armv6
  drivers
  libraries
  packages
  platform
  rt-thread
  rtconfig.h
  ...

```

```

2* * This file is part of FH8620 BSP for RT-Thread distribution.
26
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <fcntl.h>
30
31 int main(int argc, char** argv)
32 {
33     int fd;
34     char cmd[] = "/init.sh";
35
36     fd = open("/init.sh", O_RDONLY, 0);
37     if (fd >= 0)
38     {
39         close(fd);
40
41         printf("exec init.sh...\n");
42         msh_exec(cmd, sizeof(cmd));
43     }
44 }
45
46 return 0;
47

```

到这里, 整个 Project 已经启动完成, 用户可在 main 里面添加自己的代码!

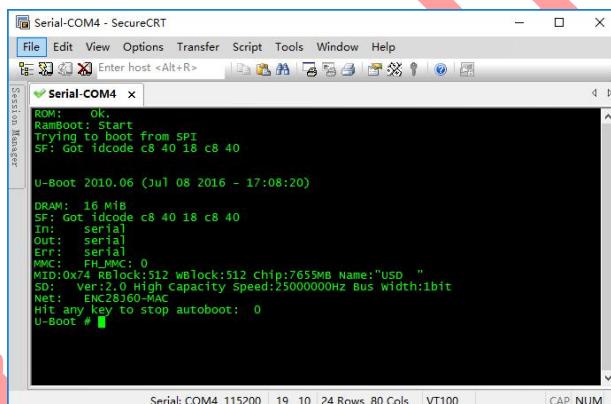
5. BIN 文件烧写运行

刚拿到手的 IOT-Camera 开发板是没有烧写摄像头固件的，所以我们要先刷机！！！
最新版本的 Firmware 已经支持 3 种不同的刷机方式。

5.1. 更新固件方式 1

手工 TF 卡刷机

- 1) Step1: 将 camera studio 编译完成后的 rtthread.bin 复制至 TF 卡根目录，插入 IoT-Camera 核心板。
- 2) Step2: 将 USB 插入电脑后，打开串口终端，波特率为 115200，本人使用的是 SecureCRT。在显示“Hit any key to stop autoboot”倒计时为 0 前，按任意键进入 uboot 命令行模式。



- 3) Step3: 选择启动方式：IoT-Camera 可以通过 2 种不同的启动模式启动
 - a) 启动方式 1：将 rtthread.bin 从 TF 复制至内部 flash，每次从内部 flash 启动 IoT-Camera，每次更新固件都需要手工进 uboot 输入命令。（也太麻烦了吧！！！）
 - 将 TF 卡中的 rtthread.bin 加载到 0xA0000000 内存地址中


```
fatload mmc 0 0xA0000000 rtthread.bin
```
 - 将镜像文件烧录至 Flash 中（从 4M 位置，烧录 2M 数据）


```
sf probe 0
sf erase 400000 200000
sf write a0000000 400000 200000
```
 - 设置 U-Boot 启动参数并保存


```
set bootcmd 'sf probe 0; sf read a0000000 400000 200000; go a0000000'
save
```
 - 重新启动


```
reboot
```

reset

```
U-Boot 2010.06 (Jul 08 2016 - 17:08:20)
DRAM: 16 MiB
SF: Got idcode c8 40 18 c8 40
In: serial
Out: serial
Err: serial
MMC: FH_MMC: 0
MID:0x41 RBlock:512 wBlock:512 Chip:14992MB Name:"SD16G"
SD: ver:2.0 High Capacity Speed:25000000Hz Bus Width:1bit
Net: ENC28J60-MAC
Hit any key to stop autoboot: 0
SF: Got idcode c8 40 18 c8 40
16384 KiB w25q128 at 0:0 is now current device
## Starting application at 0xA0000000 ...
\\_ /
- RT Thread Operating System
  2.5.0 build Oct 21 2016
  2006 - 2016 Copyright by rt-thread team
```

可以看到，程序已经正常运行

b) 启动方式 2：从 TF 卡启动 IoT-Camera，uboot 自动读取 rtthread.bin 文件

- 在 uboot 中修改启动参数

set bootcmd fatload mmc 0 0xa0000000 rtthread.bin\; go 0xa0000000

- 保存 uboot 配置

save

- 重新启动

reset

两种启动方式比较：

1. 启动方式 1 适合正式生产的时候使用，在程序调试阶段并不合适。
2. 启动方式 2 会在启动时读取 rtthread.bin 增加启动时间，但适合程序调试。
3. 在下面的更新方式中会经常用到，所以推荐使用启动方式 2。

5.2. 更新固件方式 2

通过 wget 下载 rtthread.bin 至 TF 卡并在 TF 卡上运行

更新固件方法 1 只适合的 IoT-Camera 第一次刷机使用或者由于程序出错无法正常启动时使用！如果每次更新程序都需要去通过 TF 卡在电脑上复制 bin 文件那是非常麻烦的！那我们来试试方法 2 吧！

1) Step1 格式化分区：这个很重要哦！否则 rtthread 系统访问不了 TF 卡(step1 只需要执行一次就可以了)

- a. 固件运行起来后，需要格式化 rootfs 分区。输入：

mkfs rootfs 后重启

```
msh />mkfs rootfs
msh />reset
```

b. 创建 sdcards 目录.输入:

mkidr sdcards 后重启

```
msh />mkdir sdcards  
msh />reset
```

这样，rtthread 系统能够自动装载 flash 上的 rootfs 分区到'/'根目录，而把 TF 卡装载到'sdcards'目录中。可通过 ls 命令查看当前目录

```
msh />ls  
Directory /:  
sdcards <DIR>
```



2) Step2 启动 web_server

c. 如果电脑上没有安装 python2.7，可先安装，安装完成后，在 windows 命令行输入：

python c:\iot_camera\firmware-20161002\webser.py 启动 web_server

```
c:\选择C:\WINDOWS\system32\cmd.exe - python c:\iot_camera\firmware-20161002\webser.py  
Microsoft Windows [版本 10.0.14393]  
(c) 2016 Microsoft Corporation. 保留所有权利。  
  
C:\Users\flyingcys>python  
Python 2.7.12 (v2.7.12:33e0cf91556, Jun 27 2016, 15:24:40) [MSC v.1500 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>>  
C:\Users\flyingcys>启动web_server  
C:\Users\flyingcys>python c:\iot_camera\firmware-20161002\webser.py  
serving at port 8000  
127.0.0.1 -- [21/Oct/2016 23:39:00] "GET / HTTP/1.1" 200 -
```

- 如果不想安装 python2.7，camera_studio 自带了 python2.7，在 windows 命令行输入：

```
cd c:\iot_camera\camera_studio\tools\Python27  
python.exe c:\iot_camera\firmware-20161002\webser.py
```

```
c:\命令提示符 - python.exe c:\iot_camera\firmware-20161002\webser.py  
Microsoft Windows [版本 10.0.14393]  
(c) 2016 Microsoft Corporation. 保留所有权利。  
  
C:\Users\flyingcys>cd c:\iot_camera\camera_studio\tools\Python27  
c:\iot_camera\camera_studio\tools\Python27>python.exe c:\iot_camera\firmware-20161002\webser.py  
serving at port 8000
```

启动完成后，在浏览器输入 <http://localhost:8000> 查看是否成功运行。

注：以上 2 种方式启动的 web_server 的根目录不一样，这个对于我们下面使用 wget 获取文件位置有很大关系（本人使用的是自己安装 python2）

3) Step3 将 IoT-Camera 连接网络：

使用 wifi join ssid password 命令，ssid 和 password 为需要配置的网络名和密码。

IoT-Camera 学习笔记

```
msh />
msh />
msh />
msh />wifi join xiaomi_7937 flyingcys
```

输入ssid和password连接路由器

连接成功！并可使用 ifconfig 查看 ip 确认是否正确连接

```
[WIFI] wiced_wifi_join_specific result:SUCCESS  
msh >/ifconfig  
network interface: w0  
MTU: 1500  
MAC: 44 2c 05 16 23 58  
FLAGS: UP LINK_UP ETHARP IGMP  
ip address: 192.168.31.144  
gw address: 192.168.31.1  
net mask : 255.255.255.0
```

连接成功

查看ip

4) Step4 使用 wget 命令下载

在电脑端先将 rtthread.bin 复制至对于 web_server 根目录下，在 msh 命令行输入：

```
wget http://192.168.31.166:8000/rtthread.bin /sdcard/rtthread.bin
```

如配置及输入命令正确，wget 将开始从 web_server 下载 rtthread.bin 文件

下载完成！！！重启 iot-camera 就可运行新程序。

```
U-Boot 2010.06 (Jul 08 2016 - 17:08:20)

DRAM: 16 MiB
SF: Got idcode c8 40 18 c8 40
In: serial
Out: serial
Err: serial
Net: none
Hit any key to stop autoboot: 0
reading rtthread.bin

1840672 bytes read
## Starting application at 0xA0000000 ...
\r _ / Thread Operating System
- RT \ 2.5.0 build oct 21 2016
2006 - 2016 copyright by rt-thread team
A:a=100
B:#str test
msh >/lwIP-2.0.0 initialized!
G0129 detection
root file system initialized!
SD card capacity 7561216 KB
probe mmcSD block device!
Found part[0], begin: 1048576, size: 7.215GB
sdcard file system initialized!
FH_WDT_DEBUG: rt_hw_wdt_init start
FH_WDT_DEBUG: rt_hw_wdt_init end
media process init success!!
process init success!!
vpdu init success!!
JPEG init success!!
VONI init success!!
hello, world!
```

可修改 Makefile 文件，每次编译完成后自动将 rtthread.bin 复制至 web_server 根目录(目录请自行修改)

```

Makefile main.c
1 scons := python ${SCONS}\scons.py
2
3 all:
4   @${scons}
5   cp rtthread.bin ../../Users/flyingcys
6
7 clean:
8   @${scons} -c
9
10 buildua:
11   @${scons} -target=ua -s
12
13 buildlib:
14   @${scons} --buildlib=WICED -s
15
16 cleanlib:
17   @${scons} --cleanlib -s
18

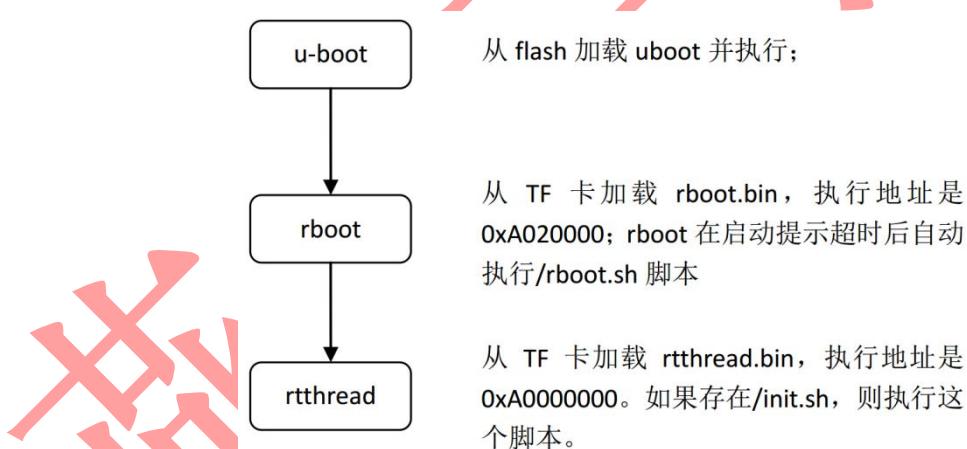
```

每次make完成后自动复制
rtthread.bin至websrv根目录

5.3. 更新固件方式 3

通过 rboot 启动并下载 rtthread.bin 至 TF 卡并在 TF 卡上运行

更新固件方式 2 在程序正常运行后，可通过 wget 获取 rtthread.bin 至 TF 卡完成更新。但如果用户修改程序后无法正常运行 wget 呢？只能用方式 1 了吧？？？当然还有更便捷的方法，通过 rboot 启动 IoT-Camera。IoT-Camera 启动流程：



让我们来试试用 rboot 启动系统，更新系统吧！

1) Step1：将 rboot.bin rboot.sh init.sh rtthread.bin 这 4 个文件复制至 TF 卡根目录。文件下载地址为：网盘 <http://pan.baidu.com/s/1pLguWt1> 的 prebuilt_demo 目

录下（密码:6c33）

2) Step2:按照前章描述启动 IoT-Camera 并进入 uboot 命令行，输入如下命令：

■ 设置执行地址

setenv bootcmd fatload mmc 0 0xa0200000 rboot.bin\; go 0xa0200000

- 设置 uboot 的提示计数等待为 1

```
setenv bootdelay 1
```

- 保存设置

```
save
```

- 重启

```
reset
```

这时候就可以看到 rboot 已经正常工作了



```
Err: serial
MMC: F1_MMC: 0
MID:0x27 RBlock:512 wBlock:512 Chip:7384MB Name:"SD08G"
SD: Ver:2.0 High Capacity Speed:25000000Hz Bus width:1bit
Net: ENC28J60-MAC
Hit any key to stop autoboot: 0
reading rboot.bin
1101800 bytes read
## Starting application at 0xA0200000 ...

powered by rt-thread 2.5.0
(C) 2016 by shanghai real-thread Technology Co., Ltd
lwIP-2.0.0 initialized!
SD card capacity 7561216 KB
probe mmcisd block device!
found part[0], begin: 1048576, size: 7.215GB
File System initialized!
Hit any key to stop rboot.sh: 0
```

rboot.sh 脚本是 rboot 启动提示计数器减到 0 后自动运行的脚本文件。在 rboot.sh 中可以指定下一个启动的 rt_thread 固件位于哪里。如:

- 加载 TF 卡根目录下的 rtthread.bin

```
boot /rtthread.bin
```

- 加载 web_server 服务器上的固件

```
boot http://192.168.31.166:8000/rtthread.bin
```

为了让 rboot 能上网，需要先在 rboot 下先进行 wifi 配置；输入：wifi cfg ssid passwd。完成 wifi 配置后，rboot 会在提示计数器时试图关联到本地 wifi 网络中，以便执行 rboot.sh 脚本时，能尽快下载到更新需要的固件

本人使用的是加载 TF 卡根目录下 rtthread.bin 这种方式，如需要更新程序直接进 rboot 命令行输入：boot http://192.168.31.166:8000/rtthread.bin；这样可提高 iot-camera 加载启动速度



```
rboot />boot http://192.168.31.166:8000/rtthread.bin
wifi connected! local address: 192.168.31.144
to open web server....done!
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

下载完成后，rboot 会自动

6. 视频测试

在 iot-camera 正常启动 rt-thread 系统后，可以使用 help 命令，或 tab 键获得当前 rt-thread 支持的命令行。

```
msh />help
RT-Thread shell commands:
dhcpd      - start dhcpcd on network interface
wifi_ap    - start wifi ap and start mjpeg
tcpclient   - startup tcp client
c_function  - C++ function test
mjpeg      - mjpeg program
rtsp       - rtsp command
test_udp   - start rtsp service with udp protocol
wifi_scan  - wifi scan test
wifi       - wifi command
lua        - lua Script Interpreter
telnetd    - start telnet server
tftpd     - tftp server
vi         - vi in ms
wget      - web download file
reset     - system reset
list_symbol - list symbol for module
list_sdio   - list sdio device
version    - show RT-Thread version information
list_thread - list thread
list_sem    - list semaphore in system
list_event  - list event in system
list_mutex  - list mutex in system
list_mailbox - list mail box in system
list_msqueue - list message queue in system
list_memheap - list memory heap in system
list_mempool - list memory pool in system
list_timer  - list timer in system
list_device - list device in system
list_module - list module in system
exit       - return to RT-Thread shell mode.
help       - RT-Thread shell help.
ls         - List information about the FILES.
cp         - Copy SOURCE to DEST.
mv         - Rename SOURCE to DEST.
cat        - Concatenate FILE(s).
rm         - Remove(unlink) the FILE(s).
cd         - Change the shell working directory.
pwd        - Print the name of the current working directory.
mkdir     - Create the DIRECTORY.
mkfs      - format disk with file system
echo       - echo string to file
ifconfig   - list the information of network interfaces
dns        - list the information of dns
```

具体该指令如何使用，可输入该指令：

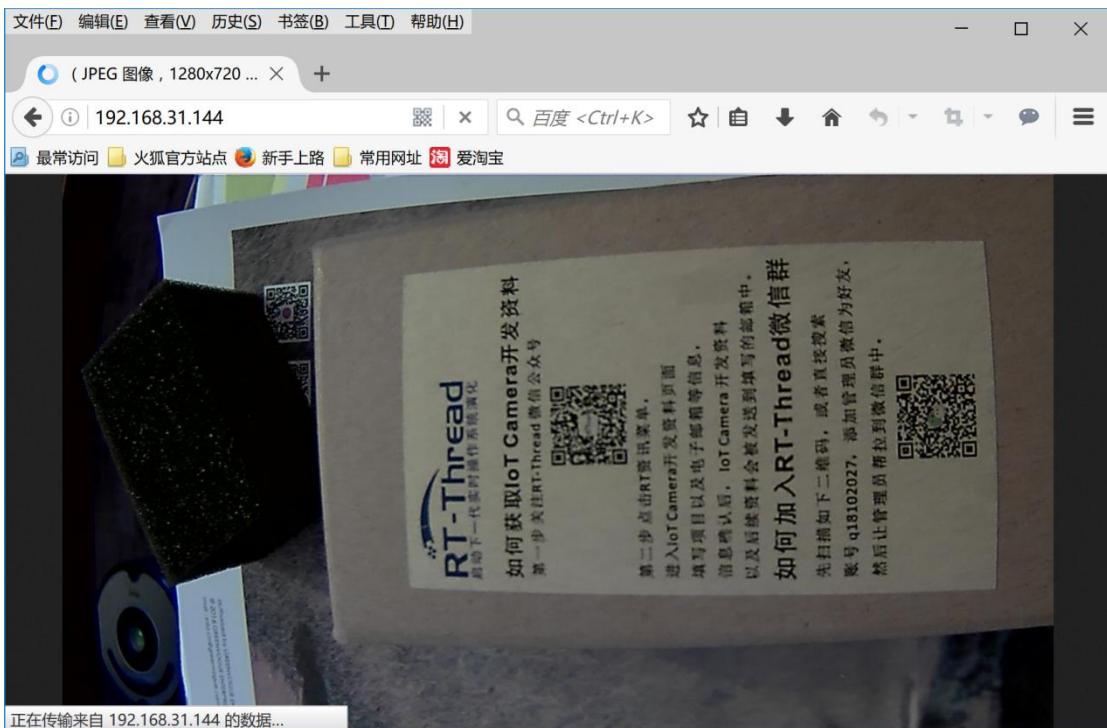
```
msh />wifi
wifi - to join a saved wlan
wifi ap AP_SSID AP_PASSWORD
wifi join SSID PASSWORD
wifi help
wifi cfg SSID PASSWORD
```

1. 将 iot-camera 连入 wifi。联网操作请参照前述操作。
2. 在 msh 命令行中输入 mjpeg start 启动 mjpeg_server， 默认端口为 80

```
msh />mjpeg start
pae open success!
[ISP]:isp core need total size : 0x1f0000.
sensor probe...
sensor: fixed (GC1024)
[ISP]:isp core memory size : 0x1f0000.
[ISP]:media_malloc used size : 0x300.
[ISP]:media_malloc used size : 0x600.
[ISP]:media_malloc used size : 0x800.
[ISP]:media_malloc used size : 0x31880.
=====
resolution:1280X720.
DRV REG :ADDR =0xa04023f4.
IPF PKG :ADDR =0xa0b68f00.
IPFB PKG:ADDR =0xa0b69200.
IPB PKG :ADDR =0xa0b69500.
STAT_BASE:ADDR=0xa0b69700.
=====
[ISP]:media_malloc used size : 0x3a6a0.
[ISP]:media_malloc used size : 0x18bea0.
[ISP]:media_malloc used size : 0x1c42a0.
[ISP]:media_malloc used size : 0x1e04a0.
gme_buf_addr :ADDR =0xa0ba0000.
nr3d_frame_buf_addr :ADDR =0xa0ba35a0.
nr3d_coeff_buf_addr :ADDR =0xa0cf4da0.
ltm_stat_buf_addr :ADDR =0xa0d2d1a0.
af_buf_addr :ADDR =0x0.
[ISP]:strategy init !!!!.
[ISP]:ISP init success !!!!!
```

IoT-Camera 学习笔记

3. 启动成功后，在浏览器中输入当前 iot-camera 的 IP 地址，就可看到视频流。



并可在 msh 中断上看到 client 连接状态

```
msh />mjpeg start
pae open success!
[ISP]:isp core need total size : 0x1f0000.
sensor probe...
sensor: fixed (GC1024)
[ISP]:isp core memory size : 0x1f0000.
[ISP]:media_malloc used size : 0x300.
[ISP]:media_malloc used size : 0x600.
[ISP]:media_malloc used size : 0x800.
[ISP]:media_malloc used size : 0x31880.

resolution:1280x720.
DRV REG :ADDR =0xa04023f4.
IPF PKG :ADDR =0xa0b68f00.
IPFB PKG:ADDR =0xa0b69200.
IPB PKG :ADDR =0xa0b69500.
STAT_BASE:ADDR=0xa0b69700.

[ISP]:media_malloc used size : 0x3aba0.
[ISP]:media_malloc used size : 0x18bea0.
[ISP]:media_malloc used size : 0x1c42a0.
[ISP]:media_malloc used size : 0x1e04a0.
gme_buf_addr :ADDR =0xa0ba0000.
nr3d_frame_buf_addr :ADDR =0xa0ba35a0.
nr3d_coeff_buf_addr :ADDR =0xa0cf4da0.
ltm_stat_buf_addr :ADDR =0xa0d2d1a0.
af_buf_addr :ADDR =0x0.
[ISP]:strategy init !!!!
[ISP]:ISP init success !!!!
msh />mjpeg_server: client connected
```

注：在 PC 上请使用 Chrome 或 firefox 浏览器来打开视频，IE 不支持这种方式