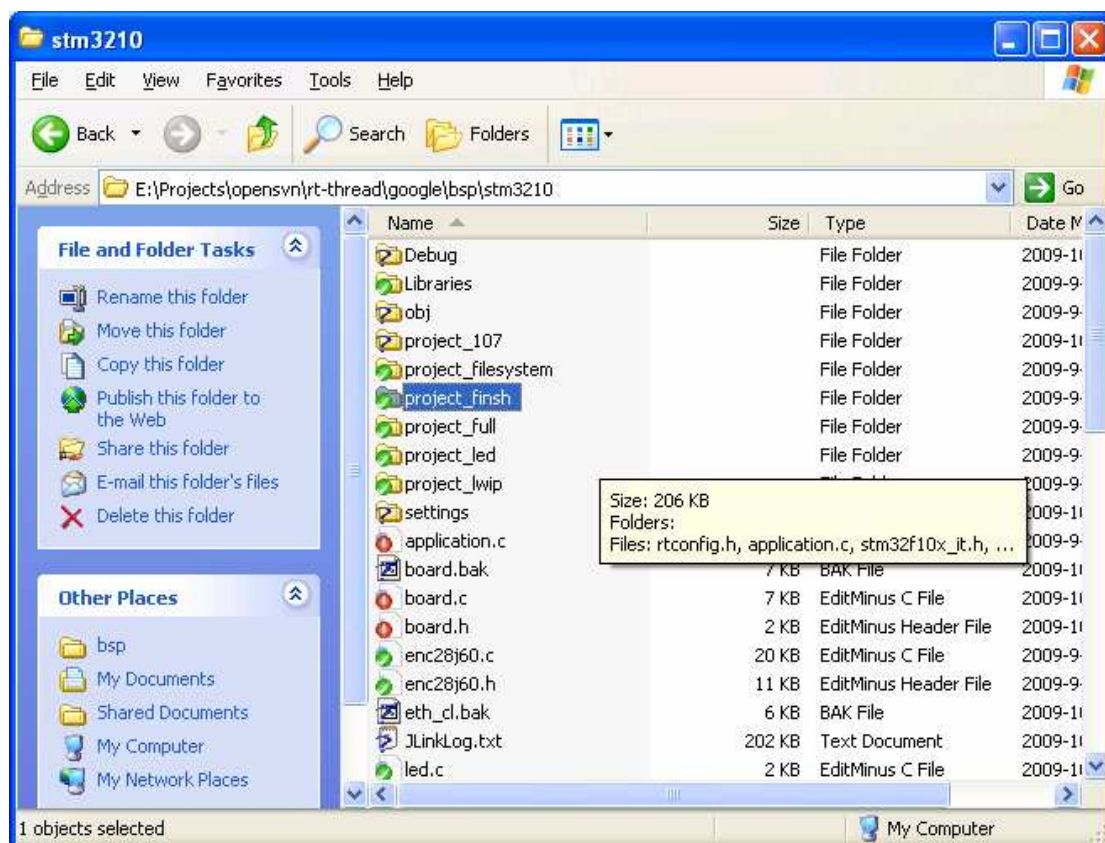


1 强大的 finsh shell

1.1 FinSH 的工程

新的 STM32 版本中，RT-Thread + FinSH 的工程如下



需要把 project_finsh 目录中所有文件都复制到 stm3210 目录下,然后打开 project 工程。

编译下载运行后,它会在串口 1 输出 RT-Thread 的信息以及 finsh>>的命令行提示符,以等待用户的命令输入。

```
\ | /
- RT - Thread Operating System
/ | \ 0.3.0 build Sep 30 2009
2006 - 2009 Copyright by rt-thread team
sdcard init failed
File System initialization failed!
TCP/IP initialized!
finsh>>
```

1.2 FinSH 组件起因

通常，大家在接触 RT-Thread 时，都会以为这是一个最不重要的组件，可有可无，而且在产品中也不会用到，用了还担心是否会对系统造成不必要的开销，甚至影响到系统的稳定性、可靠性。

做为单片机系统的研发人员，可能大多使用 JTAG 来调试程序，出问题了就祭出仿真器的尚方宝剑，似乎宝剑一出，任何前方的艰难险阻就会困难而解。只是有一类情况（而且这种情况还是困扰系统稳定的最大的问题），问题出现的几率并不是 100%的，时间上非确定，环境上非确定。例如一个系统存在内存泄漏，如何去确定它是否存在内存泄漏呢，难道等到内存全部使用完了才发觉？

看看 RT-Thread + FinSH 的解决方式：（请选择 STM3210 分支中携带 finsh 的版本）

```
\ | /
- RT - Thread Operating System
/ | \ 0.3.0 build Oct 17 2009
2006 - 2009 Copyright by rt-thread team
finsh>>list_me
--function:
list_mem          -- list memory usage information
list_mempool      -- list memory pool in system
finsh>>list_mem()
total memory: 59376
used memory : 0
maximum allocated memory: 0
                0, 0x0000
finsh>>
```

FinSH 最大的用处在于，在系统运行的时刻获取系统的信息，虽然信息的显示非常原始（通过串口等输出字符串信息），但分析这些原始的信息却最能找到有用的信息。例如上面的 `list_mem()` 命令，显示的就是系统的内存占用状况：

```
total memory: 59376
used memory: 0
maximum allocated memory: 0
```

显示的信息包括：

- 总的可用内存，59376 字节
- 当前使用的内存，0 字节
- 历史中最大的分配内存，0 字节

再能够在线的执行一些命令后，如果怀疑系统中存在内存泄漏，过一段时间就可以再运行 `list_mem()` 命令以显示系统内存占用情况。如果 `used memory` 一直持续的增长，那么就极有可能发生了内存泄漏。

FinSH 是一类非常特殊的 shell 系统，即不类似于 DOS 的命令行用法，也不类似于 UNIX

类的 `sh` 用法。而是一种看起来陌生而实际上非常非常熟悉的 C 语言模式：在命令行状态下，一条命令相对于 C 中的一条语句。例如下面的语句：

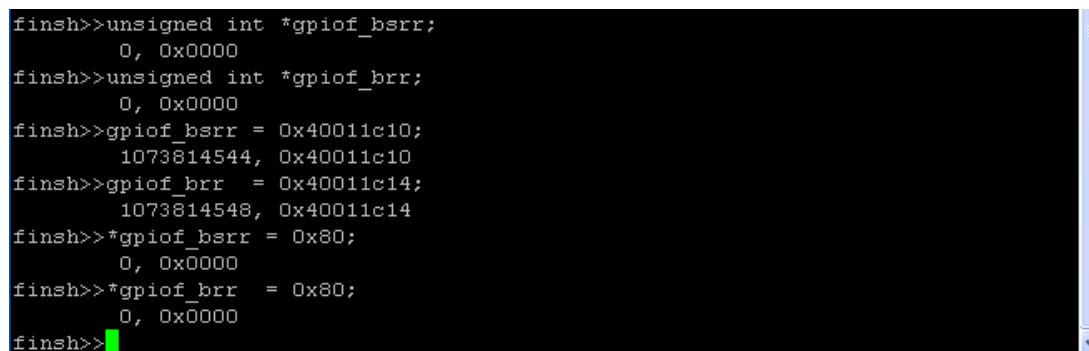
```
unsigned int *gpiof_bsrr;
unsigned int *gpiof_brr;
gpiof_bsrr = 0x40011c10; /* GPIOF 的 BSRR 地址 */
gpiof_brr = 0x40011c14; /* GPIOF 的 BRR 地址 */
*gpiof_bsrr = 0x80;      /* Set Pin7 */
*gpiof_brr = 0x80;      /* Reset Pin7 */
```

如果 GPIOF 做了相应的配置：

`*gpiof_bsrr = 0x80`，代表的是点亮连接在 PF7 上的 LED

`*gpiof_brr = 0x80`，代表的是熄灭连接在 PF7 上的 LED

我们让它在 `finsh` 下一条条执行：



```
finsh>>unsigned int *gpiof_bsrr;
      0, 0x0000
finsh>>unsigned int *gpiof_brr;
      0, 0x0000
finsh>>gpiof_bsrr = 0x40011c10;
      1073814544, 0x40011c10
finsh>>gpiof_brr = 0x40011c14;
      1073814548, 0x40011c14
finsh>>*gpiof_bsrr = 0x80;
      0, 0x0000
finsh>>*gpiof_brr = 0x80;
      0, 0x0000
finsh>>
```

每条执行完，下面会打印出一系列的数字，这些代表的是执行这些表达式的结果，例如

`gpiof_brr = 0x40011c14;`

执行完会把 `gpiof_brr` 的值显示出来，如果是一个函数，那么显示的是这个函数的返回值。

执行完后，PF7 上的 LED 也相应的点亮和熄灭。

1.3 获得在线帮助

先看看系统中默认存在哪些命令：

在 `finsh>>` 提示符下输入 `list()` 命令，将显示下面的列表

```
finsh>>list()
--Function List:
led          -- set led[0 - 3] on[1] or off[0].
list_mem     -- list memory usage information
hello        -- say hello world
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_msgqueue -- list message queue in system
list_mempool -- list memory pool in system
list_timer   -- list timer in system
list_device  -- list device in system
list         -- list all symbol in system
--Variable List:
dummy        -- dummy variable for finsh
             0, 0x0000
finsh>>
```

主要分两大类，Function List 和 Variable List，分别对应系统中可以在 shell 下调用的函数列表和变量列表。函数，例如 led、list_mem、hello。变量，例如其中的 dummy。

1.4 FinSH 在 STM32 版本中新加特性

目前 RT-Thread 在 google svn 的版本添加了下面的新功能：

- 命令历史记录

通过上、下键，浏览历史命令，并把当前命令切换到相应的命令。切换后可以继续编辑命令。

- 自动完成 & 获得帮助

在输入一部分命令时，例如 list，继续输入 TAB 键，获得相应匹配的命令帮助，如果系统中只存在唯一的命令，将自动补全后面缺少的内容：

```
\ | /
- RT - Thread Operating System
/ | \ 0.3.0 build Sep 30 2009
2006 - 2009 Copyright by rt-thread team
sdcard init failed
File System initialization failed!
TCP/IP initialized!
finsh>>list Tab->
--function:
list_date    -- set date
list_mem     -- list memory usage 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_msgqueue -- list message queue in system
list_mempool -- list memory pool in system
list_timer   -- list timer in system
list_device  -- list device in system
finsh>>list_th Tab->
--function:
list_thread  -- list thread
finsh>>list_thread
```

1.5 添加用户命令

自动补全、历史记录帮助，C 语言风格的表达式输入，让 `finsh` 使用起来非常灵活。还有一个，如何产生自己的命令。打开 `led.c` 看看最后的那个 `led` 函数：

```
51  #ifndef RT_USING_FINSH
52  #include <finsh.h>
53  void led(rt_uint32_t led, rt_uint32_t value)
54  {
55      /* init led configuration if it's not initied. */
56      if (!led_initied)
57      {
58          LED_Configuration();
59          led_initied = 1;
60      }
61
62      /* set led status */
63      if (value)
64          rt_hw_led_on(led);
65      else
66          rt_hw_led_off(led);
67  }
68  FINSH_FUNCTION_EXPORT(led, set led[0 - 3] on[1] or off[0].)
69  #endif
```

在使用 `finsh` 之前，需要先包含 `finsh.h` 头文件。输出一个函数到 `finsh` 中，需要用 `FINSH_FUNCTION_EXPORT` 宏定义，函数的参数自动从 `shell` 中获取，但为了系统安全性考虑，需要检查参数的值是否在可允许的范围内。