[FreeModbus--完全分析](http://blog.csdn.net/rianboe917/article/details/16811963)

说明：freemodbus-v1.5.0

**主流程**

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16811963)

1. /\* ----------------------- Start implementation -----------------------------\*/
2. **int**
3. main( **void** )
4. {
5. eMBErrorCode    eStatus;
7. eStatus = eMBInit( MB\_RTU, 0x0A, 0, 38400, MB\_PAR\_EVEN );
9. /\* Enable the Modbus Protocol Stack. \*/
10. eStatus = eMBEnable(  );
12. **for**( ;; )
13. {
14. ( **void** )eMBPoll(  );
16. /\* Here we simply count the number of poll cycles. \*/
17. usRegInputBuf[0]++;
18. }
19. }

由上述主函数可知协议栈经eMBInit和eMBEnable初始化、使能后进入协议栈的循环eMBPoll中。

**eMBInit分析**

首先，使用eMBInit初始化协议栈，根据你使用的参数eMBMode eMode初始化相应的函数入口！

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16811963)

1. #if MB\_RTU\_ENABLED > 0
2. **case** MB\_RTU:
3. pvMBFrameStartCur = eMBRTUStart;
4. pvMBFrameStopCur = eMBRTUStop;
5. peMBFrameSendCur = eMBRTUSend;
6. peMBFrameReceiveCur = eMBRTUReceive;
7. pvMBFrameCloseCur = MB\_PORT\_HAS\_CLOSE ? vMBPortClose : **NULL**;
8. pxMBFrameCBByteReceived = xMBRTUReceiveFSM;
9. pxMBFrameCBTransmitterEmpty = xMBRTUTransmitFSM;
10. pxMBPortCBTimerExpired = xMBRTUTimerT35Expired;
12. eStatus = eMBRTUInit( ucMBAddress, ucPort, ulBaudRate, eParity );
13. **break**;
14. #endif
15. #if MB\_ASCII\_ENABLED > 0
16. **case** MB\_ASCII:
17. pvMBFrameStartCur = eMBASCIIStart;
18. pvMBFrameStopCur = eMBASCIIStop;
19. peMBFrameSendCur = eMBASCIISend;
20. peMBFrameReceiveCur = eMBASCIIReceive;
21. pvMBFrameCloseCur = MB\_PORT\_HAS\_CLOSE ? vMBPortClose : **NULL**;
22. pxMBFrameCBByteReceived = xMBASCIIReceiveFSM;
23. pxMBFrameCBTransmitterEmpty = xMBASCIITransmitFSM;
24. pxMBPortCBTimerExpired = xMBASCIITimerT1SExpired;
26. eStatus = eMBASCIIInit( ucMBAddress, ucPort, ulBaudRate, eParity );
27. **break**;
28. #endif

以上代码中pvMBFrameStartCur、pvMBFrameStopCur等即协议栈函数的接口，对于不同模式使用不同的函数进行赋值初始化！！此编写模式可以借鉴学习！！

其中eMBRTUInit函数对底层驱动(串口和定时器)进行了初始化。

在上述初始化完成并且成功后对事件功能也进了初始化，最后全局变量eMBState = STATE\_DISABLED。

**eMBEnable的分析**

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16811963)

1. <span style="font-size:10px;">eMBErrorCode
2. eMBEnable( **void** )
3. {
4. eMBErrorCode    eStatus = MB\_ENOERR;
6. **if**( eMBState == STATE\_DISABLED )
7. {
8. /\* Activate the protocol stack. \*/
9. pvMBFrameStartCur(  );
10. eMBState = STATE\_ENABLED;
11. }
12. **else**
13. {
14. eStatus = MB\_EILLSTATE;
15. }
16. **return** eStatus;
17. }</span>

由第一节的分析，此时将启动协议栈pvMBFrameStartCur，查看程序该函数指针被分配到为eMBRTUStart。

该函数中将全局变量eRcvState = STATE\_RX\_INIT，并使能串口和定时器，注意此时的定时开始工作！！！

全局变量eMBState =STATE\_ENABLED。

**eMBPoll的分析**

在此循环函数中xMBPortEventGet(&eEvent ) == TRUE先判断是否有事件，无事件发生则不进入状态机！

还记得第二节定时器开始工作了吗？我们先看看该定时器如果超时了会发生什么事件！

在超时中断中我们将会调用pxMBPortCBTimerExpired函数，其中有以下代码：

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16830479)[在CODE上查看代码片](https://code.csdn.net/snippets/73761)

1. **BOOL**
2. xMBRTUTimerT35Expired( **void** )
3. {
4. **BOOL**            xNeedPoll = FALSE;
6. **switch** ( eRcvState )
7. {
8. /\* Timer t35 expired. Startup phase is finished. \*/
9. **case** STATE\_RX\_INIT:
10. xNeedPoll = xMBPortEventPost( EV\_READY );
11. **break**;
13. /\* A frame was received and t35 expired. Notify the listener that
14. \* a new frame was received. \*/
15. **case** STATE\_RX\_RCV:
16. xNeedPoll = xMBPortEventPost( EV\_FRAME\_RECEIVED );
17. **break**;
19. /\* An error occured while receiving the frame. \*/
20. **case** STATE\_RX\_ERROR:
21. **break**;
23. /\* Function called in an illegal state. \*/
24. default:
25. assert( ( eRcvState == STATE\_RX\_INIT ) ||
26. ( eRcvState == STATE\_RX\_RCV ) || ( eRcvState == STATE\_RX\_ERROR ) );
27. }
29. vMBPortTimersDisable(  );
30. eRcvState = STATE\_RX\_IDLE;
32. **return** xNeedPoll;
33. }

上一节分析中全局变量eRcvState =STATE\_RX\_INIT，因此第二节所说的定时器第一次超时将会发送xNeedPoll =xMBPortEventPost( EV\_READY )事件，

然后关闭定时器，全局变量eRcvState =STATE\_RX\_IDLE。此时，在主循环eMBPoll中将会执行一次EV\_READY下的操作，

之后会一直执行eMBPoll，整个协议栈开始运行！

接收数据分析

由于FreeModbus只支持从机模式，因此我们分析一下其在接收到数据后的操作！！！

接收数据

在上三节的操作中，我们可以知道进入eMBPoll循环后，串口中断是开启的。因此在接收到数据的时候，首先响应的应该是串口中断程序。

接收中断中将会调用接收状态机:

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16830479)[在CODE上查看代码片](https://code.csdn.net/snippets/73761)

1. **BOOL**
2. xMBRTUReceiveFSM( **void** )
3. {
4. **BOOL**            xTaskNeedSwitch = FALSE;
5. **UCHAR**           ucByte;
7. assert( eSndState == STATE\_TX\_IDLE );
9. /\* Always read the character. \*/
10. ( **void** )xMBPortSerialGetByte( ( **CHARCHAR** \* ) & ucByte );
12. **switch** ( eRcvState )
13. {
14. /\* If we have received a character in the init state we have to
15. \* wait until the frame is finished.
16. \*/
17. **case** STATE\_RX\_INIT:
18. vMBPortTimersEnable(  );
19. **break**;
21. /\* In the error state we wait until all characters in the
22. \* damaged frame are transmitted.
23. \*/
24. **case** STATE\_RX\_ERROR:
25. vMBPortTimersEnable(  );
26. **break**;
28. /\* In the idle state we wait for a new character. If a character
29. \* is received the t1.5 and t3.5 timers are started and the
30. \* receiver is in the state STATE\_RX\_RECEIVCE.
31. \*/
32. **case** STATE\_RX\_IDLE:
33. usRcvBufferPos = 0;
34. ucRTUBuf[usRcvBufferPos++] = ucByte;
35. eRcvState = STATE\_RX\_RCV;
37. /\* Enable t3.5 timers. \*/
38. vMBPortTimersEnable(  );
39. **break**;
41. /\* We are currently receiving a frame. Reset the timer after
42. \* every character received. If more than the maximum possible
43. \* number of bytes in a modbus frame is received the frame is
44. \* ignored.
45. \*/
46. **case** STATE\_RX\_RCV:
47. **if**( usRcvBufferPos < MB\_SER\_PDU\_SIZE\_MAX )
48. {
49. ucRTUBuf[usRcvBufferPos++] = ucByte;
50. }
51. **else**
52. {
53. eRcvState = STATE\_RX\_ERROR;
54. }
55. vMBPortTimersEnable(  );
56. **break**;
57. }
58. **return** xTaskNeedSwitch;
59. }

经过第3节的分析，此时全局变量eRcvState =STATE\_RX\_IDLE。接收状态机开始后，读取UART串口缓存中的数据，

并进入STATE\_RX\_IDLE分支中存储一次数据后开启超时定时器，进入STATE\_RX\_RCV分支继续接收后续的数据，

直至定时器超时！为什么要等待超时才能停止接收呢！

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16830479)[在CODE上查看代码片](https://code.csdn.net/snippets/73761)

1. /\* A frame was received and t35 expired. Notify the listener that
2. \* a new frame was received. \*/
3. **case** STATE\_RX\_RCV:
4. xNeedPoll = xMBPortEventPost( EV\_FRAME\_RECEIVED );
5. **break**;

可以发现接收数据时发生超时后，协议栈会发送EV\_FRAME\_RECEIVED接收完成这个信号。此时eMBPoll接收到此信号后会调用eMBRTUReceive函数。

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16830479)[在CODE上查看代码片](https://code.csdn.net/snippets/73761)

1. eMBErrorCode
2. eMBRTUReceive( **UCHARUCHAR** \* pucRcvAddress, **UCHARUCHAR** \*\* pucFrame, **USHORTUSHORT** \* pusLength )
3. {
4. **BOOL**            xFrameReceived = FALSE;
5. eMBErrorCode    eStatus = MB\_ENOERR;
7. ENTER\_CRITICAL\_SECTION(  );
8. assert( usRcvBufferPos < MB\_SER\_PDU\_SIZE\_MAX );
10. /\* Length and CRC check \*/
11. **if**( ( usRcvBufferPos >= MB\_SER\_PDU\_SIZE\_MIN )
12. && ( usMBCRC16( ( **UCHARUCHAR** \* ) ucRTUBuf, usRcvBufferPos ) == 0 ) )
13. {
14. /\* Save the address field. All frames are passed to the upper layed
15. \* and the decision if a frame is used is done there.
16. \*/
17. \*pucRcvAddress = ucRTUBuf[MB\_SER\_PDU\_ADDR\_OFF];
19. /\* Total length of Modbus-PDU is Modbus-Serial-Line-PDU minus
20. \* size of address field and CRC checksum.
21. \*/
22. \*pusLength = ( **USHORT** )( usRcvBufferPos - MB\_SER\_PDU\_PDU\_OFF - MB\_SER\_PDU\_SIZE\_CRC );
24. /\* Return the start of the Modbus PDU to the caller. \*/
25. \*pucFrame = ( **UCHARUCHAR** \* ) & ucRTUBuf[MB\_SER\_PDU\_PDU\_OFF];
26. xFrameReceived = TRUE;
27. }
28. **else**
29. {
30. eStatus = MB\_EIO;
31. }
33. EXIT\_CRITICAL\_SECTION(  );
34. **return** eStatus;
35. }

eMBRTUReceive函数完成了CRC校验、帧数据地址、长度的赋值，便于给上层进行处理！之后发送( void)xMBPortEventPost( EV\_EXECUTE )事件。

处理数据时根据功能码调用相应的函数，这些函数存储在xFuncHandlers数组中！之后发送响应！完成一次操作！

**功能码0x04读输入寄存器**

在一个远程设备中，使用该功能码读取 1 至大约 125 的连续输入寄存器。请求 PDU 说明了起始地址和寄存器数量。

将响应报文中的寄存器数据分成每个寄存器为两字节，在每个字节中直接地调整二进制内容。  
对于每个寄存器，第一个字节包括高位比特，并且第二个字节包括低位比特。

实例：



以下是FreeModbus的代码：

**[objc]** [view plaincopy](http://blog.csdn.net/rianboe917/article/details/16831473)[在CODE上查看代码片](https://code.csdn.net/snippets/73834)

1. eMBException
2. eMBFuncReadInputRegister( **UCHARUCHAR** \* pucFrame, **USHORTUSHORT** \* usLen )
3. {
4. **USHORT**          usRegAddress;
5. **USHORT**          usRegCount;
6. **UCHAR**          \*pucFrameCur;
8. eMBException    eStatus = MB\_EX\_NONE;
9. eMBErrorCode    eRegStatus;
11. **if**( \*usLen == ( MB\_PDU\_FUNC\_READ\_SIZE + MB\_PDU\_SIZE\_MIN ) )
12. {
13. usRegAddress = ( **USHORT** )( pucFrame[MB\_PDU\_FUNC\_READ\_ADDR\_OFF] << 8 );
14. usRegAddress |= ( **USHORT** )( pucFrame[MB\_PDU\_FUNC\_READ\_ADDR\_OFF + 1] );
15. usRegAddress++;
17. usRegCount = ( **USHORT** )( pucFrame[MB\_PDU\_FUNC\_READ\_REGCNT\_OFF] << 8 );
18. usRegCount |= ( **USHORT** )( pucFrame[MB\_PDU\_FUNC\_READ\_REGCNT\_OFF + 1] );
20. /\* Check if the number of registers to read is valid. If not
21. \* return Modbus illegal data value exception.
22. \*/
23. **if**( ( usRegCount >= 1 )
24. && ( usRegCount < MB\_PDU\_FUNC\_READ\_REGCNT\_MAX ) )
25. {
26. /\* Set the current PDU data pointer to the beginning. \*/
27. pucFrameCur = &pucFrame[MB\_PDU\_FUNC\_OFF];
28. \*usLen = MB\_PDU\_FUNC\_OFF;
30. /\* First byte contains the function code. \*/
31. \*pucFrameCur++ = MB\_FUNC\_READ\_INPUT\_REGISTER;
32. \*usLen += 1;
34. /\* Second byte in the response contain the number of bytes. \*/
35. \*pucFrameCur++ = ( **UCHAR** )( **usRegCount** \* 2 );
36. \*usLen += 1;
38. eRegStatus =
39. eMBRegInputCB( pucFrameCur, usRegAddress, usRegCount );
41. /\* If an error occured convert it into a Modbus exception. \*/
42. **if**( eRegStatus != MB\_ENOERR )
43. {
44. eStatus = prveMBError2Exception( eRegStatus );
45. }
46. **else**
47. {
48. \*usLen += **usRegCount** \* 2;
49. }
50. }
51. **else**
52. {
53. eStatus = MB\_EX\_ILLEGAL\_DATA\_VALUE;
54. }
55. }
56. **else**
57. {
58. /\* Can't be a valid read input register request because the length
59. \* is incorrect. \*/
60. eStatus = MB\_EX\_ILLEGAL\_DATA\_VALUE;
61. }
62. **return** eStatus;
63. }

底层在获得数据阶经过校验后，将数据帧放入到pucFrame中传给eMBFuncReadInputRegister进行处理，该函数获取要读取的地址、数量。

该函数将提取的信息再提交给eMBRegInputCB填充响应的数据帧，该函数就是要自己根据需要编写的函数！