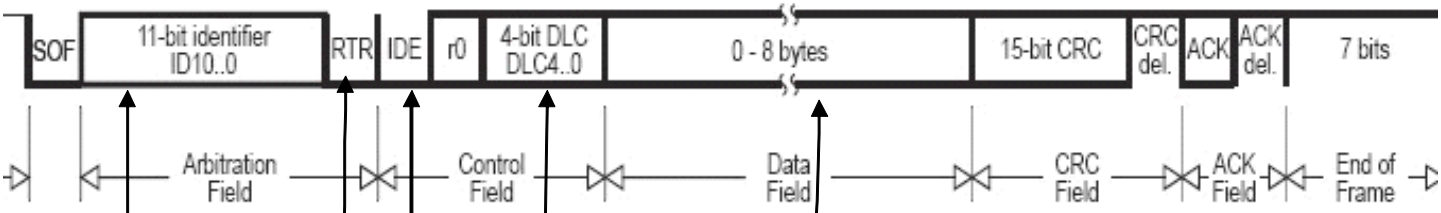
STM32 CAN 发送和接收过滤原理

通过对 CANBUS 协议的理解，我们知道：CAN 总线上的节点接收或发送数据都是以帧为单位的!!! CAN 协议规定了好几种帧类型，但是对于我们应用来说，只有数据帧和远程帧可以通过软件编程来控制。(其他几种帧都是由 CAN 控制器硬件实现的，我们想管也管不了)。而数据帧和远程帧最大的区别在于：远程帧没有数据域。数据帧分为标准数据帧和扩展数据帧，它们之间最大的区别在于：标识符(ID)长度不同（标准帧为 11 位，扩展帧为 29 位）。为了能更好地理解下面的内容，让我们先来回忆一下标准数据帧是什么样子的：



我们首先来看发送：

前面已经强调了 CAN 总线上的节点接收或者发送数据都是以帧为单位。假如我们要发送一个字节的数据：0x5A,是不是像串口发送数据那样，直接把 0x5A 写入发送缓冲寄存器，然后发送就可以了呢？NO!刚才已经强调了,CAN 总线上的数据都是以帧为单位的!!!我们必须按照帧的格式填充它!TNND,这么复杂？没关系，ST 库函数已经提供了一个结构，我们只需要填充该结构就可以了。我们来看一下这个结构的样子：

```
typedef struct
{
  uint32_t StdId;    //标准帧 ID，如果您要发送扩展帧。可以不管它
  uint32_t ExtId;    //扩展帧 ID，如果您要发送标准帧。可以不管它
  uint8_t IDE;       //您是想发送标准帧还是扩展帧?
  uint8_t RTR;       //您是想发送数据帧还是远程帧?
  uint8_t DLC;       // 您想发送数据的长度。
  uint8_t Data[8];   //您想要发送的数据。
} CanTxMsg;
```

StdId

StdId 用来设定标准标识符。它的取值范围为 0 到 0x7FF。

ExtId

ExtId 用来设定扩展标识符。它的取值范围为 0 到 0x1FFFFFFF。

IDE

IDE 用来设定消息标识符的类型。

IDE 值

IDE	描述
CAN_Id_Standard	使用标准标识符
CAN_Id_Extended	使用扩展标识符

RTR

RTR 用来设定待传输消息的帧类型。它可以设置为数据帧或者远程帧。

RTR 值

RTR	描述
CAN_RTR_Data	数据帧
CAN_RTR_Remote	远程帧

DLC

DLC 用来设定待传输消息的帧长度。它的取值范围是 0 到 0x8。

Data[8]

Data[8]包含了待传输数据，它的取值范围为 0 到 0xFF。

先声明一个 CanTxMsg 类型的变量，然后按照自己具体的需要，填充此结构变量就可以发送了（帧结构中其他没有填充的部分由硬件自动完成）。

强调：这里的 StdId 或者 ExtId 是根据自己的实际需要设置的！我们先抛开它们所代表的实际意义，认为它们存在的目的是为了“进攻”。(发送出去让别人过滤)

接下来，我们来看接收，第一个问题就是它是怎样接收的?------过滤!!! 无数的初学者都倒在了这里。

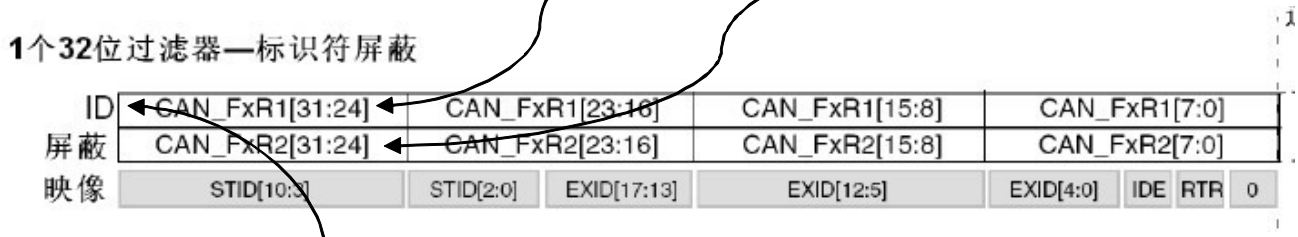
STM32 参考手册中提到：bxCAN 控制器为应用程序提供了 14 个位宽可变的、可配置的过滤器组(13~0)。（互联型有 28 个）。每个过滤器组的位宽都可以独立配置。可以配置成 16 位或者 32 位。过滤器组还可配置为屏蔽位模式或标识符列表模式。TNND,看起来好像很复杂!

先来理解一句话：共有14个过滤器组，**每个过滤器组x由2个32位寄存器，CAN_FxR1和CAN_FxR2组成。**

搞定一个过滤器组，其他的都可以以此类推。

我们先来看 1 个 32 位过滤器-标示符屏蔽的情况 **CAN_FxR1 作 ID，CAN_FxR2 作屏蔽**：

先看图：



这里的 ID 是什么意思？难道就是前面所说的发送数据帧里面的 ID?NO!这里的 ID 和自己发送的帧里面的 ID 没有一点关系。完全是两个东西，在硬件上属于不同的寄存器。**强调：这里的 ID 也是根据自己的实际需要设置的！我们先抛开它们所代表的实际意义，认为它们存在的目的是为了“防守”。(过滤别人发送过来的帧的 ID)**

这里的屏蔽是什么意思？这里的屏蔽和 ID 共同配合完成过滤。

这里的映像是什么意思？映像的意思就是假定收到的帧的 ID 信息。

下面举个例子，一切都会一目了然：

1：假如我们只想收到别人发过来的 ID 为 0x317 的标准数据帧：
0X317 二进制位：011 0001 0111
那么可以这样设置：

CAN_FxR1：0110 0010 111X XXXX XXXX XXXX XXXX X00X （ID）
CAN_FxR2：1111 1111 1110 0000 0000 0000 0000 0110 （屏蔽）

这里是我们设置的想要收到的数据帧的 ID。
这里为 1 的位，意味着收到的数据帧中相应的 ID 位必须和设置的 ID 位一样（必须匹配）。
2：假如我们想收到别人发过来的 ID 为 0x310 到 0x317 的标准数据帧：
那么可以这样设置：

CAN_FxR1：0110 0010 xxxX XXXX XXXX XXXX XXXX X00X （ID）
CAN_FxR2：1111 1111 0000 0000 0000 0000 0000 0110 （屏蔽）

这里的红色 xxx 就代表 000 到 111 的任意组合。
这里为 0 的位，意味着收到的数据帧中的相应位的 ID 不一定非要与设置的 ID 一样（不用关心）。
这里红色部分表明必须是标准数据帧。
3：假如我们想收到别人发过来的 ID 为 0x000 到 0x7FF 的标准数据帧：
那么可以这样设置：

CAN_FxR1：xxxx xxxx xxxX XXXX XXXX XXXX XXXX X00X （ID）
CAN_FxR2：0000 0000 0000 0000 0000 0000 0000 0110 （屏蔽）

接下来看 2 个 32 位过滤器-标识符列表模式，**CAN_FxR1 和 CAN_FxR2 都作为 ID.**
这种情况就很简单了。只有收到的帧的 ID 必须和 CAN_FxR1 或者 CAN_FxR2 完全一样才接收。这样的话，就只能接收两种不同的 ID。

2个32位过滤器—标识符列表

ID	CAN_FxR1[31:24]		CAN_FxR1[23:16]		CAN_FxR1[15:8]		CAN_FxR1[7:0]		
ID	CAN_FxR2[31:24]		CAN_FxR2[23:16]		CAN_FxR2[15:8]		CAN_FxR2[7:0]		
映像	STID[10:3]		STID[2:0]	EXID[17:13]	EXID[12:5]		EXID[4:0]	IDE	RTR	0

举个例子：
如果我们把 CAN_FxR1 和 CAN_FxR2 设置为下面的样子。
CAN_FxR1：0110 0010 1110 0000 0000 0000 0000 0000 （ID）
CAN_FxR2：0000 0001 1110 0000 0000 0000 0000 0000 （ID）

这样的话，就只能过滤出 ID 为 0x317(与 CAN_FxR1 必须一样)和 0x00F(与 CAN_FxR2 必须一样)两

种标准数据帧了。
以上是 32 位模式下标识符屏蔽模式和标识符列表模式下的设置方法。

在 16 位模式下，只不过把两个 32 位寄存器拆成了 4 个 16 位的而已，原理和 32 位模式下是一样的。就不赘述了。

接收数据是通过指向 ***CanRxMsg*** 结构体变量的指针传递的。直接调用 CAN_Receive 即可轻松完成。

```
typedef struct
{
  uint32_t StdId;
  uint32_t ExtId;
  uint8_t IDE;
  uint8_t RTR;
  uint8_t DLC;
  uint8_t Data[8];
  uint8_t FMI;
} CanRxMsg;
```

CanRxMsg 结构与 CanTxMsg 差不多。只是多了一个 FMI 域。

FMI

FMI 设定为消息将要通过的过滤器索引，这些消息存储于邮箱中。该参数取值范围 0 到 0xFF。