哥很郁闷，为了 CAN 研究了不少，看了不少资料，现在我给大家总结一下
先看看工作原理

当 CAN 总线上的一个节点（站）发送数据时，它以报文的形式广播给网络中所有节点，对每个节点来说，无论数据是否是发给自己的，都对其接收。每组报文开头的 11 位字符为标识符，定义了报文的优先级，这种报文格式成为面向内容的编制方案。同一系统中标识符是唯一的，不可能有两个站发送具有相同标识符的报文，当几个站同时竞争总线读取时，这种配置十分重要。

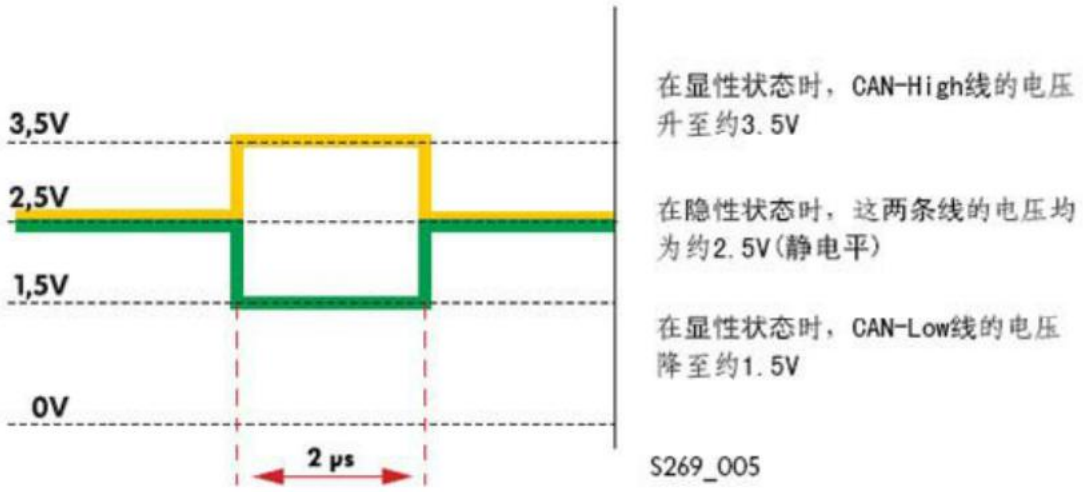
大体的工作原理我们搞清了，但是根本的协议我们还要花一番功夫。下面介绍一个重要的名词，“显性“和”隐性“

在我看到的很多文章里，有很多显性和隐性的地方，为此我头痛不已，最终我把它们彻底弄明白了。

首先 CAN 数据总线有两条导线，一条是黄色的，一条是绿色的。分别是 CAN_High 线和 CAN_Low 线

当静止状态时，这两条导线上的电平一样。这个电平称为静电平。大约为 2.5 伏。这个静电平状态就是**隐形状态，也称隐性电平。也就是没有任何干扰的时候的状态称为隐性状态.**当有信号修改时，CAN_High 线上的电压值变高了，一般来说会升高至少 1V，而 CAN_Low 线上的电压值会降低一个同样值，也是 1v，那么这时候。CAN_High 就 是 2.5v+1v=3.5v,它就处于激活状态了。而 CAN_Low 降为 2.5v-1v=1.5v。

可以看看这个图



由此我们得到

在隐性状态下，CAN_High 线与 CAN_Low 没有电压差，这样我们看到没有任何变化也就检测不到信号。但是在显性状态时，改值最低为 2V，我们就可以利用这种变化才传输数据了。所以出现了那些帧，那些帧中的场，那些场中的位，云云~~~~~~~~~~~

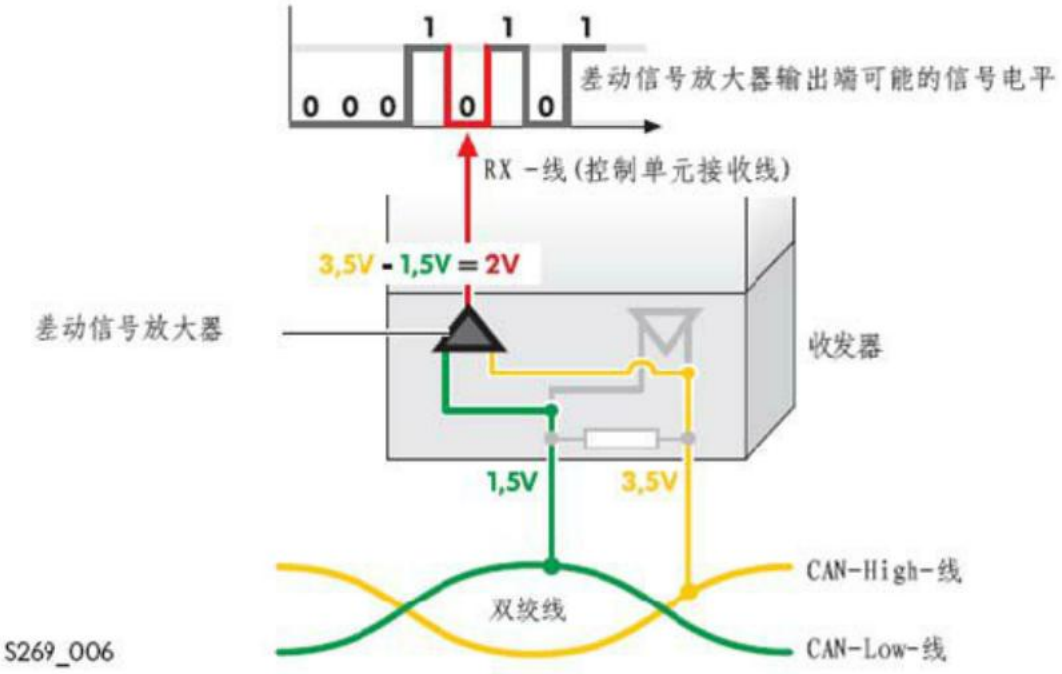
在总线**上通常逻辑 1 表示隐性。而 0 表示显性。这些 1 啊，0 啊，就可以利用起来为我们传数据了。**

利用这种电压差，我们可以接收信号。

一般来说，控制单元通过收发器连接到 CAN 驱动总线上，这个收发器（顾名思义，可发送，可接收）内有一个接收器，该接收器是安装在接收一侧的差动信号放大器。然后，这个放大器很自然地就放大了 CAN_High 和 CAN_Low 线的电平

差，然后传到接收区。如下图

CAN驱动数据总线的差动信号放大器



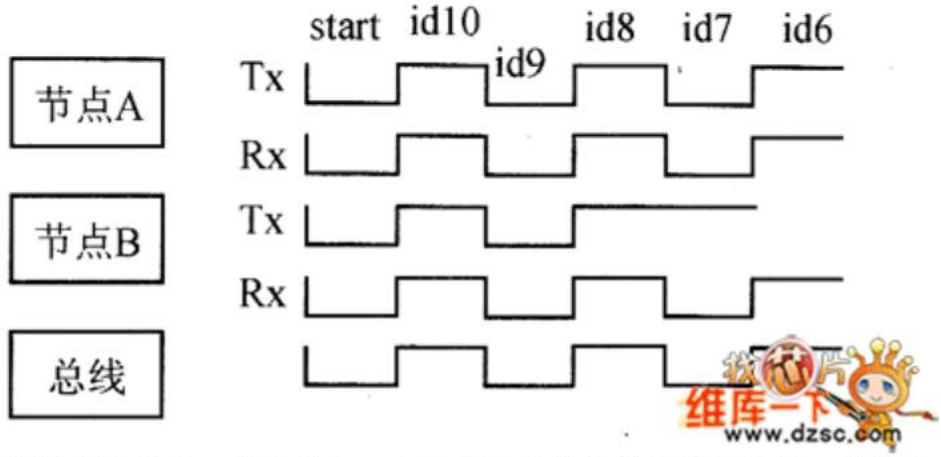
由上图可知，当有电压差，差动信号放大器放大传输，将相应的数据位任可为 0。
下面我们进入重点难点。报文
所谓报文，就是 CAN 总线上要传输的数据报，为了安全，我们要给我们传输的数据报编码定一下协议，这样才能不容易出错，所以出现了很多的帧，以及仲裁啊，CRC 效验。这些都是难点。
识别符的概念。
识别符顾名思义，就是为了区分不同报文的可以鉴别的好多字符位。有标准的，和扩展的。标准的是 11 位，扩展的是 29 位。他有一个功能就是可以提供优先级，也就是决定哪个报文优先被传输，报文标识符的值越小，报文具有越高的优先权。

CAN 的报文格式有两种，不同之处其实就是识别符长度不同，具有 11 位识别符的帧称为标准帧，而还有 29 位识别符的帧为扩展帧，CAN 报文有以下 4 个不同的帧类型。分别是

（1） 数据帧：数据帧将数据从发送器传输到接收器。
（2） 远程帧：总线节点发出远程帧，请求发送具有同一标识符的数据帧
（3） 错误帧：任何节点检测到总线错误就发出错误帧
（4） 过载帧：过载帧用已在先行的后续的数据帧（或远程帧）之间提供一附加的延时

我们先研究数据帧吧。
一，数据帧由 7 个不同位场组成。
这里的位场，就是不同位的组合，这名字起的很烂，让人看了感觉很抽象。我们来看看这些个不同的位场吧。
一开始是一位**帧起始**，也叫 SOF。它用显性位表示，也就是 0.它告诉我们，两个线上有电压差了，也就是有数据了。这个帧起始看起来只有一位，起始不简单了。

为了让所有的分站都同步于发送报文的发送站，好接收数据，有很多要考虑的地方。
然后下一个场是**仲裁场。**这个仲裁场是个难点。但是不要怕，有我在，你会很明白地搞定的。
这个仲裁很抽象，其实在这里就是为了解决一个问题。**如果 2 个或 2 个以上的单元同时开始传送报文，那么就会有总线访问冲突。**那么仲裁机制就是用来根据标识符优先级来一个一个的去掉低级别的数据。我们可以详细的描述这场生动的争抢总线的战斗。
当总线处于空闲状态时呈隐性电平，此时任何节点都可以向总线发送显性电平作为帧的开始。2 个或 2 个以上的节点同时发送开始争抢总线，但是总线只能被一个人抢走。总线只属于一个他。这时候到底怎么决定谁留下，谁滚蛋呢。我们开始考虑，思索，我们以前定义了标识符，标识符有优先级，它越小，它优先级越高。那么怎么实现的呢。看下面把首先搞明白两点，一 下面的图 低波形代表 0，高波形代表 1 二 当隐性碰到显性，就变为显性。
如图所示，节点 A 和节点 B 的标识符的第 IO、9、8 位电平相同，因此两个节点侦听到的信息和它们发出的信息相同。第 7 位节点 B 发出一个“1”，但从节点上接收到的消息却是“0”，为什么呢，因为 A 节点同时发出显性位，让总线也变成显性了，也就是 0。节点 B 会退出发送处于单纯监听方式而不发送数据；节点 A 成功发送仲裁位从而获得总线的控制权，继而发送全部消息。总线中的信号持续跟踪最后获得总线控制权发出的报文，本例中节点 A 的报文将被跟踪。这种非破坏性位仲裁方法的优点在于，在网络最终确定哪个节点被传送前，报文的起始部分已经在网络中传输了，因此具有高优先级的节点的数据传输没有任何延时。在获得总线控制权的节点发送数据过程中，其他节点成为报文的接收节点，并且不会在总线再次空闲之前发送报文



在这逐位的比较中，最终节点 B 因为第七位的偏差丢掉了总线。从此单纯监听，江山就拱手让给了节点 A 了。**这就是仲裁机制**
这里我们涉及到总线值
总线值
总线有二个互补的逻辑值：“显性”或“隐性”。“显性”位和“隐性”位同时传送时，总线的结果值为“显性”。比如，在总线的“写与”执行时，逻辑 0 代表“显性”等级，逻辑 1 代表“隐性”等级。

上面我们说过，报文有两种格式，标准和扩展。这里，不同的格式仲裁场是不一

样的。标准格式下，仲裁场由 11 位识别符和 RTR 位组成。但在扩展格式里，包括 29 位识别符，SRR 位，IDE 位，RTR 位。
RTR 位。Remote Tranmission Request BIT 全称为 远程发送请求位。它在数据帧里必须为显性 0 ，但在远程帧里为隐性 1。我晕，为什么这么搞呢，不急，先留着这个问题。
SRR 位，替代远程请求位，SRR 是一隐性位，也就是 1，它在扩展格式的标准帧 RTR 位位置，那么**标准帧怪不得优先于扩展帧了，**因为在传输完 11 位标识符之后（扩展帧的后 18 位在最后发送，先发送 11 位标识符），轮到标准帧的 RTR 位和扩展帧的 SRR 位了。这时候，标准帧的 RTR 为显性，而扩展帧 SRR 为隐性，这样，总线自然就被标准帧占据。同时上面那个问题，也一目了然了，CAN 总线协议设计者，肯定是设计了数据帧优先于远程帧所以
IDE 全称识别符扩展位(Identifier Extension Bit)，它属于
扩展格式的仲裁场
标准格式的控制场
标准格式的 IDE 位为显性，扩展格式里 IDE 位为隐性。这样扩展格式的数据帧优先级又落下了一截。
控制场
控制场有 6 位组成。标准的跟扩展的又不同。标准的格式里的帧先是 IDE 位，然后保留位 r0，然后数据长度代码（共四位，分别是 DLC3，DLC2，DLC1，DLC0）而扩展格式里，IDE 替换为 r1 保留位，其余不变。
数据长度代码指示了数据场里的字节数量。
数据场：
数据场由发送数据组成，可以为 0~8 个字节，从高位开始（MSB）先发送。
CRC 场
包括 CRC 序列，和 CRC 界定符。
这个 CRC 序列又是一个难点，具体什么是 CRC 序列呢，

CRC 即循环冗余校验码（Cyclic Redundancy Check）：是数据通信领域中最常用的一种差错校验码，其特征是信息字段和校验字段的长度可以任意选定。

生成 CRC 码的基本原理：任意一个由二进制位串组成的代码都可以和一个系数仅为‘0’和‘1’取值的多项式一一对应。例如：代码 1010111 对应的多项式为 x6+x4+x2+x+1，而多项式为 x5+x3+x2+x+1 对应的代码 101111。

参考一下下面的例题.自己再领悟一下吧！已知信息位为1100，生成多项式G(x) = x3+x+1，求CRC码。 M(x) = 1100 M(x)*x3 = 1100000 G(x) = 1011 M(x)*x3 / G(x) = 1110 + 010 /1011 R(x) = 010 CRC码为： M(x)*x 3+R(x)=1100000+010 =1100010 其原理是：CRC码一般在k位信息位之后拼接r位校验位生成。编码步骤如下： （1）将待编码的k位信息表示成多项式 M(x)。 （2）将 M(x)左移r 位，得到 M(x)*xr 。 （3）用r+1位的生成多项式G(x)去除M(x)*xr 得到余数R(x)。 （4）将M(x)*xr 与R(x)作模2加，得到CRC码。

应答场

应答场（ACK）长度为2个位，包含应答间隙和应答界定符，在ACK场里，发送站发送两个隐性位。当接收器正确接收到有效地报文，接收器就会在应答间隙期间(发送 ACK 信号)向发送器发送一显性位以示应答。

二 **远程帧**

通过发送远程帧，总线的节点发出远程帧，请求以前发送给它数据帧的节点再发送一遍。具体发送哪个数据帧，由远程帧的标识符决定。

远程帧的RTR是隐性的。没有数据场，其余都与数据帧相同。

三

错误帧

错误帧由两个不同的场组成，第一个场是不同站提供的错误标志的叠加，第二个场是错误界定符。

1 错误标志

有两种形式的错误标志，主动地和被动的。这就让人很明白了。也就是说主动发出错误的节点发出错误帧时，就是主动地错误标志，而接收错误帧的节点，就发出被动错误标志。

主动地错误标志由6个连续的显性位组成。

被动的错误标志由6个连续的隐形位组成，除非被其他节点的显性位重写。

检测到错误条件的错误激活的站通过发送主动错误标志指示错误。这个错误帧也可以看做有着跟数据帧类似的场结构，错误标志的形式显然破坏了从帧起始到CRC 界定符的位填充规则（检测到 5 个相同的位，就插入一个补充位，但是错误帧却有 6 个相同的位，显然破坏了）。或者破坏了 ACK 场或帧结尾场的固定形式。所有其他的站由此检测到错误条件。并于此同时发送错误标志。并且假如有很多站都有自己的错误发送，它们会都发送主动错误标志，这种显性标志显然被叠加在一起。

(2)错误界定符

错误界定符包括8个“隐性”的位。

错误标志传送了以后，每一个节点就发送一个“隐性”的位，并一直监视总线直到检测出一个“隐性”的位为止，然后就开始发送其余7个“隐性”位。

四 过载帧

过载帧包括2个位场：过载标志和过载界定符

有三种过载的情况，这三种情况都会引起过载标志的发送

1 接收器的内部情况（此接收器对于下一数据帧或远程帧需要一定的延时）

这种情况引发的过载帧只允许起始于所期望间歇的第一个位时间。

2 在间歇的第1和第2字节检测到一个显性位

这里有个间歇的概念。我们可以讲讲。间歇属于帧间空间的一部分。它包含三个隐性位。间歇期间，所有的站不允许传送数据帧或远程帧。它唯一要做的就

是标示一个过载条件。

3 如果 CAN 节点在错误界定符或过载界定符的第 8 位采样到一个显性位，有了上面的情况，则节点会发送一个过载帧。错误计数器不会增加。

对于情况 2，3 引发的过载帧应起始于所检测到显性位之后的位。

通常为了延时下一个数据帧或远程帧，两种过载帧均可产生。

过载标志

由 6 个显性位组成。过载标志的所有形式和主动错误标志一样。

过载标志的形式破坏了间歇场的固定形式，因此，所有其他的站都检测到过载条件并于此同时发出过载标志。

过载界定符 8 个隐形位

帧间空间

数据帧（或远程帧）与先行帧的隔离是通过帧间空间实现的。无论此先行帧类型如何。所不同的是过载帧与错误帧之间没有帧间空间。多个过载帧之间也不是由帧间空间隔离的。

帧间空间包括间歇，总线空闲的位场。如果错误被动的站已作为前一报文的发送器，则其帧空间除了间歇，总线空闲外，还包括称作挂起传送的位场。

图9.13 非“错误认可”帧间空间

对于作为前一报文发送器的“错误认可”的节点，其帧间空间如图9.14所示：

图9.14 “错误激活”帧间空间

（1）间歇（Intermission）

间歇包括3个“隐性”的位。间歇期间，所有的节点均不允许传送数据帧或远程帧，唯一要做的是标示一个过载条件。

如果 CAN 节点有一报文等待发送并且节点在间歇的第三位采集到一显性位，则此位被解释为帧的起始位，并从下一位开始发送报文的标识符首位，而不用首先发送帧的起始位或成为一接收器。

（2）总线空闲（Bus Idle）

总线空闲的时间是任意的。只要总线被认定为空闲，任何等待发送报文的节点就会访问总线。在发送其他报文期间，有报文被挂起，对于这样的报文，其传送起始于间歇之后的第一个位。