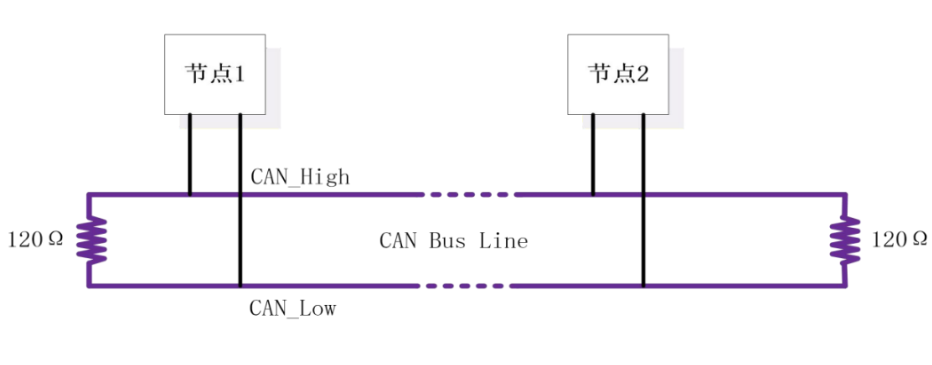
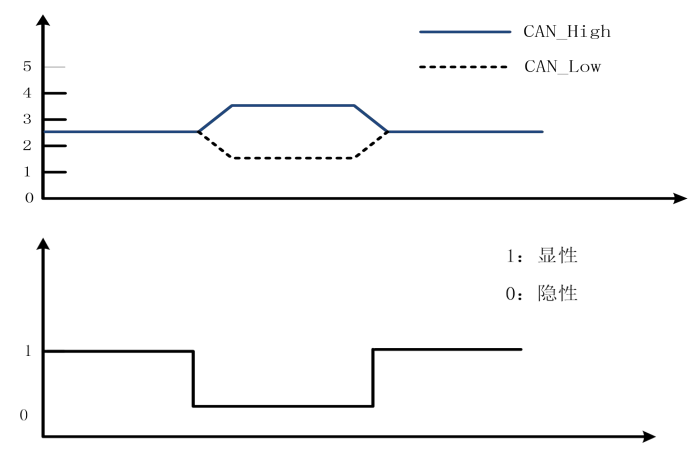
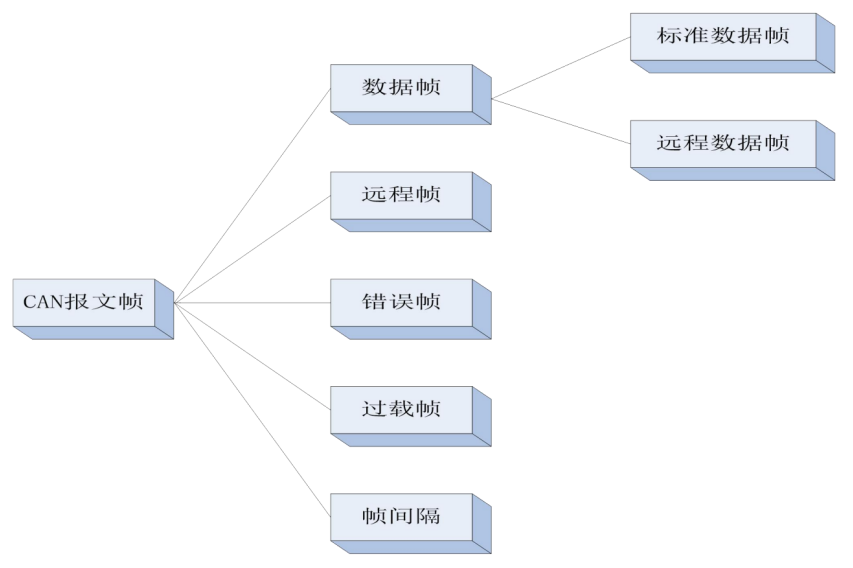
CAN总线是一种常用的总线，对于刚开始接触CAN总线的，面对着各式各样的资料，可能不知道从何看起，今天科普一下CAN总线的基础知识。CAN2.0协议分为A版本和B版本，A版本协议为11位标识符（标准帧），B版本在兼容11位ID标识符的同时，向上扩展到29位ID标识符。

**CAN总线的物理电平**  
对 CAN 总线的协议了解之前，先介绍一下 CAN总线的电平，CAN总线的基本状态有两种，分别为“显性”和“隐性”，也就是“逻辑 0”和“逻辑 1”，如图：图中两个 CAN 总线节点分别接在 CAN 总线上，**两个终端为什么要接两个 120Ω的终端电阻？**

首先解释一下“终端电阻”这个名词，终端电阻是一种信号在传输过程中遇到的阻碍，高频信号传输时，信号波长相对传输线较短，信号在传输线终端会形成反射波，干扰原信号，所以需要在传输线末端加终端电阻，使信号到达传输线末端后不反射。对于低频信号则不用。在长线信号传输时，一般为了避免信号的反射和回波，也需要在接收端接入终端匹配电阻。终端匹配电阻值取决于电缆的阻抗特性，特别注意的是与电缆的长度无关。RS-485/RS-422 /CAN 总线一般采用双绞线（屏蔽或非屏蔽）连接，终端电阻一般介于100至140Ω之间，典型值为120Ω。在实际配置时，在电缆的两个终端节点上，即最近端和最远端，各接入一个终端电阻，而处于中间部分的节点则不能接入终端电阻，否则将导致通讯出错。现在继续介绍 CAN 总线的逻辑，CAN 总线传输数据时，依靠的是差分电平，因为 CAN 总线是双绞线，所以在没有数据传输时，两条线的电压相同，均为2.5V，差值为 0，所以为“隐形”，若是有数据传输时，两条线就会出现电压不通的情况，CAN\_H 为 3.5V、CAN\_L 为 1.5V，电压差为 2V，所以为“显性”。现在用两个等式来对 CAN 的电平做一个总结，那就是：CAN 总线为隐性 = 无电压差 = 逻辑 1CAN 总线为显性 = 有电压差 = 逻辑 0

**CAN报文帧详解**  
在了解 CAN 硬件之后，知道了逻辑“0”和逻辑“1”，CAN 报文帧就是由这些0和1组成。由图可以看出，CAN-bus 的通信帧共分为五种帧：**数据帧、远程帧、错误帧、过载帧和帧间隔**，数据帧又有标准帧和扩展帧两种。

**数据帧和遥控帧**

在 CAN 协议中，数据帧和遥控帧有着诸多相同之处，将数据帧和遥控帧放在一起来讲。数据帧是指包含了要传输的数据的帧，作用是承载发送节点要传递给接收节点的数据。**遥控帧的作用：**请求其它节点发出与本遥控帧具有相同 ID 号的数据帧。发起方发起特定ID的遥控帧，并且只发送ID部分，那么与其 ID相符的终端设备就有义务在后半段的数据部分接管总线控制权并发送自己的数据。

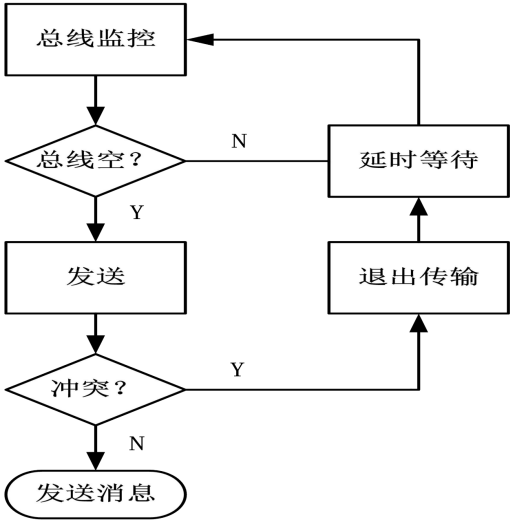
例如中控机需要定时获取某个节点的数据（例如转速计的实时转速、油量计的实时油量等），可以向总线发送远程帧；相应节点在接收判断帧 ID与自己相符、并且是远程帧的情况下，就可以将自己的实时数据发送到总线上；这样中控机就获取到了相关节点的实时数据。

**远程帧最大的好处就是只需要一帧的时间就能完成一次双向交互。**

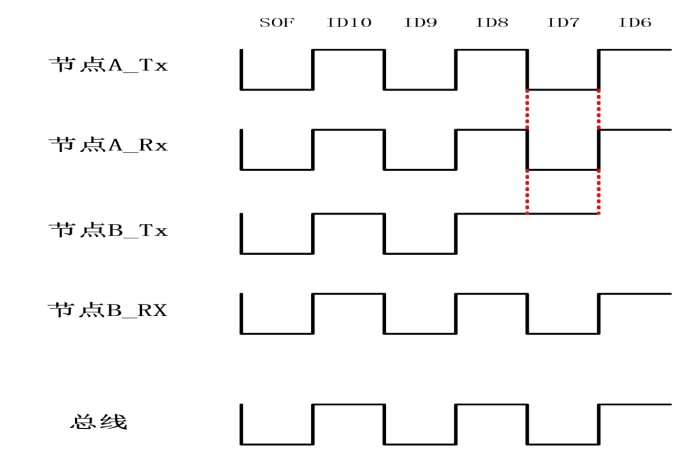
两种帧格式由帧起始、仲裁帧、控制端、数据段、CPC 段、ACK 段、帧结束， 不同的段有不同的用途。

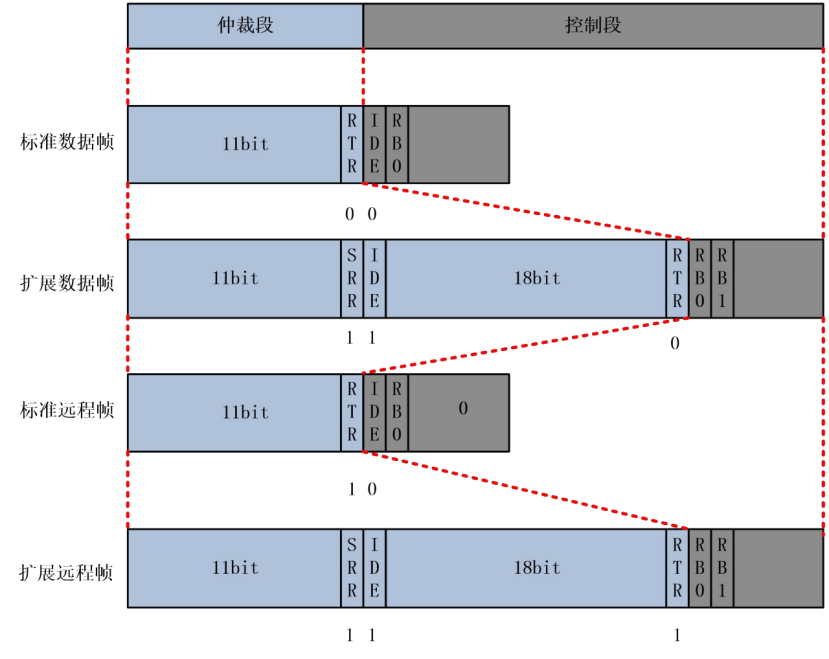
**起始段**

帧的最开始的一位是帧起始，也叫 SOF（Start Of Frame），SOF 恒为显性位，即逻辑0。帧起始表示CAN\_H和CAN\_L上有了电位差，也就是说，一旦总线上有了 SOF 就表示总线上开始有报文了。

**仲裁段**  
CAN 总线有一个特点，就是所有节点均可以获得总线的控制权并且向 CAN 总线发送数据，当同时有 2 个或 2 个以上的设备要求发送数据时，就会产生总线冲突，这时就需要判断一个先后顺序，而仲裁段就是判断先后顺序的裁判员，仲裁段会对 CAN 数据进行比较，ID 码值越小的数据越具有优先权，从而使具有高优先级的数据不受任何损坏地传输。

仲裁段是如何仲裁的？各节点在向总线发送电平的同时，也对总线上的电平读取，并与自身发送的电平进行比较，如果电平相同继续发送下一位，不同则停止发送退出总线竞争。剩余的节点继续上述过程，直到总线上只剩下1个节点发送的电平，总线竞争结束，优先级高的节点获得总线的控制权。

假设 CAN 总线上有两个节点，节点A和节点 B，节点A 的 ID为 10101000000，节点 B 的ID为 101100000，当节点 A 和节点 B 同时向CAN 总线发送数据时，如下图，当发送到 ID7 时，节点 A 仲裁成功，从而获得 CAN 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，在这里值得注意的是，CAN 总线的这种仲裁方式优点在于，无论是总线的控制权在哪个节点，CAN 总线传输的报文已经在总线上传输了。因此，CAN 总线具有高优先级的节点的数据在传输时，没有任何延迟，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。

**仲裁段是用来判定一帧报文优先级的依据**，仲裁段中的 ID 号也是实现报文过滤机制的基础。那么不同帧的优先级是怎么体现的呢？**RTR 位：**Tranmission Request Bit （远程发送请求位），RTR 用来区分该帧是数据帧还是远程帧。当 RTR为逻辑 0 时，代表该帧为数据帧；当 RTR 为逻辑 1 时，代表该帧为远程帧。由图可以看出，当四种传输帧时，由于标准数据帧和扩展数据帧的RTR 都为 0，可以看出数据帧的优先级大于远程帧的优先级。RTR 的作用是在前 11 位 ID 号相同的情况下，保证数据帧优先级高于远程帧。

**SRR 位：**Substitutes for Remote Requests Bit（替代远程请求位）在扩展帧（数据帧或遥控帧）中，SRR 恒为隐性位 1，并且可以发现，扩展帧的隐性 SRR 位正好对应标准数据帧的显性 RTR 位，可以看出标准帧的优先级高于扩展帧。SRR 位的作用，在前 11 位 ID 号相同的情况下，标准数据帧的优先级高于扩展数据帧。

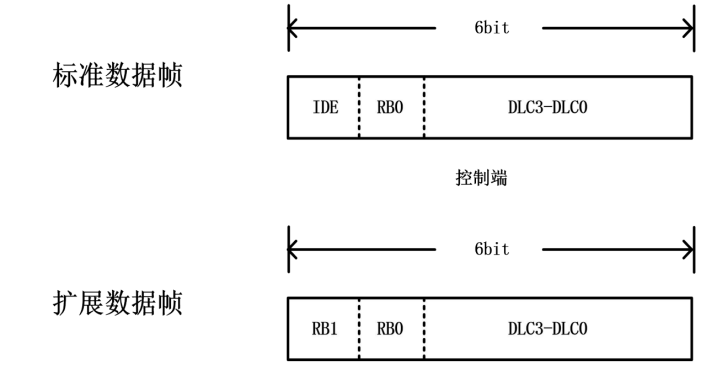
**IDE 位：**Identifier Extension Bit（标识符扩展位），IDE 用来区分该帧是标准帧帧还是扩展帧。当 IDE 为逻辑 0 时，代表该帧为标准帧；当 IDE 为逻辑 1 时，代表该帧为扩展帧。扩展帧 IDE 位和标准帧 IDE 位位置对应，可以看出，标准遥控帧的优先级一定高于扩展遥控帧。IDE 位的作用，在前 11 位相同的情况下，标准数据帧的优先级高于扩展数据帧。

**在 ID 号前 11 位相同的情况下：**

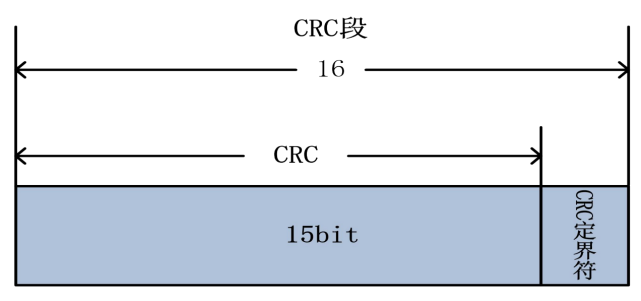
RTR ：保证数据帧优先级高于远程帧；

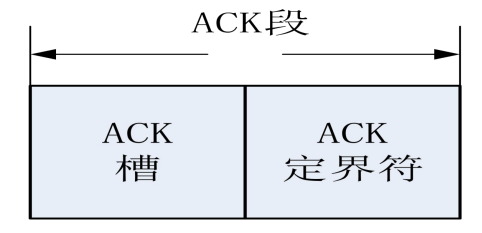
SRR ：保证标准数据帧的优先级高于扩展数据帧。

IDE ：保证标准遥控帧的优先级高于扩展远程帧。

**控制段**  
控制段由6个位组成，标准格式和扩展格式的控制场格式不同。标准格式里的帧包括：数据长度代码、IDE 位（为显性位）及保留位 RB0。扩展格式里的帧包括：数据长度代码、两个保留位 RB0 和 RB1。其保留位必须发送为显性，但是接收器认可“显性”和“隐性”位的任何组合，其结构如图所示：数据帧长度代码（DLC），指示了数据场里的字节数量。其中：d—“显性”， r—“隐性”， 数据帧允许的数据字节数为{0，1，...，7，8}。其他的数值不允许使用。

**数据段**  
数据段由数据帧里的发送数据组成。它可以为 0~8 个字节，每字节包含了 8 个位，首先发送最高有效位（MSB）。

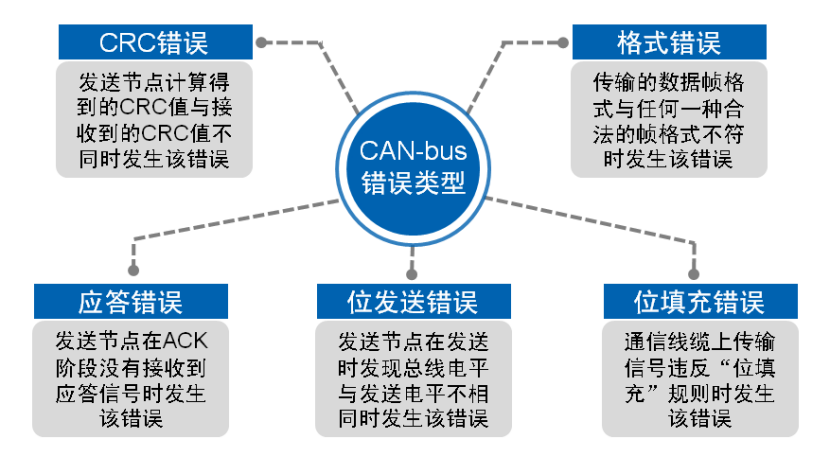
**CRC校验段**  
CRC 包含 CRC 校验序列和 CRC 界定符（恒为隐形，即逻辑 1），通过多项式生成 CRC 值，比较发送节点与接受节点 CRC 是否一致，来确保帧的有效性，计算范围包括发送节点【帧起始、仲裁场、控制场、数据场】是否与接收节点【帧起始、仲裁场、控制场、数据场】是否一致。传统 CAN 使用的是 CRC15 的算法，这个通过 CAN 的数据结构也可以看出，CRC段的长度就是 15 位。CAN FD 之所以有两种是因为 CAN FD 的数据长度是可变的， 针对不同的数据长度使用的方法不同，低于 16 字节的使用的是 CRC17，高于 16 字节的使用的是 CRC21。

**ACK段**  
ACK段包含 ACK 槽和 ACK 界定符两个位。

发送节点在 ACK 段发送两个隐性位，即发送方发出的报文中 ACK 槽为隐性 1；接收节点在接收到正确的报文之后会在 ACK 槽发送显性位 0，通知发送节点正常接收结束。所谓接收到正确的报文指的是接收到的报文没有填充错误、格式错误、CRC 错误。

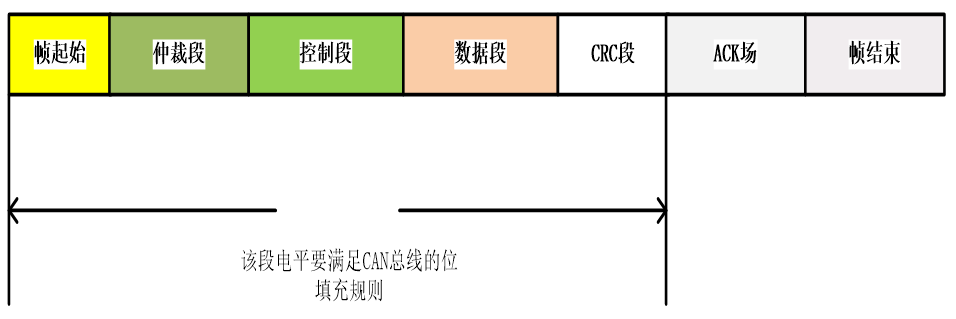
以标准数据帧为例来分析 ACK 段的工作方式，Node\_A 为发送节点， Node\_B 为接收节点。Node\_A 在 ACK 段发送两个隐性位 1。Node\_B 正确接收到这一报文后，在 ACK 段的 ACK 槽中填充了一个显性位 0。注意，这个时候 Node\_A 回读到的总线上的额电平为显性 0，于是这个时候，Node\_A 就知道自己发出去的报文至少有一个节点正确接收了。

**结束段（EOF）**  
每一个数据帧或远程帧均由 7 个连续的隐性位 1 组成。帧结束强制不遵守位填充特性，表达出明显的结束标识。这样接收节点可以正确检测到一个帧的传输结束。

**错误帧种类**  
错误帧种类分为5种，分别是：**位发送错误、ACK错误、位填充错误、CRC错误、格式错误。**

**位发送错误**  
节点将自己发送到总线上的电平与同时从总线上回读到的电平进行比较，如果发现二者不一致，那么这个节点就会检测出一个位错误。实际上所谓“发出的电平与从总线上回读的电平不一致”，指的就是节点向总线发出隐性位，却从总线上回读到显性位或者节点向总线发出显性位，却从总线上回读到隐性位这两种情况。

**ACK错误**  
ACK在ACK段中讲解过，按照CAN协议的规定，发送节点Node\_A在一帧报文（数据帧或者遥控帧）发出之后，如果接收节点Node\_B成功接收了该帧报文，那么接收节点Node\_B就要在该帧报文ACK槽对应的时间段内向总线上发送一个显性位来应答发送节点Node\_A。这样发送节点Node\_A就会在ACK槽时间段内从总线上回读到一个显性位。因此：当发送节点Node\_A在ACK槽时间段内没有回读到显性位，那么发送节点Node\_A就会检测到一个ACK应答错误。这表示没有一个节点成功接收该帧报文，此时CAN总线认为是ACK应答错误

**位填充错误**  
帧起始到CRC校验之前的物理上电平不允许有6个连续的相同电平，发送器只要检测到位流中有5个连续相同逻辑的位，便会自动在下一位插入一个相反的电平。从帧起始到CRC之间，接收节点检测到有6个连续相同的位电平时，也就是违反5位相同位插入1位相反位的“位填充”原则；因为ACK域和帧结束域电平固定，也无需填充；帧起始、仲裁域、控制域、数据域以及CRC校验和域，均通过位填充方法编码。位填充是指：无论何时，发送器只要检测到位流中有5个连续相同逻辑的位，便会自动在位流中插入一个补码位。

举例来说，如果连续5个显性位，则在5个显性位之后自动插入1个隐性位，接收器会自动删除这个插入的填充位。数据帧或远程帧的剩余位域（CRC界定符、应答域和帧结尾域）形式固定，不填充；错误帧和过载帧也不填充。

**那么位填充规则的作用是什么呢？**

**原因有两点：**

1.CAN网络同步需要足够多的上升沿，这是CAN协议规定位填充的目的之一。

2.确保数据帧不会被当作错误帧（由6个连续的显性或隐性位组成）、确保正确识别帧结束标志（7个连续隐性位）。

**CRC错误**  
发送端送出的CRC序列由发送器算出，接收器执行同样的CRC算法，若计算结果与接收到的CRC序列不符，则认为CRC错误。

**帧格式错误**  
**主动错误状态：**处于主动错误状态的节点（可能是接收节点也可能是发送节点）在检测出错误时，发出主动错误标志。如果发出主动错误帧的节点是发送节点，这个情况下就相当于：刚刚发送的那一帧报文发错了，现在破坏掉它（发送主动错误帧），你们不管收到什么都不算数；如果发出主动错误帧的节点是接收节点，这个情况就相当于：刚刚收报文的时候发现了错误，不管你们有没有发现这个错误，现在主动站出来告诉大家这个错误，并把这一帧报文破坏掉（发送主动错误帧），刚才你们收到的东西不管对错都不算数了。处于主动错误状态，说明这个节点目前是比较可靠的，出现错误的原因可能不是它本身的问题，即刚刚检测到的错误可能不仅仅只有它自己遇到，正是因为这一点，整个总线才相信它报告的错误，允许它破坏掉发送中的报文，也就是将这一次的发送作废。

**被动错误状态：**错误比较多，很可能错误是人为导致的，通知其他节点有错但是不干扰他们正常收发数据，也不要求重发，同时不能连续发送了，得再插入8位隐性位的“延迟传送”段；这样是为了让其他正常节点（处于主动错误）优先使用总线。被动错误的节点很可能存在硬件故障，不能让它拖累整个网络；过载帧：接收节点向总线上其它节点报告自身接收能力达到极限的，可以这样理解：接收节点Node\_A接收报文的能力达到极限了，于是Node\_A就会发出过载帧来告诉总线上的其它节点（包括发送节点），接收节点Node\_A已经没有能力处理你们发来的报文了。过载帧包括：过载标志和过载界定符两个部分。过载标志：连续6个显性位。过载界定符：连续8个隐性位。与错误帧类似，过载帧中有过载帧重叠部分，且形成过载重叠标志的原因与形成错误帧中的错误重叠标志的原因是相同的。

**那么怎么通俗的理解过载帧呢？**

接收节点Node\_A达到接收极限时，就会发出过载帧到总线上，显然，过载标志的6个连续显性位会屏蔽掉总线上其它节点的发送，也就是说这个时候Node\_A通过发送过载帧的方式来破坏其它节点的发送，这样在Node\_A发送过载帧期间，其它节点就不能成功发送报文，于是就相当于把其它节点的发送推迟了，也就是说Node\_A在其发送过载帧的这段时间得以“休息”。

**帧间隔：**用于将数据帧或远程帧和他们之前的帧分离开，但过载帧和错误帧前面不会插入帧间隔。也就是说数据帧（或者远程帧）通过插入帧间隔可以将本帧与先行帧（数据帧、远程帧、错误帧、过载帧）分隔开来。