CAN总线学习笔记（2）- CAN协议数据帧、遥控帧、错误帧

专辑是依照瑞萨公司《CAN入门书》的组织思路来学习CAN通信的相关知识，并结合网上相关资料以及学习过程中的领悟整理成的笔记。希望对初学者有所帮助。

01 CAN 协议中的帧

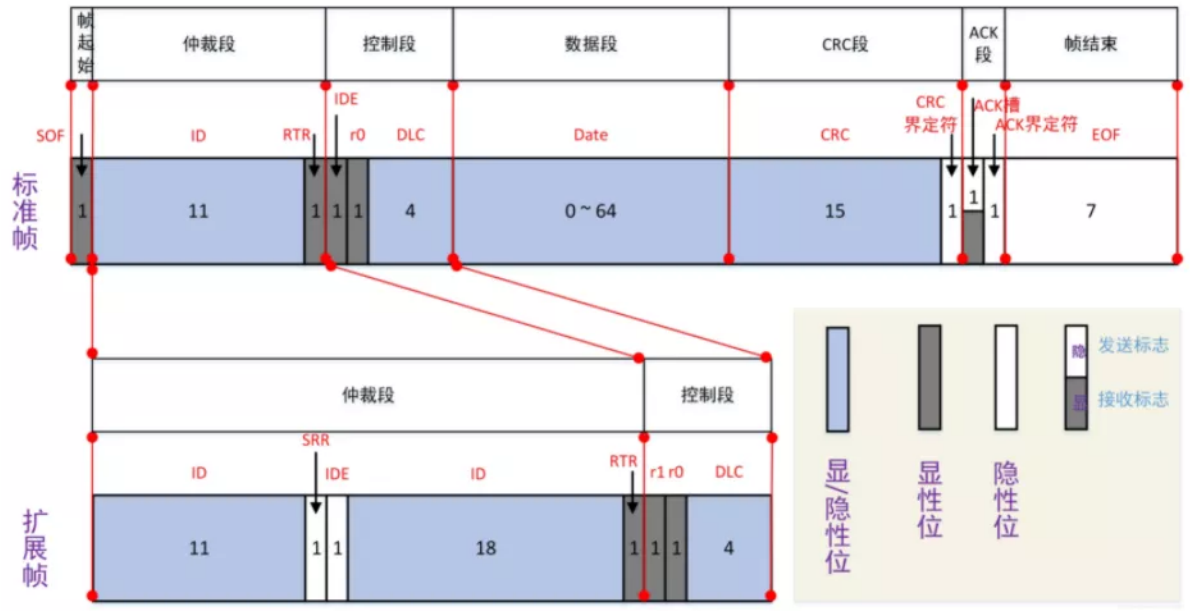
在了解CAN总线的通信机制之前，首先需要了解CAN协议中五种类型的帧结构：

- 数据帧
- 遥控帧
- 错误帧
- 过载帧
- 帧间隔

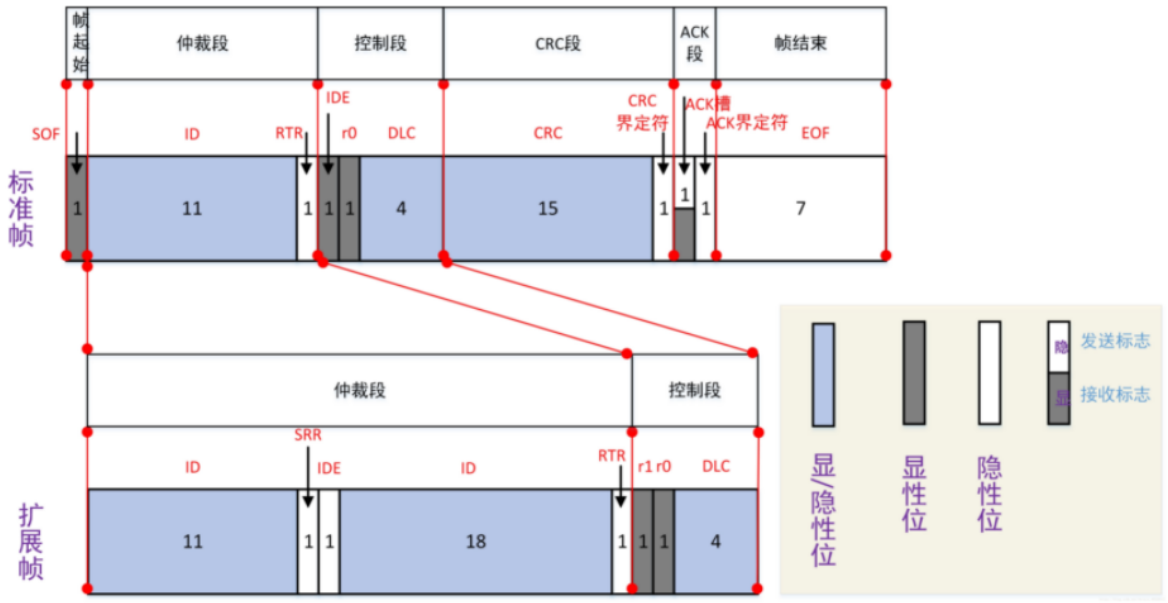
在讲述五种帧结构的过程中，穿插讲述CAN总线的通信机制。

02 数据帧与遥控帧

在CAN协议中，数据帧和遥控帧有着诸多相同之处，所以，在这里，我们将数据帧和遥控帧放在一起来讲。顾名思义，所谓数据帧，就是包含了我们要传输的数据的帧，其作用当然也就是承载发送节点要传递给接收节点的数据。而遥控帧的作用可以描述为：请求其它节点发出与本遥控帧具有相同ID号的数据帧。比如：在某一个时刻，节点Node_A向总线发送了一个ID号为ID_2的遥控帧，那么就意味着Node_A请求总线上的其他节点发送ID号为ID_2的数据帧。节点Node_B能够发出ID号为ID_2的数据帧，那么Node_B就会在收到Node_A发出的遥控帧之后，立刻向总线上发送ID号为ID_2的数据帧。数据帧的帧结构如下图所示，包含七个段:帧起始、仲裁段、控制段、数据段、**CRC**段、**ACK**段、帧结束。



遥控帧相比于数据帧，从帧结构上来看，只是少了数据段，包含六个段：帧起始、仲裁段、控制段、**CRC**段、**ACK**段、帧结束。



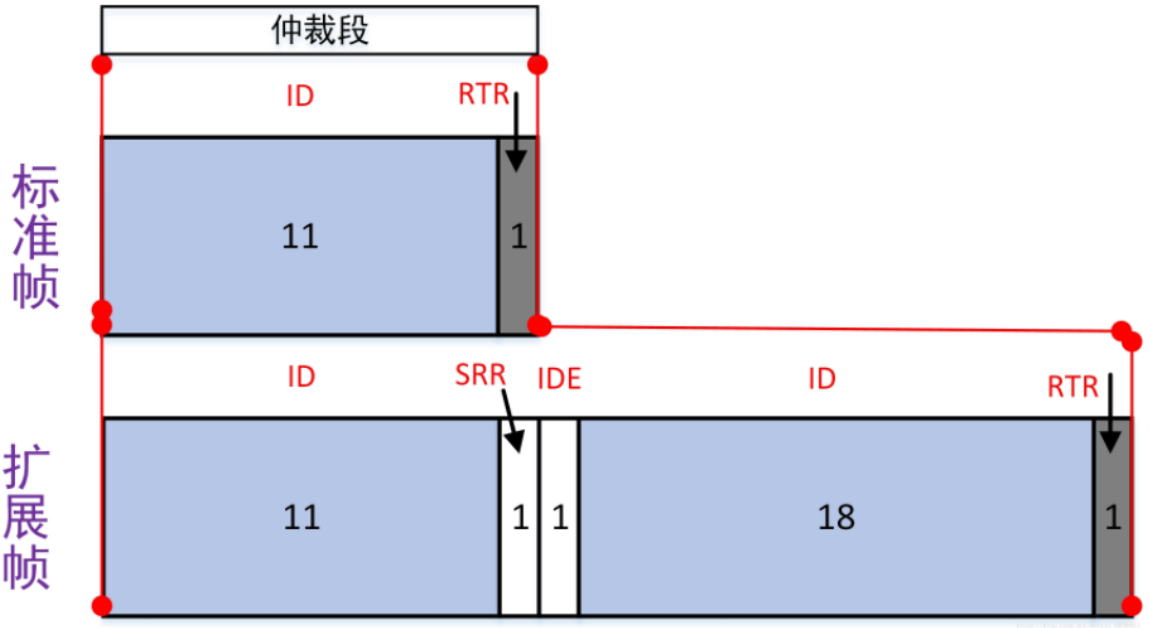
数据帧和遥控帧都分为标准帧（CAN 2.0 A）和扩展帧（CAN 2.0 B）两种结构。遥控帧相比于数据帧除了缺少数据段之外，遥控帧的**RTR**位恒为隐性**1**，数据帧的**RTR**位恒为显性**0**。

2.1 帧起始

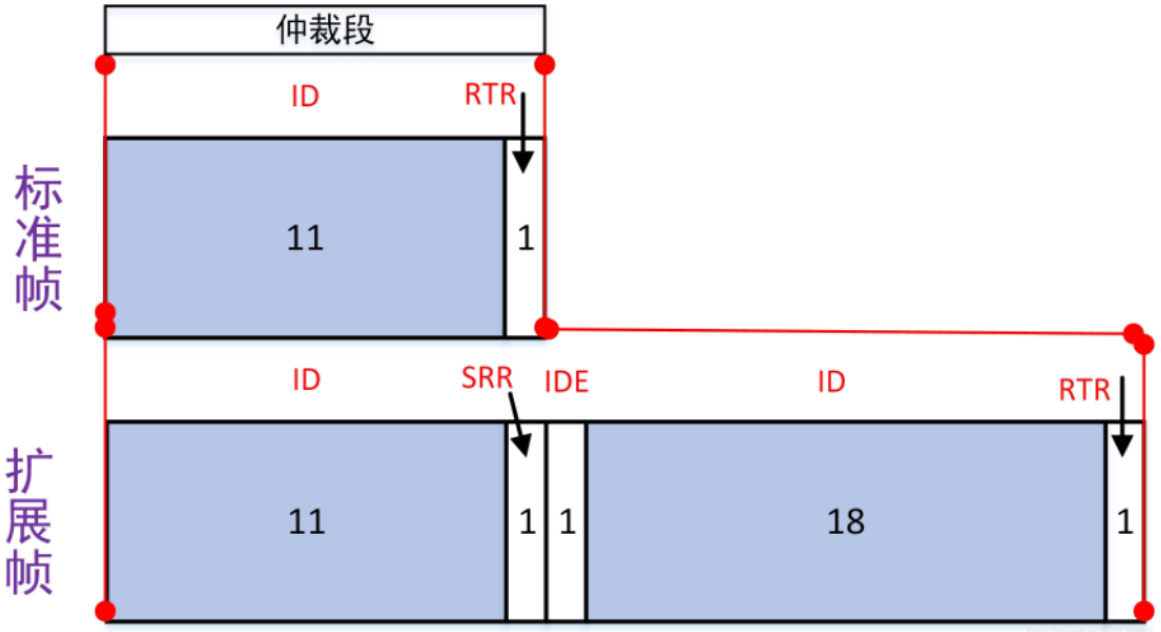
帧的最开始的一位是帧起始，也叫SOF（Start Of Frame），SOF恒为显性位，即逻辑0，帧起始表示CAN_H 和 CAN_L上有了电位差，也就是说，一旦总线上有了SOF就表示总线上开始有报文了。

2.2 仲裁段

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。仲裁段由以下几个部分组成，数据帧仲裁段：



遥控帧仲裁段：

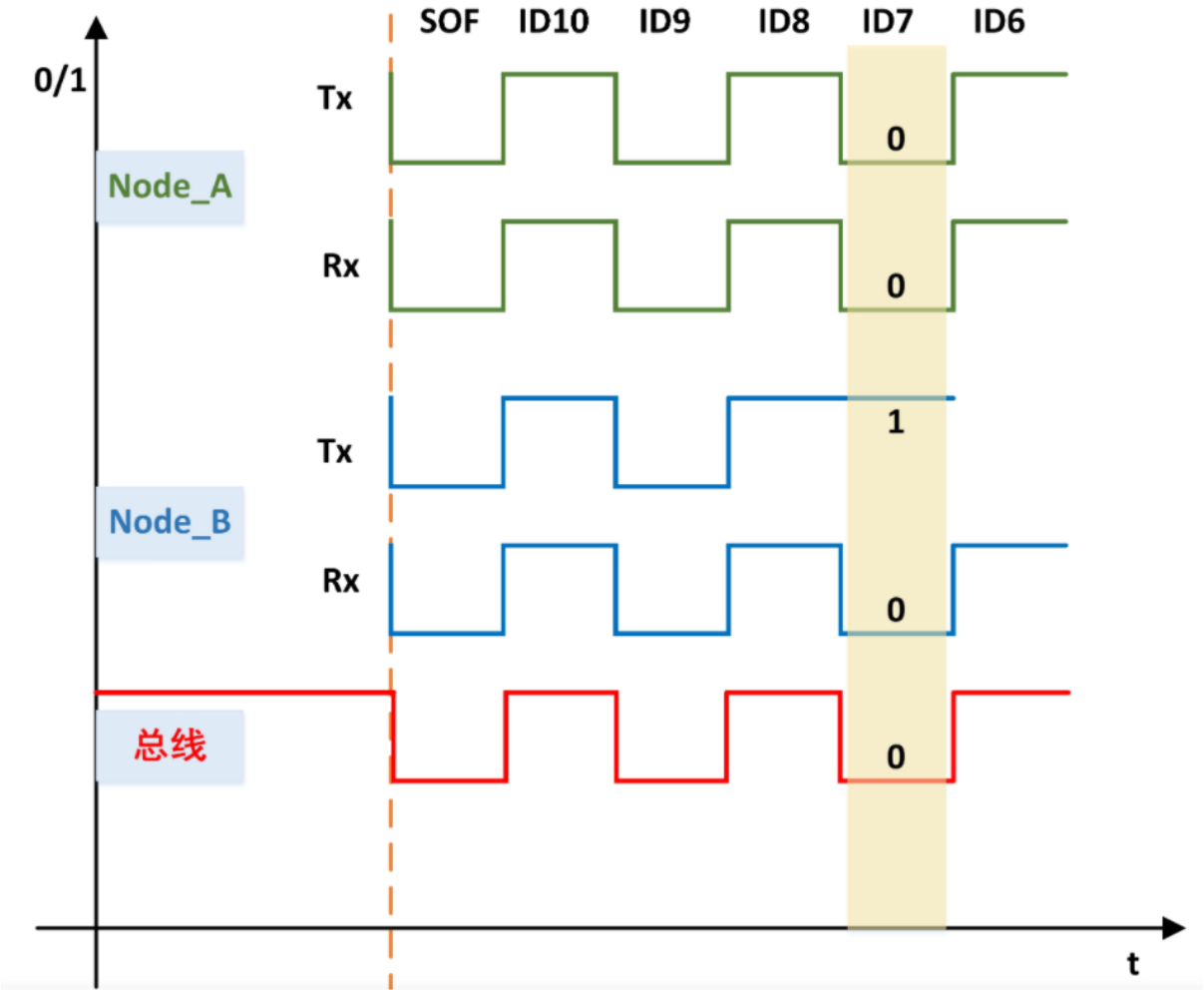


可以看到相比于数据帧仲裁段RTR位恒为显性0，遥控帧仲裁段的RTR位恒为隐性1。

2.2.1仲裁过程

在CAN总线通信中，有一种回读机制：指的是节点在向总线上发送报文的过程中，同时也对总线上的二进制位进行“回读”。通过这种机制，节点就可以判断出本节点发出的二进制位与总线上当前的二进制位是否一致。还有一种叫做线与机制：指的是在总线上，显性位能够覆盖隐性位。举个例子:在某一个时刻，节点Node_A向总线发送了一个显性位0，Node_B向总线发送了一个隐性位1，那么在该时刻，总线上的电平为显性0。下面将以标准数据帧的一个例子来分析CAN总线的非破坏性逐位仲裁机制。

一条CAN总线上有Node_A 和 Node_B两个节点，在总线空闲时，总线上为隐性电平，就在这个时候Node_A 和 Node_B 这两个节点同时向总线上发送数据，如下图：



从图中可以看出，在Node_A 和 Node_B 传输数据前，总线处于空闲状态，为隐性电平1，这也就意味着，此时总线上的任意节点都可以向总线发送数据。在某一时刻，Node_A 和 Node_B两个节点同时向总线上发送数据。按照线与机制，总线上的电位为：

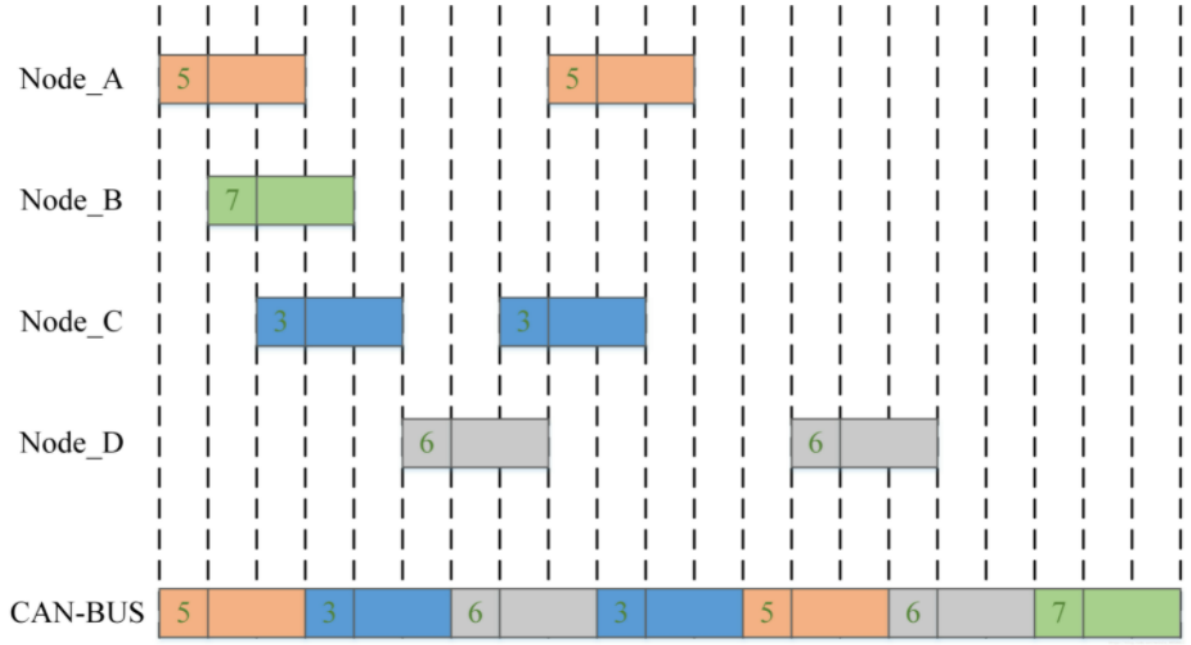
节点/ID号	ID10	ID9	ID8	ID7	ID6	...
Node_A	1	0	1	0	1	...
Node_B	1	0	1	1	...	...
总线	1	0	1	0	1	...

在Node_A和Node_B两个节点向总线发送数据时，他们同时回读总线上的电平。从图中我们可以看到，Node_A 和Node_B的ID10 、ID9、ID8电位相同，因此这两个节点从总线上读到的电位与他们自己发出的电位也相同，这个时候还没有分出胜负。当Node_B回读总线上的ID7 这一位时，发现总线上的电平跟它自己发送到总线上的不一样，此时，Node_B知道自己

在争夺总线的仲裁中失败了，那么它主动地转换为接收状态，不再发出信息。于是在此之后，总线上的电平和Node_A发出的电平一致，也就是说，Node_A占据了总线的发送权。通过上面的分析我们可以看到，在整个仲裁过程中：

- 在Node_A获取总线的发送权之后，Node_A接着发送自己的Msg_A，因此在竞争总线的过程中不会对Msg_A的传输造成延时；
- 在两个节点竞争总线的过程中，不会破坏Msg_A;

正是由于上面的两点，才称之为非破坏性仲裁机制。*Tips:* 通过上面仲裁过程的分析，我们可以解释*CAN*总线通信的三个特点：*1*）多主控制方式：只要总线空闲，总线上的任意节点都可以向总线上发送数据，直到节点在仲裁中一个个失败，最后只留下一个节点获得总线的发送权。*2*）非破坏性仲裁机制：仲裁段逐位总裁，依靠回读机制、线与机制得以实现。*3*）半双工通信:所谓半双工通信，指的是节点不能在自己发送报文的时候，同时接收其他节点发送来的报文。这是显然的，一个节点正在发送报文时，已经占据了总线的发送权，其他节点肯定不能向总线上发送报文。看一个CAN报文发送的实例，CAN总线上有四个节点：Node_A、Node_B、Node_C、Node_D。发送的报文的ID号分别为5、7、3、6.



2.2.2 仲裁段中的RTR，SRR和IDE位

通过上面标准数据帧的仲裁过程分析，我们已经理解了CAN总线的仲裁机制。但同时也注意到仲裁段除了ID号之外，还有其他的位。

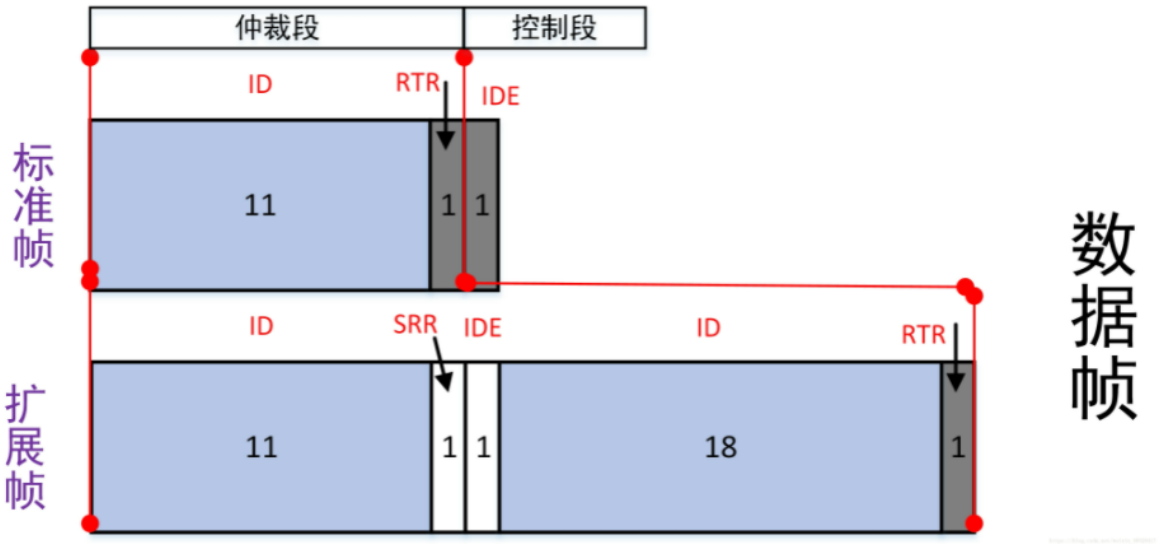
1）RTR位：Tranmission Request Bit（远程发送请求位）。在数据帧中，RTR位恒为显性位0，在遥控帧中，恒为隐性1。

Tips: 这么做的原因是保证数据帧优先级高于遥控帧。比如：在某一时刻*t*，节点*Node_A*发出了*ID*号为*ID_2*遥控帧报文来请求总线上的其它节点发出*ID*号为*ID_2*的数据帧报文。但是就

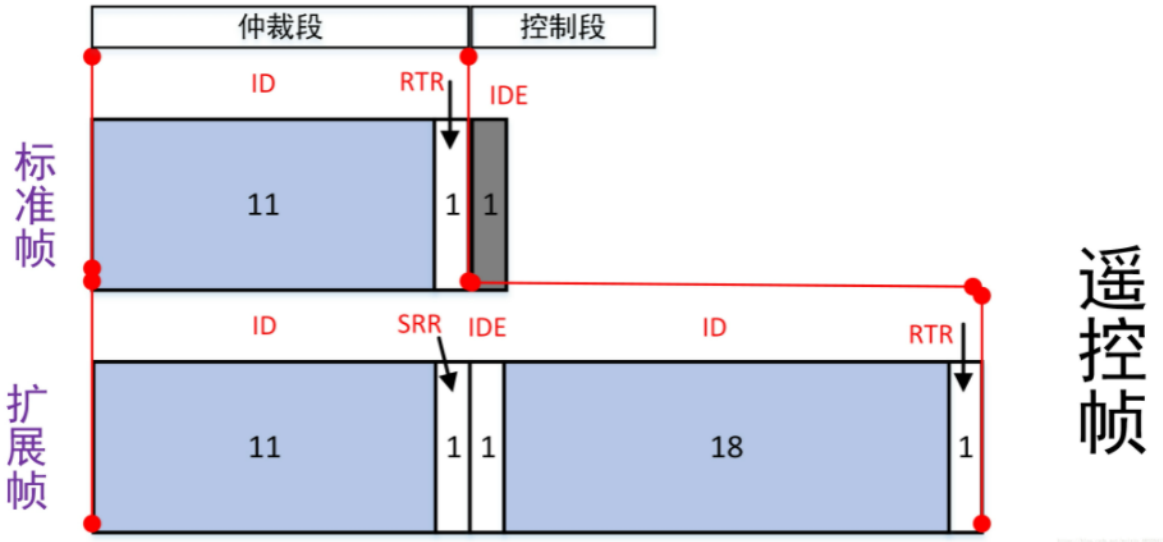
在同一时刻*t*，节点*Node_B*发出了*ID*号为*ID_2*的数据帧报文。这个时候怎么办呢，显然依靠*ID*号不能仲裁出这两帧报文（一个遥控帧，一个数据帧，*ID*号相同）谁能占据总线的发送权，这种情况下，*RTR*位就起作用了，由于*RTR*在数据帧中恒为显性*0*，在遥控帧中恒为隐性*1*，所以在*ID*号相同的情况下，一定是数据帧仲裁获胜。这就解释了 *RTR*位的作用：在*ID*号相同的情况下，保证数据帧的优先级高于遥控帧。

2）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位正好对应标准帧的显性RTR位，这就解释了 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；



3）IDE位全称：Identifier Extension Bit（标识符扩展位）。在扩展帧中恒为隐1，在标准帧中，IDE位于控制段，且恒为显性0。且扩展帧IDE位和标准帧IDE位位置对应，这就保证了：在前11位ID号相同的情况下，标准遥控帧的优先级一定高于扩展遥控帧。



总结 :在ID号前11位相同的情况下:

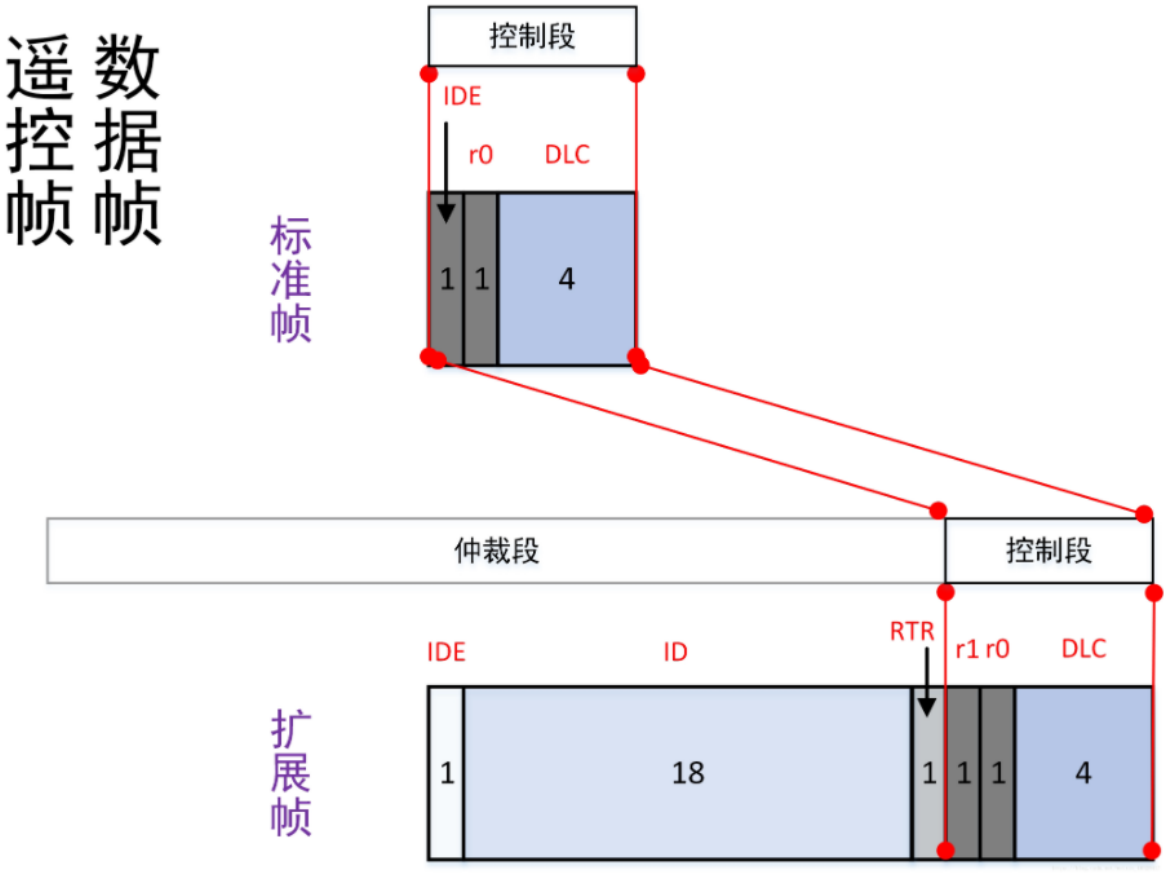
- RTR:保证数据帧优先级高于遥控帧；
- SRR :保证标准数据帧的优先级高于扩展数据帧。
- IDE :保证标准遥控帧的优先级高于扩展遥控帧。

2.2.3 报文过滤

在CAN总线中没有地址的概念，CAN总线是通过报文ID来实现收发数据的。CAN节点上都会有一个验收滤波ID表，其位于CAN节点的验收滤波器中，如果总线上的报文的ID号在某个节点的验收滤波ID表中，那么这一帧报文就能通过该节点验收滤波器的验收，该节点就会接收这一帧报文。比如：Node_A发送了一帧ID号为ID_1的报文Msg_1，Node_B的验收滤波表ID中恰好有ID_1，于是乎Msg_1就会被Node_B接收。*Tips:* 报文过滤机制体现了*CAN*通信的两条特点:*1*)一对一、组播和广播*2*）系统的柔性：正是因为*CAN*总线上收发报文是基于报文*ID*实现的，所以总线上添加节点时不会对总线上已有的节点造成影响。

2.3 控制段

数据帧和遥控帧的控制段结构相同：



- 标准帧中IDE位对应扩展帧中的IDE位，保证在前11位ID号相同的情况下，标准帧的优先级一定高于扩展帧；

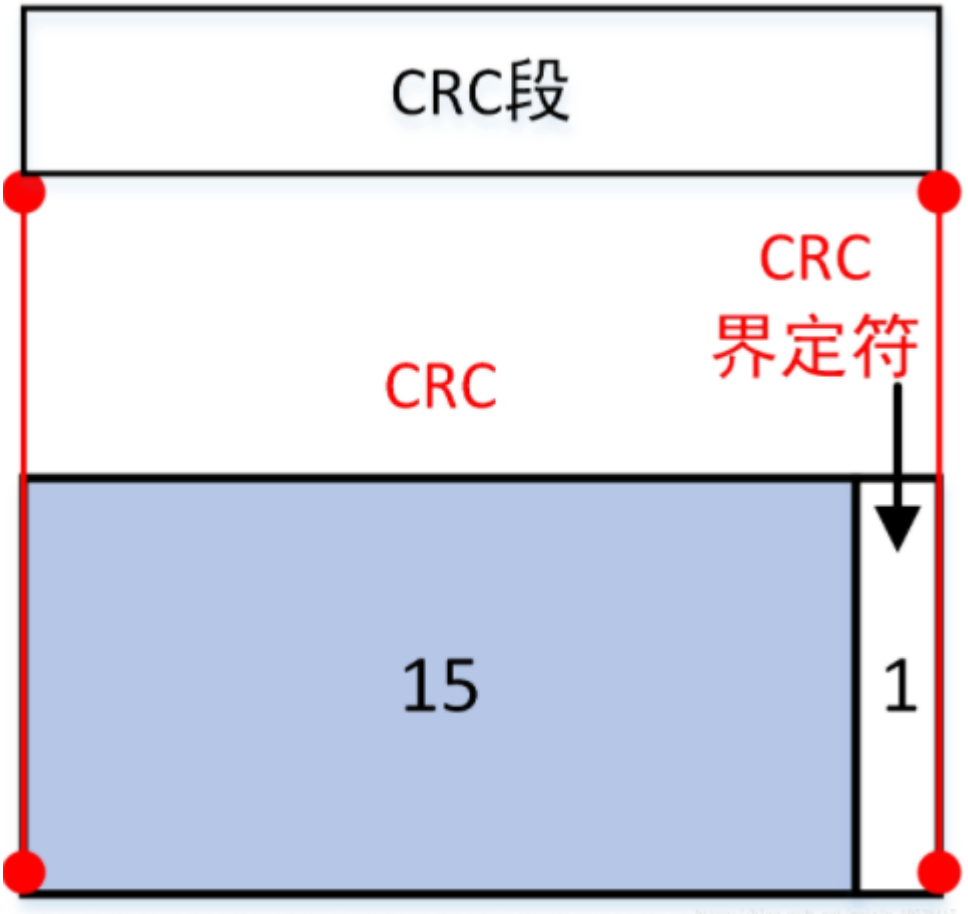
- 然后是保留位r0和r1（扩展帧），保留位r0和r1必须以显性电平发送，但是接受方可以接受显性、隐性及其任意组合的电平；
- 最后是4个字节的DLC（DLC3、DLC2、DLC1、DLC0）代表数据长度，指示了数据段中的字节数。对于没有数据段的遥控帧，DLC表示该遥控帧对应的数据帧的数据段的字节数。

2.4 数据段

数据段可以包含0~8个字节的数据，从MSB（最高位）开始输出。

2.5 CRC段

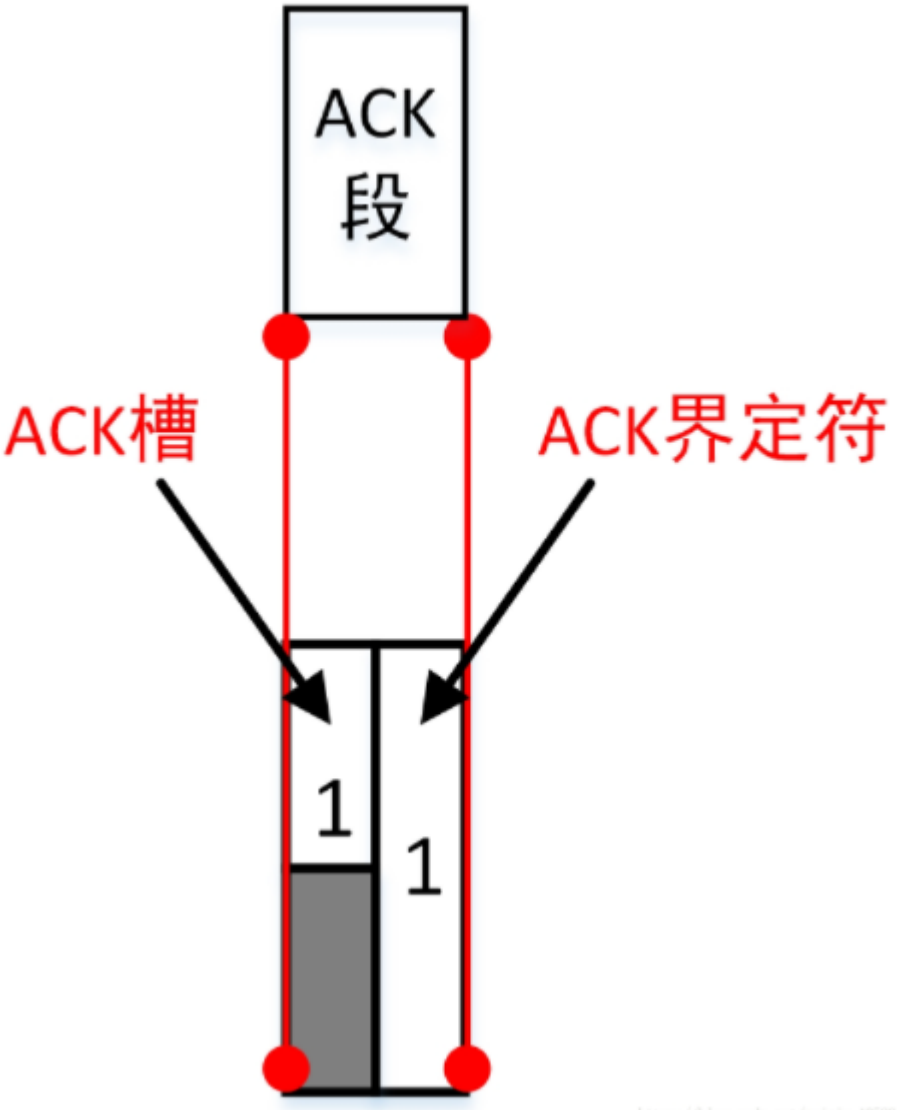
CRC段包含CRC校验序列和CRC界定符。



CRC校验序列是根据多项式生成的CRC值，其计算范围包括：帧起始、仲裁段、控制段和数据段。CRC界定符恒为隐性1。

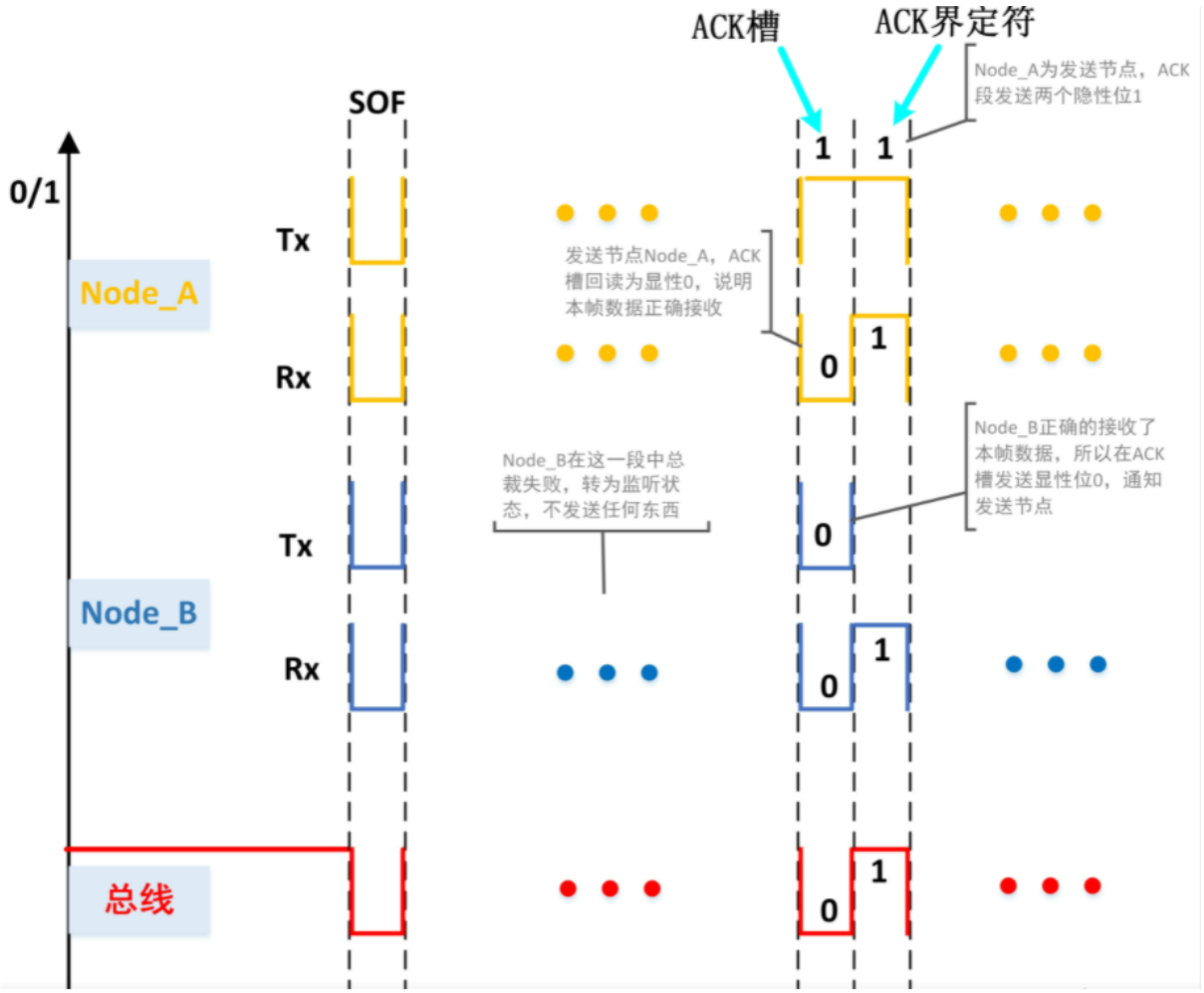
2.6 ACK段

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。

Tips: 我们以标准数据帧为例来分析*ACK*段的工作方式：如图所示，*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。



2.7 帧结束

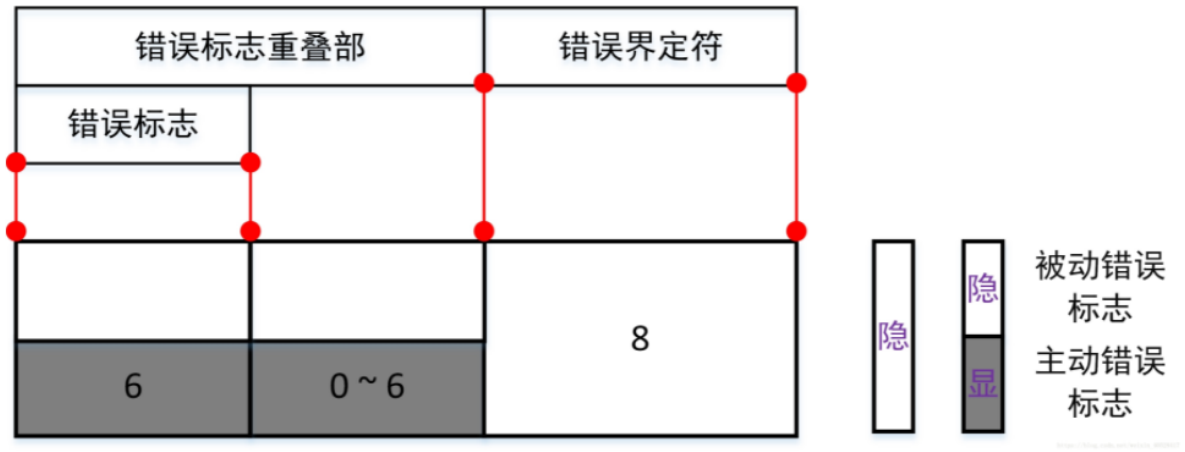
帧结束段表示该帧报文的结束，由7个隐性位构成。

03 CAN协议错误帧

3.1 错误帧的帧结构

在发送和接收报文时，总线上的节点如果检测出了错误，那么该节点就会发送错误帧，通知总线上的节点，自己出错了。

错误帧由错误标志和错误界定符两个部分组成。



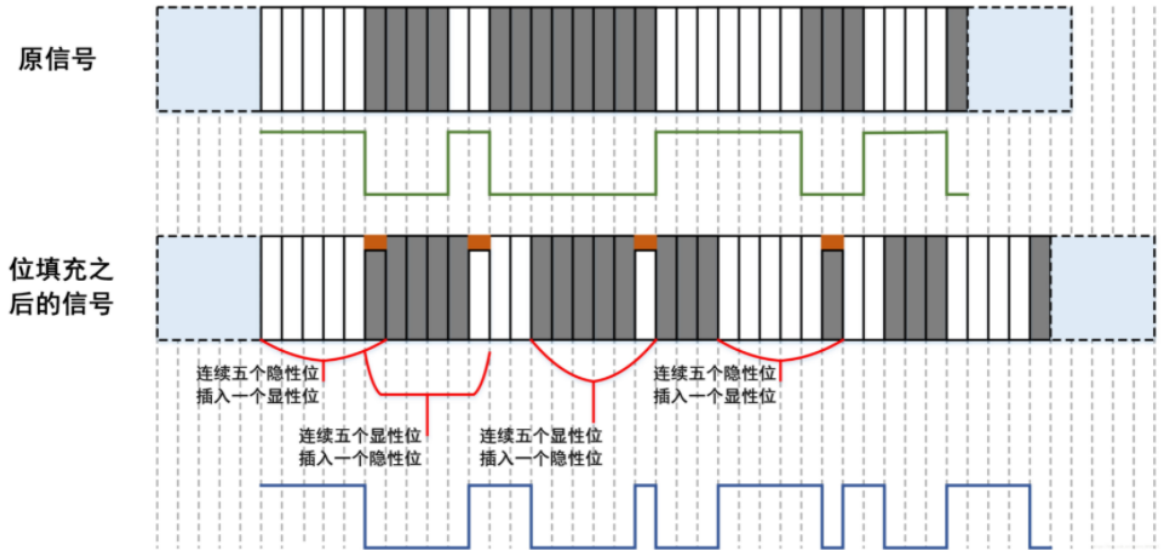
- 主动错误标志：6个连续的显性位；
- 被动错误标志：6个连续的隐性位；
- 错误界定符：8个连续的隐性位。

可以看到在错误标志之后还有0~6位的错误标志重叠，这一段最低有0个位，最多有6个位，关于这一段是怎么形成的，将在下文中解释。

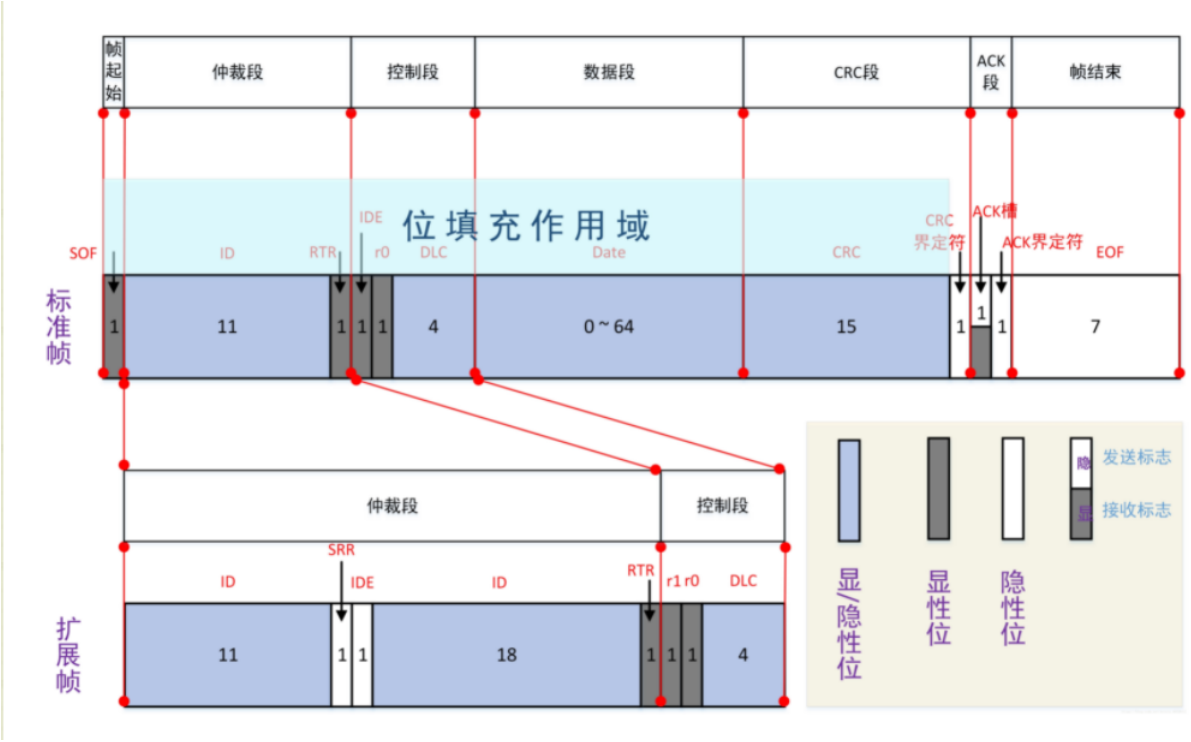
3.2 错误检测

3.21 位填充原则

在了解CAN总线中的错误检测之前，首先需要了解什么是位填充。CAN协议中规定，当相同极性的电平持续五位时，则添加一个极性相反的位。



对于发送节点而言：在发送数据帧和遥控帧时，对于SOF~CRC(除去CRC界定符) 之间的位流，相同极性的电平如果持续5位，那么在下一个位插入一个与之前5位反型的电平；对于接收节点而言：在接收数据帧和遥控帧时，对于**SOF~CRC(除去CRC界定符)**之间的位流，相同极性的电平如果持续5位，那么需要删除下一位再接收。



Tips: 注意:填充位的添加和删除是由发送节点和接收节点完成的，*CAN-BUS*只负责传输，不会操纵信号。

3.22 错误的种类

在CAN总线通信中，一共有五种错误，分别是：位错误、ACK错误、填充错误、CRC错误、格式错误。

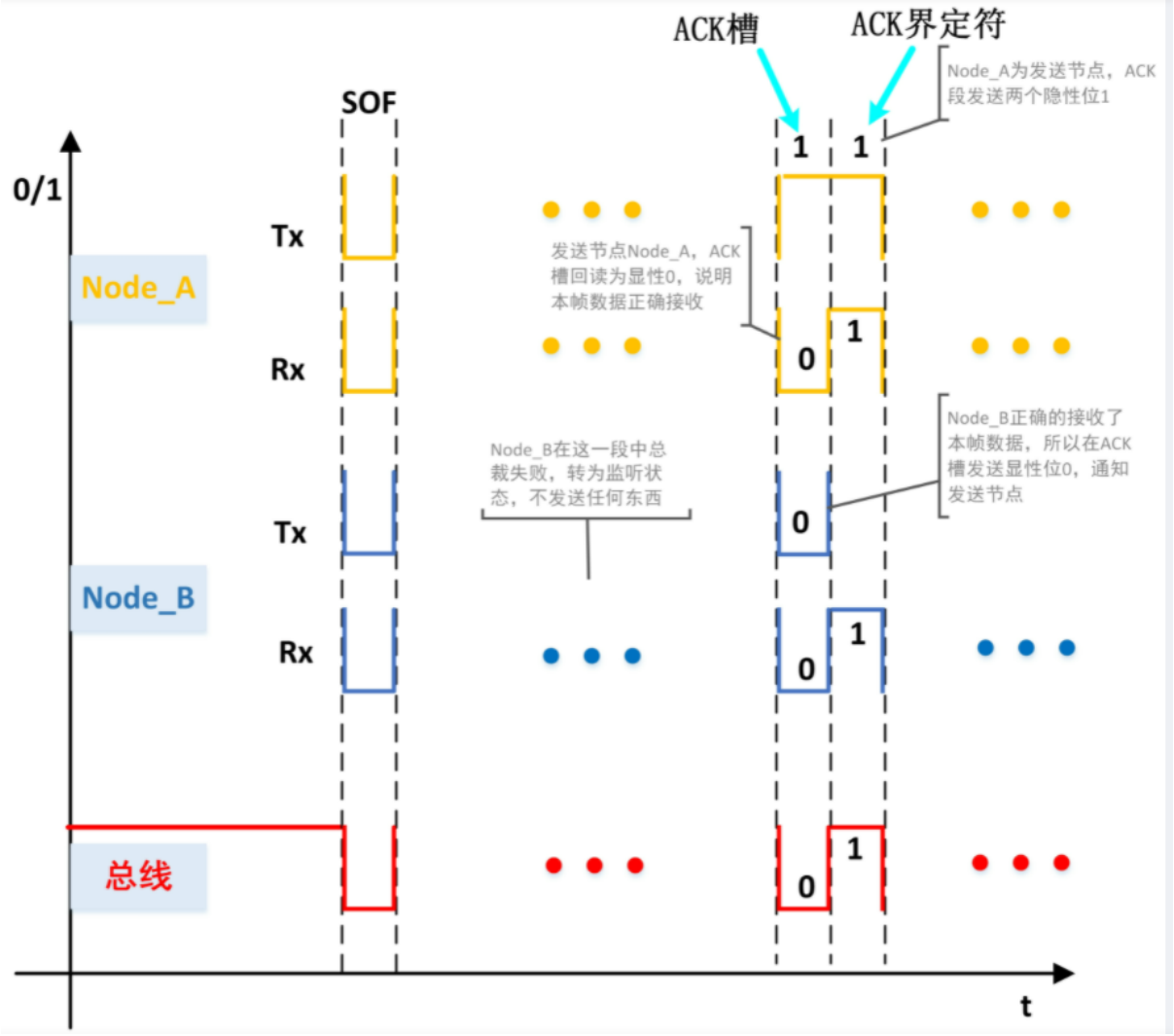
位错误（Bit Check Error）节点将自己发送到总线上的电平与同时从总线上回读到的电平进行比较，如果发现二者不一致，那么这个节点就会检测出一个位错误。实际上所谓“发出的电平与从总线上回读的电平不一致”，指的就是节点向总线发出隐性位，却从总线上回读到显性位或者节点向总线发出显性位，却从总线上回读到隐性位这两种情况。*Tips:* 有三种例外情况不属于位错误：

- 在仲裁区，节点向总线发送隐性位却回读到显性位，不认为是位错误，这种情况表示该节点仲裁失败；
- 在*ACK*槽，节点向总线发送隐性位却回读到显性位，不认为是位错误，这种情况表示，该节点当前发送的这一帧报文至少被一个其它节点正确接收；
- 该节点发送被动错误标志，节点*Node_A*向总线发送连续六个隐性位（被动错误标志）却回读到显性位，不认为是位错误。因为被动错误标志是六个连续的隐性位，所以在总线上按照线与机制，有可能这六个连续隐性位被其它节点发送的显性电平“吃掉”；

ACK错误（Acknowledgment Error）按照CAN协议的规定，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。



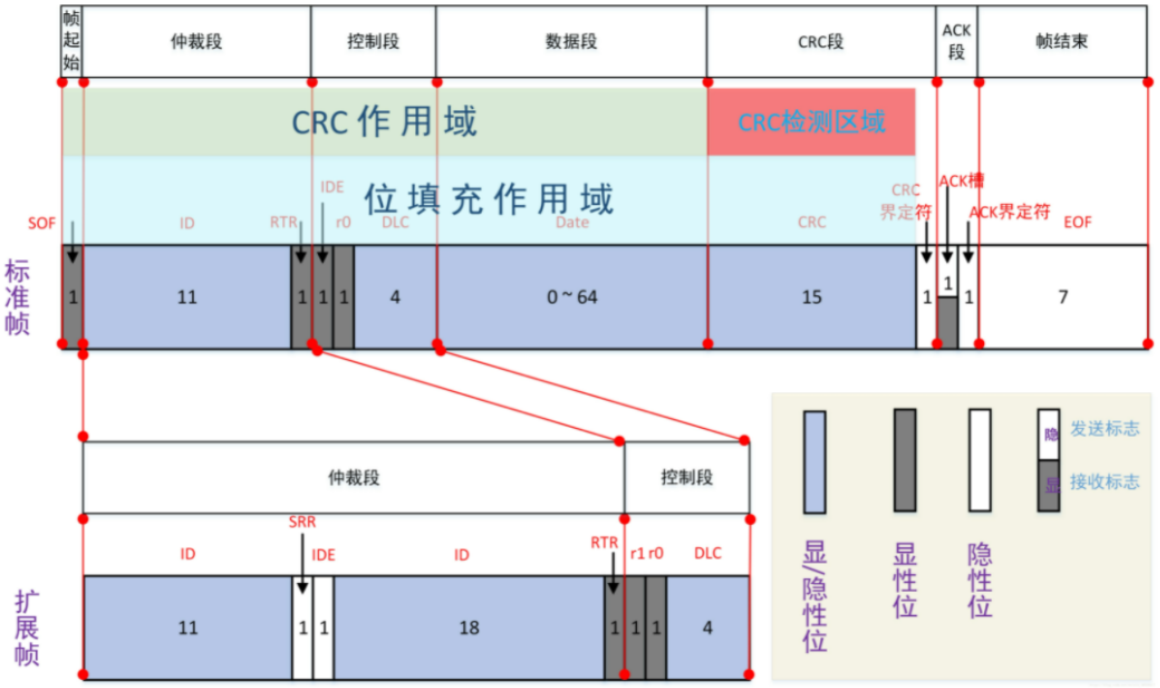
填充错误（Fill Error）

在需要执行位填充原则的帧段（数据帧遥控帧的SOF~CRC序列），检测到连续六个同性位，则检测到一个填充错误。

CRC错误

发送节点Node_A在发送数据帧或者遥控帧时，会计算出该帧报文的CRC序列。接收节点Node_B在接收报文时也会执行相同的CRC算法，如果接收节点Node_B计算出的CRC序列

值与发送节点Node_A发来的CRC序列值不一致，那么接收节点就检测到一个CRC错误。



格式错误

在一帧报文发送时，如果在必须发送预定值的区域内检测到了非法值，那么就检测到一个格式错误。CAN报文中，有预定值的区域包括：

- 数据帧和遥控帧的CRC界定符、ACK界定符、EOF；
- 错误帧界定符
- 过载帧界定符

3.3 错误通知

上一节中，讲到CAN通信中有五种错误，并且介绍了在什么情况下能够检测到这几种错误，在检测到错误之后，检测到错误的节点就要发送错误帧到总线上来通知总线上的其他节点。错误帧有的带有主动错误标志，有的带有被动错误标志，而且错误标志重叠部分的字节数也不一样，那么问题就来了：

- 什么情况下发送带有主动错误标志的错误帧；
- 什么情况下发送带有被动错误标志的错误帧；
- 在哪个时间点发送错误帧；
- 错误标志重叠部分是怎样形成的；

3.31 节点错误状态

按照CAN协议的规定，CAN总线上的节点始终处于以下三种状态之一。

- 主动错误状态

- 被动错误状态
- 总关闭状态

当满足一定的条件时，节点可以从一种状态转换为另外一种状态。*Tips:* 需要注意的是:

- 处于主动错误状态，表示该节点具备发出主动错误标志的能力;
- 处于被动错误状态，表示节点具备发出被动错误标志的能力。

1)主动错误状态

- 节点处于主动错误状态可以正常通信；
- 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。

2)被动错误状态

- 节点处于被动错误状态可以正常通信；
- 处于被动错误状态的节点（可能是接收节点也可能是发送节点）在检测出错误时，发出被动错误标志。

Tips: 注意:这里说处于主动错误状态或被动错误状态的节点仍然可以正常通信，这里的正常通信指的是:节点仍然能够从总线上接收报文，也能够竞争总线获胜后向总线上发送报文。但是不代表接收的报文一定正确也不代表一定能正确的发送报文。

3)总线关闭状态

- 节点处于总线关闭状态，那么该节点不能收发报文；
- 处于总线关闭状态的节点，只能一直等待，在满足一定条件的时候，再次进入到主动错误状态。

3.32 错误状态的转换

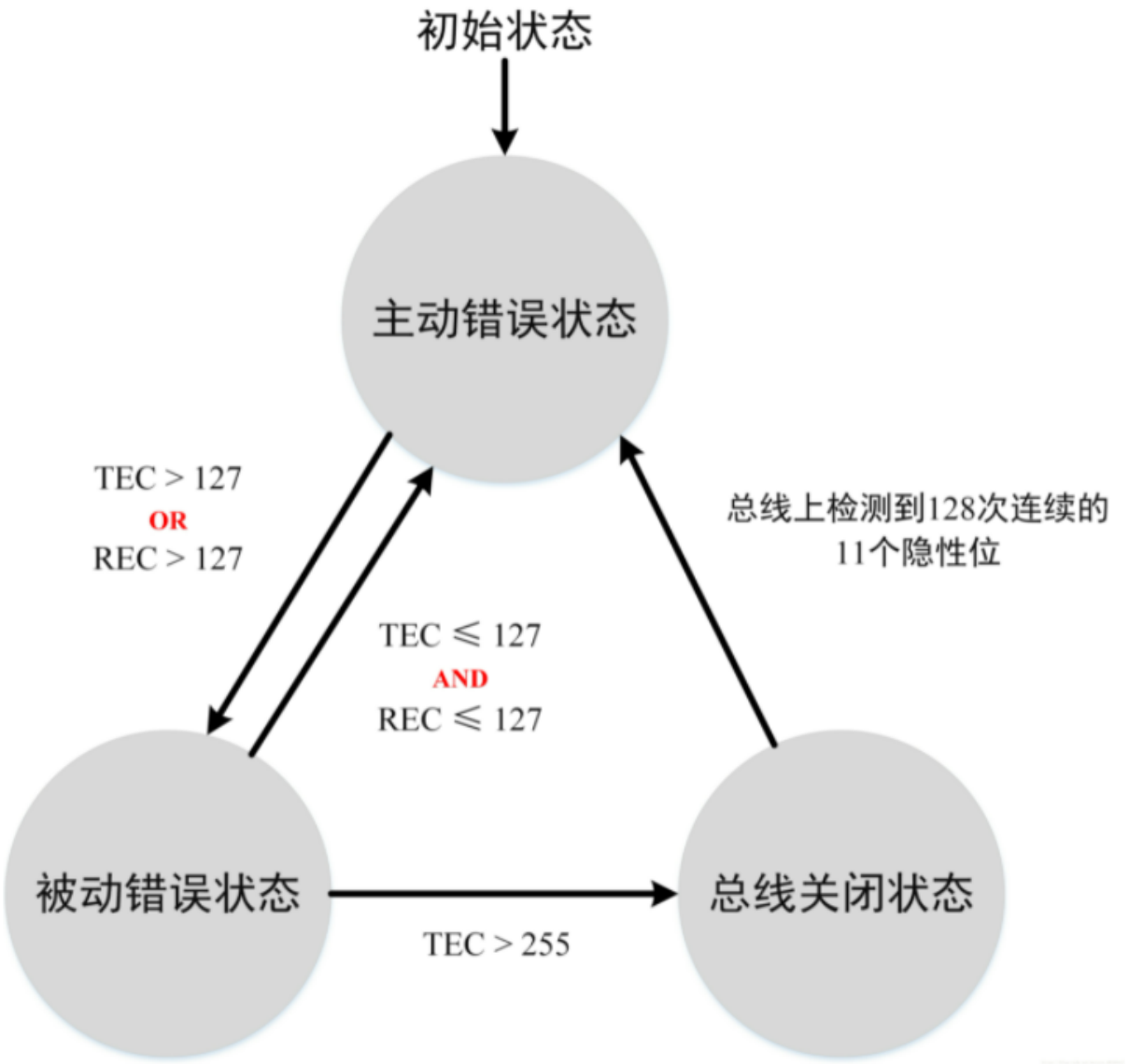
现在我们知道：

- 处于主动错误状态的节点在检测到错误时会发送带有主动错误标志的错误帧；
- 处于被动错误状态的节点在检测到错误时会发送带有被动错误标志的错误帧。

那么一个CAN节点在什么情况下处于主动错误状态，什么情况下处于被动错误状态呢？根据CAN协议的规定，在CAN节点内，有两个计数器：发送错误计数器（TEC）和接收错误计数器（REC）。*Tips:* 需要注意的是:这两个计数器计得不是收发报文的数量，也不是收发错误帧的数量。*TEC*和*RCE*计数值的变化，是根据下表的规定来进行的

	接受和发送错误计数值的变动条件	发送错误计数值 (TEC)	接收错误计数值 (REC)
1	接收单元检测出错误时。 例外：接收单元在发送错误标志或过载标志中检测出“位错误”时，接收错误计数值不增加。	—	+1
2	接收单元在发送完错误标志后检测到的第一个位为显性电平时。	—	+8
3	发送单元在输出错误标志时。	+8	—
4	发送单元在发送主动错误标志或过载标志时，检测出位错误。	+8	—
5	接收单元在发送主动错误标志或过载标志时，检测出位错误。	—	+8
6	各单元从主动错误标志、过载标志的最开始检测出连续 14 个位的显性位时。 之后，每检测出连续的 8 个位的显性位时。	发送时 +8	接收时 +8
7	检测出在被动错误标志后追加的连续 8 个位的显性位时。	发送时 +8	接收时 +8
8	发送单元正常发送数据结束时（返回 ACK 且到帧结束也未检测出错误时）。	–1 TEC=0 时±0	—
9	接收单元正常接收数据结束时（到 CRC 未检测出错误且正常返回 ACK 时）。	—	1≤REC≤127 时-1 REC=0 时±0 REC>127 时 设 REC=127
10	处于总线关闭态的单元，检测到 128 次连续 11 个位的隐性位。	TEC=0	REC=0

CAN节点错误状态的转换，就是基于这两个计数器来进行的。



可以看出，节点错误状态的转换就是一个**“量变”到“质变”的过程：

1) 主动错误状态

最开始TCE和REC都小于127时**，就处于主动错误状态。在这一状态下，节点检测到一个错误就会发送带有主动错误标志的错误帧，因为主动错误标志是连续六个显性位，所以这个时候主动错误标志将会“覆盖”掉总线上其它节点的发送，而之前在CAN总线上传输的报文就被这“六个连续显性位”破坏掉了。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我发错了，现在我破坏掉它（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：刚刚我收报文的时候发现了错误，不管你们有没有发现这个错误，我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。

Tips: 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。

2）被动错误状态

如果某个节点发送错误帧的次数较多，必将使得TCE > 127 或者 REC > 127，那么该节点就处于被动错误状态。在这一状态下，节点Node_A检测到一个错误就会发送带有被动错误标志的错误帧，因为被动错误标志是连续六个隐性位，所以这个时候总线上正在传输的报文位流不会受到该被动错误帧的影响，其它的节点该发送的发送，该接收的接收，没人搭理这个发送被动错误帧的节点Node_A。如果发出被动错误帧的节点Node_A为报文的发送节点，那么在发送被动错误帧之后，刚刚正在发送的报文被破坏，并且Node_A不能在错误帧之后随着连续发送刚刚发送失败的那个报文。随之而来的是帧间隔，并且连带着8位隐性位的 “延迟传送” 段；这样总线电平就呈现出连续11位隐性位，总线上的其它节点就能判定总线处于空闲状态，就能参与总线竞争。此时如果Node_A能够竞争成功，那么它就能接着发送，如果竞争不能成功，那么就接着等待下一次竞争。这种机制的目的正是为了让其它正常节点（处于主动错误）优先使用总线。

Tips: 处于被动错误状态，说明这个节点目前是不太可靠的，出现错误的原因可能是它本身的问题，即刚刚检测到的错误可能仅仅只有它自己遇到，正是因为这一点，整个总线才不信任它报告的错误，从而只允许它发送六个连续的隐性位，这样它才不会拖累别人。

3）总线关闭状态

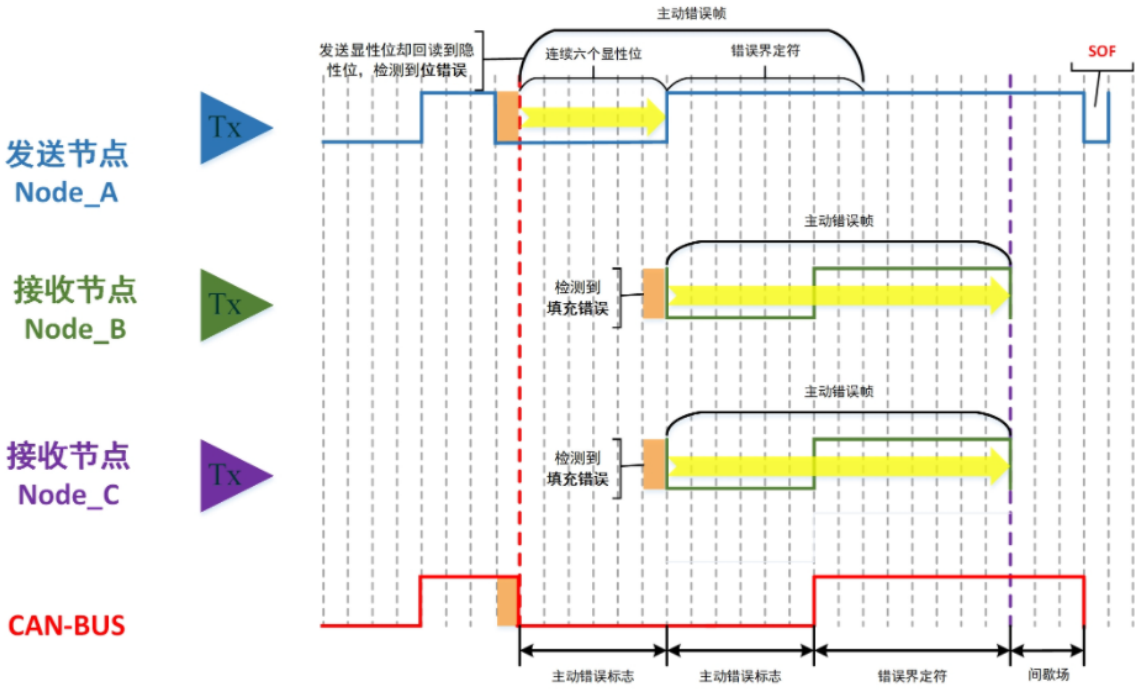
如果一个处于被动错误状态的节点，仍然多次发送被动错误帧，那么势必导致TEC ＞ 255，这样就处于总线关闭状态。在总线关闭状态下的节点Node_A不能向总线上发送报文，也不能从总线上接收报文，整个节点脱离总线。等到检测到128次11个连续的隐性位时，TEC和REC置0，重新回到主动错误状态。按照我的理解这个所谓“检测到128次11个连续隐性位”其实就是让这个节点隔离一段时间冷静下，因为它一旦处于总线关闭状态，就不会和总线有任何的联系，这个时候只要它计算时间等于达到传送128次11个连续隐性位所用的时间，就可以重新连到总线上。

Tips: 处于总线关闭状态说明，这个节点目前挂掉了，总线先把它踢开，这样它才不会拖累别人，等到它冷静一段时间之后再回到总线上。

3.33 错误帧的发送

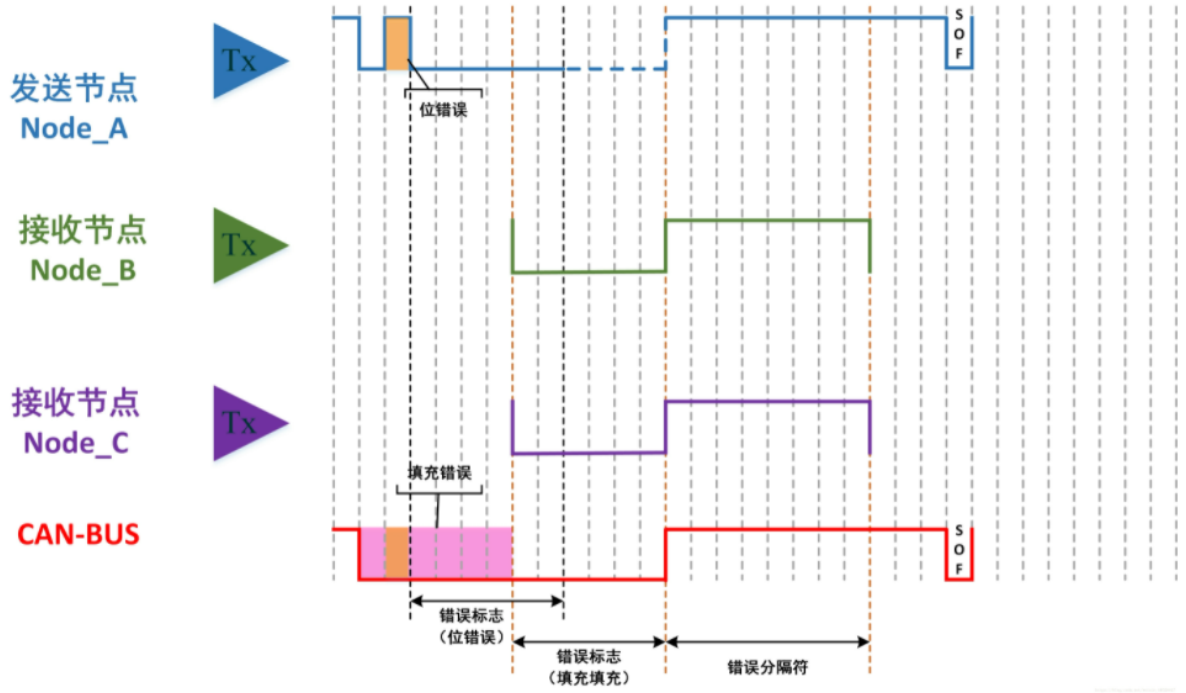
在检测到错误之后，什么时候发送错误帧呢？按照CAN协议的规定：1、位错误、填充错误、格式错误、ACK错误。在错误产生的那一位的下一位开始发送错误帧。2、CRC错误紧随ACK界定符后的位发送错误帧。

例子1：



(1)发送节点Node_A发送一个显性位，但是却从总线上听到一个隐形位，于是Node_A节点就会检测到一个位错误；
(2)Node_A检测到位错误之后，立即在下一位开始发送主动错误帧：6个连续显性位的主动错误标志+8个连续隐性位的错误界定符；
(3)对应Node_A发出的主动错误标志，总线上电平为6个连续显性位；
(4)接收节点Node_B和Node_C从总线上听到连续6个显性位，那么就会检测到一个填充错误，于是这两个节点都会发送主动错误帧；
(5)对应Node_B和Node_C发出的主动错误标志，总线电平又有6个连续显性电平，对应Node_B和Node_C发出的错误界定符，总线电平有8个连续的隐性电平。
(6)在间歇场之后，Node_A节点重新发送刚刚出错的报文。

例子2:



从上图中可以看出错误帧之中，错误标志重叠部分是怎样形成的，这个例子中，位错误的错误标志与填充错误的错误标志重叠两位，剩下的部分还有四位：