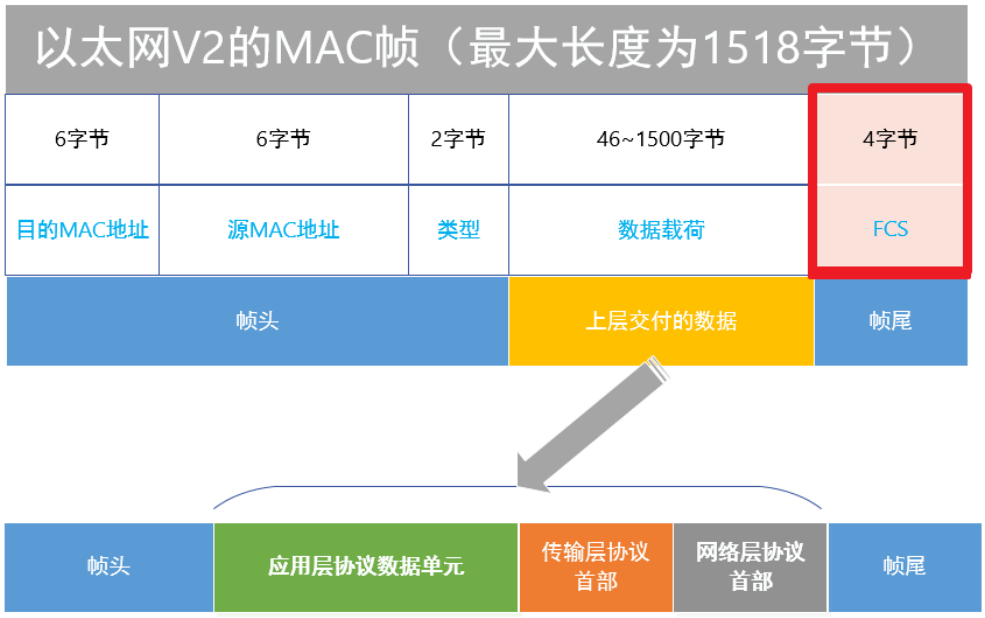
我们知道，实际通信链路都不是理想的，比特在传输过程中可能会产生差错，1可能会变成0，0可能会变成1，这称为**比特差错**。

那么接收方如何判断数据在传输过程中是否出现了差错呢？**我们可以在发送数据后面增加差错检测码来检测**。本篇文章来看看数据链路层中差错检测问题。

我们以以太网MAC帧格式为例说明：



我们可以看到帧尾是一个4字节的帧检验序列FCS字段，其作用是让接收方数据链路层检查帧在传输过程中是否出现了误码。

其基本思想为：为了保证数据在传输过程中的完整性，采用一种指定的算法对原始数据进行计算，得出的一个校验值。接收方接收到数据时，采用同样的校验算法对原始数据进行计算，如果计算结果和接收到的校验值一致，说明数据校验正确，这一帧数据可以使用，如果不一致，说明传输过程中出现了差错，这一帧数据丢弃，请求重发。

常用的差错检测方法有奇偶校验、CRC循环冗余算法等方法。

PS：FCS是指帧检验序列，就是添加在数据后面的冗余码；CRC或奇偶校验是一种检测算法。注意区分FCS和CRC概念上的区别。

一、奇偶校验

用奇偶校验法算出来的冗余码叫做奇偶校验码，可以用来检测数据传输过程中是否发生错误，是众多校验码中最为简单的一种。

顾名思义，它有两种校验方法：奇校验和偶校验。

奇校验：原始码流+校验位 总共有奇数个1

偶校验：原始码流+校验位 总共有偶数个1

这个算法特别简单，比如双方约定以奇校验为校验方式，那么发送方检测比特流中1的个数，如果数据部分1的个数为偶数个，则校验位要放1，这样1的个数才是奇数个；接收方接收后，判断该帧中比特1的数量，如果为奇数个，那么表示没有误码，如果是偶数个，则产生了误码。

比如原始数据为1011000，其中1的个数为3，如果发送方和接收方约定采用奇校验，那么此时发送方数据中1的个数已经为3符合条件，那么奇校验位为0即可，接收方判断1的个数为奇数即可通过校验；反之如果约定采用偶校验，那么奇校验位为1，使得1的个数为4符合偶数发送出去，接收端判断此时1的个数是否为偶数。

原始码	奇校验 (保证有奇数个1)	偶校验 (保证有偶数个1)
1011000	10110000	10110001
1010000	10100001	10100000
0011010	00110100	00110101
0001000	00010000	00010001
0000000	00000001	00000000

可以看到，这个方法十分简单，但是不可靠，为什么这么说呢？假设约定的是奇校验，即发送方构建的一帧的比特流中1的个数为奇数个。但是不幸的是中间还是出现了比特差错，如果正好是错了两个比特呢？很有可能还是奇数个1，那么接收方就会误判为无误码。这种只有一半概率的校验算法着实让人睡不着觉，因此计算机网络的数据链路层不会采用此检测方法。

二、循环冗余校验CRC

循环冗余校验（Cyclic Redundancy Check， CRC）是一种根据网络数据包或计算机文件等数据产生简短固定位数校验码的一种信道编码技术，主要用来检测或校验数据传输或者保存后可能出现的错误。它是利用除法及余数的原理来作错误侦测的。

这是一种具有很强检错能力的检错方法，漏检率极低，校验计算速度快，易于用编码器等硬件电路实现。从检错的正确率与速度、成本等方面，都比奇偶校验等校验方式具有优势。因而，CRC 成为计算机信息通信领域最为普遍的校验方式。其基本思想为：

- 收发双方约定好一个生成多项式G(x)
- 发送方基于待发送的数据和生成多项式计算出差错检测码（冗余码），将其添加到待传输数据的后面一起传输
- 接收方通过生成多项式来计算收到的数据是否产生了误码

生成多项式是最重要的参数，需要双方使用相同的生成多项式，比如生成多项式为：

$$G(x) = x^4 + x^2 + x + 1 = 1 * x^4 + 0 * x^3 + 1 * x^2 + 1 * x + 1 \quad (1)$$

此示例生成多项式对应的二进制比特串就为：10111。算法要求生成多项式必须包含最低次项，即最高位和最低位必须均为1，常用的生成多项式有：

$$CRC - 16 = x^{16} + x^{15} + x^2 + 1 \quad (2)$$

$$CRC - CCITT = x^{16} + x^{12} + x^5 + 1 \quad (3)$$

$$CRC - 32 = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (4$$

发送端和接收端约定好了生成多项式后，这个生成多项式对应的二进制比特串将被作为除数，我们完整介绍下CRC校验步骤：

- 第一步：预先确定一个发送端和接收端都用来作为除数的二进制比特串（生成多项式）；
- 第二步：发送端原始帧后面需要先补0，0的个数就看多项式最高次幂是多少，比如G(x) = x^4 + x^2 + x + 1这个生成多项式，最高次数为4，那么就在数据二进制形式后面补4个0；
- 第三步：把原始帧与上面选定的除数进行二进制除法运算（模2除法），所得到的余数（也是二进制的比特串）就是该帧的CRC校验码，也称之为FCS；
- 第四步：再把这个校验码附加在原数据帧后面，构建一个新帧发送到接收端；
- 第五步：接收端再把这个新帧以“模2除法”方式除以前面选择的除数，如果没有余数，则表明该帧在传输过程中没出错，否则出现了差错。

可以看出来，有两个关键点，第一个是要预先确定一个发送端和接收端都用来作为除数的二进制比特串（生成多项式）；第二个是把原始帧与上面选定的除数进行二进制除法运算，计算出FCS。那么模2除法又是如何计算的呢?

三、模2除法的计算规则

要说模2除法，先说模2加法和模2减法。

模2加法运算为：1+1=0，0+1=1，0+0=0，无进位，也无借位。可以看到加法过程符合异或运算结果。

模2减法运算为：1-1=0，0-1=1，1-0=1，0-0=0，也无进位，无借位。可以看到减法过程也符合异或运算结果。

模2乘法跟普通的二进制乘法一样：0 * 0 = 0，0 * 1 = 0，1 * 0 = 0，1 * 1 = 1。只是在中间结果的计算上采用的是模2加法来计算。

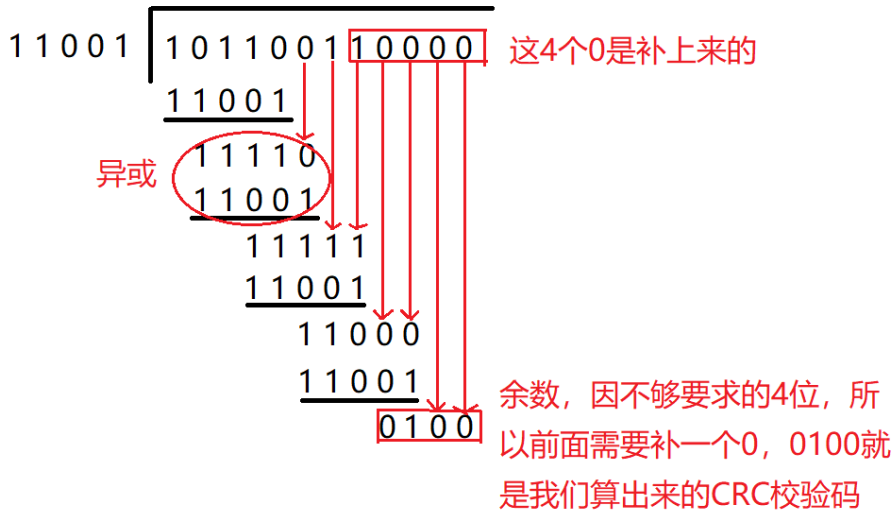
模2除法则采用的是模2减法运算。可以看到，实际上减法就是异或运算！那么下面对于除法的运算就全部转为异或运算。

四、CRC计算示例

我们以一个例子来理解CRC如何进行校验的。

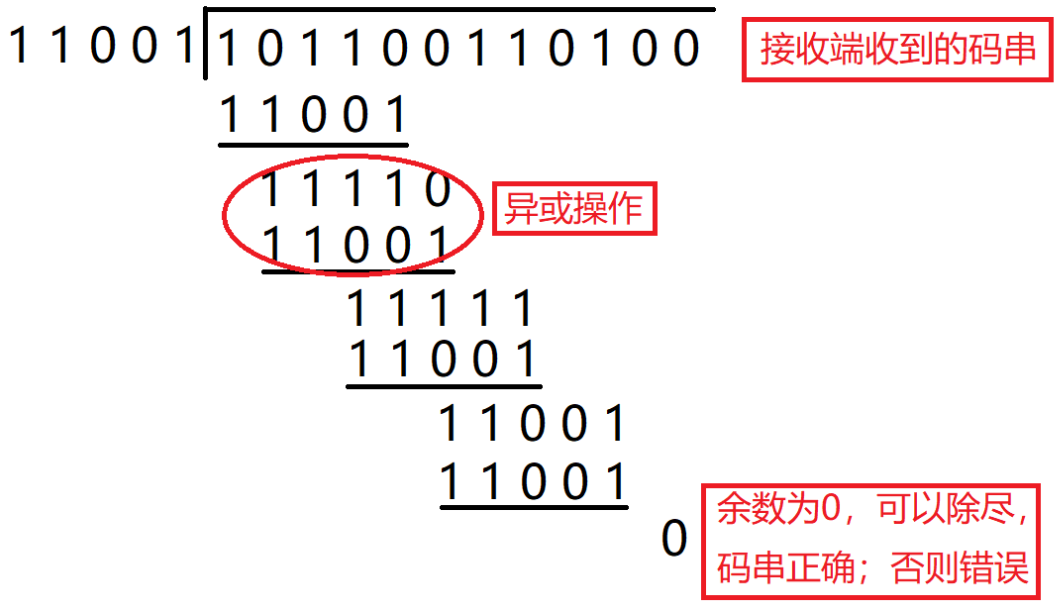
假设原始帧的二进制比特串为10110011，生成多项式为G（X）= X4 + X3 + 1。

- 第一步：生成多项式对应的二进制比特串为11001；
- 第二步：由于生成多项式最高次数为4，因此在原始帧上后面补4个0，原始帧变为：101100110000；
- 第三步：101100110000按照“模2除法”方式除以11001，得到余数0100，这个余数就是CRC校验码，具体计算方式如下图；



最后余数位数小于除数位数计算停止，余数的位数一定要是比除数位数只能少一位，哪怕前面位是0，甚至是全为0（附带好整除时）也都不能省略。

- 把这个校验码附加在原数据帧后面，构建一个新帧：101100110100发送到接收端；
- 最后，接收方根据生成多项式和数据串去验证是否能除尽，能则说明数据串正确，接收端的处理如下图：



好了，关于CRC校验算法就介绍完毕了，我们只需要了解其大概思想即可。

检错码只能检测出帧在传输过程中出现了差错，但是不能定位错误，因此无法纠正错误。

要想纠正传输中的差错，可以使用冗余信息更多的纠错码进行前向纠错，但纠错码的开销比较大，计算机网络中较少使用。

循环冗余校验CRC有很好的检错能力（漏检率非常低），虽然计算比较复杂，但非常易于用硬件实现，因此被广泛应用于数据链路层。

在计算机网络中通常采用我们后续将要讨论的检错重传方式来纠正传输中的差错，或者仅仅是丢弃检测到差错的帧，这取决于数据链路层向其上层提供的是可靠传输服务（TCP）还是不可靠传输服务（UDP）。