星火spark

CRC校验算法

CRC(Cyclic Redundancy Check)循环冗余校验是常用的数据校验方法，讲CRC算法的文章很多，之所以还要写这篇，是想换一个方法介绍CRC算法，希望能让大家更容易理解CRC算法。

先说说什么是数据校验。数据在传输过程（比如通过网线在两台计算机间传文件）中，由于传输信道的原因，可能会有误码现象（比如说发送数字5但接收方收到的却是6），如何发现误码呢？方法是发送额外的数据让接收方校验是否正确，这就是数据校验。最容易想到的校验方法是和校验，就是将传送的数据(按字节方式)加起来计算出数据的总和，并将总和传给接收方，接收方收到数据后也计算总和，并与收到的总和比较看是否相同。如果传输中出现误码，那么总和一般不会相同，从而知道有误码产生，可以让发送方再发送一遍数据。

CRC校验也是添加额外数据做为校验码，这就是CRC校验码，那么CRC校验码是如何得到的呢？

非常简单，CRC校验码就是将数据除以某个固定的数（比如ANSI-CRC16中，这个数是0x18005），所得到的余数就是CRC校验码。

那这里就有一个问题，我们传送的是一串字节数据，而不是一个数据，怎么将一串数字变成一个数据呢？这也很简单，比如说2个字节B1，B2,那么对应的数就是(B1<<8)+B2；如果是3个字节B1，B2，B3，那么对应的数就是((B1<<16)+(B2<<8)+B3),比如数字是0x01,0x02,0x03，那么对应的数字就是0x10203；依次类推。如果字节数很多，那么对应的数就非常非常大，不过幸好CRC只需要得到余数，而不需要得到商。

从上面介绍的原理我们可以大致知道CRC校验的准确率，在CRC8中出现了误码但没发现的概率是1/256，CRC16的概率是1/65536，而CRC32的概率则是1/2^32，那已经是非常小了，所以一般在数据不多的情况下用CRC16校验就可以了，而在整个文件的校验中一般用CRC32校验。

这里还有个问题，如果被除数比除数小，那么余数就是被除数本身，比如说只要传一个字节，那么它的CRC就是它自己，为避免这种情况，在做除法之前先将它移位，使它大于除数，那么移多少位呢？这就与所选的固定除数有关了，左移位数比除数的位数少1，下面是常用标准中的除数：

CRC8：多项式是X8+X5+X4+1，对应的数字是0x131，左移8位

CRC12：多项式是X12+X11+X3+X2+1，对应的数字是0x180D，左移12位

CCITT CRC16：多项式是X16+X12+X5+1，对应的数字是0x11021，左移16位

ANSI CRC16：多项式是X16+X15+X2+1，对应的数字是0x18005，左移16位

CRC32：多项式是X32+X26+X23+X22+X16+X12+X11+X10+X8+X7+X5+X4+X2+X1+1，对应数字是0x104C11DB7，左移32

因此，在得到字节串对应的数字后，再将数字左移M位（比如ANSI-CRC16是左移16位），就得到了被除数。

昵称：星火spark
园龄：6年4个月
粉丝：8
关注：0
+加关注

< 2019年2月 >

日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	1	2
3	4	5	6	7	8	9

搜索

找找看

谷歌搜索

常用链接

我的随笔

好了，现在被除数和除数都有了，那么就要开始做除法求CRC校验码了。CRC除法的计算过程与我们笔算除法类似，首先是被除数与除数高位对齐后，被除数减去除数，得到了差，除数再与差的最高位对齐，进行减法，然后再对齐再减，直到差比除数小,这个差就是余数。不过和普通减法有差别的是，CRC的加(减)法是不进(借)位的，比如10减01，它的结果是11，而不是借位减法得到的01，因此，实际上CRC的加法和减法所得的结果是一样的，比如10加01的结果是11，10减01的结果也是11，这其实就是异或操作。虽然说了这么多也不一定能说清楚，我们还是看一段CRC除法求余程序吧：

```
unsigned short crc16_div()
{
	unsigned long data = 0x880000;
	unsigned long ccitt16 = 0x11021;


	unsigned long cmp_value = 0x10000;

	int i;
	ccitt16 <<= 7;
	cmp_value <<= 7;

	while (cmp_value >= 0x10000)
	{
		if ((data & cmp_value) != 0)
		{
			data ^= ccitt16;
		}
		else
		{

		}


		ccitt16 >>= 1;
		cmp_value >>= 1;
	}


	return (data & 0xFFFF);
}
```

好了，现在我们已经会计算0x88的CRC校验码了，它只是对0x880000做除法运算求余数而已，不过这只是求单字节的CRC校验码，那如果有十多个字节怎么办？我们的计算机也存不下那么大的数呀，看来我们还要对程序进行些改进，使它能对大数求除法了。

```
unsigned short crc16_div_2()
{


	unsigned short data = 0x88;


	unsigned short ccitt16 = 0x1021;
```

我的评论
我的参与
最新评论
我的标签

随笔分类

ant
bootstrap
dwz
eos
ext
GIS专业相关(1)
greenetplum
java(10)
javascript(6)
jetty
jquery
luence
maven(1)
mdrill
mysql(3)
oracle(15)
tomcat(1)
weblogic(2)
常见问题(18)
大数据
感悟(1)
工作流
后台(34)
前端(2)
数据库(12)
知识库(5)

随笔档案

2018年1月 (1)
2017年11月 (1)

```
	int i;

	data <<= 8;

	for (i=0; i<8; i++)
	{
		if (data & 0x8000)
		{
			data <<= 1;
			data ^= ccitt16;
		}
		else
		{
			data <<= 1;
		}
	}

	return data;
}
```

现在对单字节0x88求CRC16，我们只需要两字节short型的整数就行了，而不需要象以前必须用long型0x880000，其实不管多少字节，只用short型就能够计算它的CRC16。下面说说怎么求多字节的样验码：

当我们求得一个字节（比如0x88）的CRC校验码后，如果这时又有一个字节(比如0x23)加入，需要求校验码，该怎么办呢？以前得到的是0x880000除以0x11021的余数，但现在需要得到的是0x88230000除以0x11021的余数，以前求得的校验码是不是白费了？不是的，因为当我们得到0x880000除以0x11021的余数（这里余数是0x1080）后，需要求0x88230000除以0x11021的余数，只要将原来的余数0x1080左移8位除以0x11021就得到了0x98000000除以0x11021的余数（想一想为什么），再加上0x230000除以0x11021的余数。这其实就是求0x108000+0x230000除以0x11021的余数.。因此根据这个方法，我们就可以写出求多个字节的CRC算法：

```
unsigned short crc16_ccitt(unsigned char data, unsigned short crc)
{
	unsigned short ccitt16 = 0x1021;

	int i;

	crc ^= (data<<8);


	for (i=0; i<8; i++)
	{
		if (crc & 0x8000)
		{
			crc <<= 1;
			crc ^= ccitt16;
		}
		else
		{
			crc <<= 1;
		}
	}
```


文章分类

```
	return crc;
}

void main()
{
	int i;
	unsigned short crc;
	char data[5] = { 0x71, 0x88, 0x93, 0xa5, 0x13 };

	crc = 0;

	for (i=0; i<5; i++)
	{
		crc = crc16_ccitt(data[i], crc);
	}

	printf("crc is %x", crc);
}
```

好了，讲到这里CRC算法已经全部介绍完了。什么，讲完了？不对呀，我怎么记得CRC程序都有个数组叫CRC表什么的，你这里怎么没有？

呵呵，其实使用CRC表是为了节省一些运算时间，事先将算好的CRC保存在数组里，省得临时再计算，它保存的是0到0xff的CRC码，下面我们来自己生成一个CRC表：

```
unsigned short CRC16_CCITT_TABLE[256];
void init_crc16_ccitt_table()
{
	int i;

	for (i=0; i<256; i++)
	{
		CRC16_CCITT_TABLE[i] = crc16_ccitt(i, 0);
	}
}
```

上面只是一个字节的CRC表，那多个字节的CRC如何计算呢？与前面求多字节的CRC方法差不多，它的代码如下：

```
unsigned short crc16_ccitt_2(unsigned char data, unsigned short crc)
{
	unsigned char c;
	c = crc >> 8;
	crc <<= 8;
	crc ^= CRC16_CCITT_TABLE[data ^ c];
	return crc;
}
```

最后要说的是CRC的正序和反转问题，比如前面ccitt-crc16的正序是0x1021，如果是反转就是0x8408（就是将0x1021倒过来低位变高位）为什么要反转？这是因为数据传输可能是先传低位再传高位（比如串口就是低位在前高位在后）。反转的CRC算法与正序类似，只是需要注意移位的方向相反。

```
unsigned short crc16_ccitt_r(unsigned char data, unsigned short crc)
{
	unsigned short ccitt16 = 0x8408;

	int i;
```


最新评论

1. Re:关于业务主键和逻辑主键
很有帮助
--shadowdoor

2. Re:linux lsof命令详解
老铁写的很详细，666
--season_qd

3. Re:CRC校验算法
你那个想想，为什么就是想不通，指点下，大神。
--crxczz

4. Re:-bash: ./test.sh: /bin/bash^M: bad interpreter: No such file or directory
下载个 linux 的 tomcat 啊， 别用 windows的
--凌海森

5. Re:ServiceLoader详解
老铁，一看你名字你知道英语不错~~~~
--十禾。

阅读排行榜

```
        crc ^= data;

        for (i=0; i<8; i++)
        {
                if (crc & 1)
                {
                        crc >>= 1;
                        crc ^= ccitt16;
                }
                else
                {
                        crc >>= 1;
                }
        }

        return crc;
}

unsigned short crc16_ccitt_r2(unsigned char data, unsigned short crc)
{
        unsigned char c;

        c = crc & 0xff;
        crc >>= 8;
        crc ^= CRC16_CCITT_R_TABLE[data ^ c];
        return crc;
}
```

文章到这里就结束了，随blog附有两个C程序，一个是这篇文章的全部源程序的例子，一个是crc16的查表程序，可是不知道怎么添加附件，就把CRC16表及其例程写在下面：

```
unsigned short CRC16_TABLE[256] = {
        0x0000, 0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
        0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027, 0x0022,
        0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D, 0x8077, 0x0072,
        0x0050, 0x8055, 0x805F, 0x005A, 0x804B, 0x004E, 0x0044, 0x8041,
        0x80C3, 0x00C6, 0x00CC, 0x80C9, 0x00D8, 0x80DD, 0x80D7, 0x00D2,
        0x00F0, 0x80F5, 0x80FF, 0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1,
        0x00A0, 0x80A5, 0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1,
        0x8093, 0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
        0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197, 0x0192,
        0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE, 0x01A4, 0x81A1,
        0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB, 0x01FE, 0x01F4, 0x81F1,
        0x81D3, 0x01D6, 0x01DC, 0x81D9, 0x01C8, 0x81CD, 0x81C7, 0x01C2,
        0x0140, 0x8145, 0x814F, 0x014A, 0x815B, 0x015E, 0x0154, 0x8151,
        0x8173, 0x0176, 0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162,
        0x8123, 0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
        0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104, 0x8101,
        0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D, 0x8317, 0x0312,
        0x0330, 0x8335, 0x833F, 0x033A, 0x832B, 0x032E, 0x0324, 0x8321,
        0x0360, 0x8365, 0x836F, 0x036A, 0x837B, 0x037E, 0x0374, 0x8371,
```

```
        0x8353, 0x0356, 0x035C, 0x8359, 0x0348, 0x834D, 0x8347, 0x0342,
        0x03C0, 0x83C5, 0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1,
        0x83F3, 0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
        0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7, 0x03B2,
        0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E, 0x0384, 0x8381,
        0x0280, 0x8285, 0x828F, 0x028A, 0x829B, 0x029E, 0x0294, 0x8291,
        0x82B3, 0x02B6, 0x02BC, 0x82B9, 0x02A8, 0x82AD, 0x82A7, 0x02A2,
        0x82E3, 0x02E6, 0x02EC, 0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2,
        0x02D0, 0x82D5, 0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1,
        0x8243, 0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
        0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264, 0x8261,
        0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x023E, 0x0234, 0x8231,
        0x8213, 0x0216, 0x021C, 0x8219, 0x0208, 0x820D, 0x8207, 0x0202,
};

unsigned short CRC16R_TABLE[256] = {
        0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
        0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
        0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
        0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
        0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
        0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
        0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
        0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
        0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
        0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
        0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
        0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
        0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
        0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
        0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
        0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
        0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
        0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
        0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
        0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
        0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
        0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
        0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
        0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
        0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
        0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
        0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
        0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
        0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
        0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
        0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
        0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040,
};

unsigned short CCITT_CRC16_TABLE[256] = {
```

```
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
        0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
        0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
        0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
        0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
        0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
        0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
        0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0,
};

unsigned short CCITT_CRC16R_TABLE[256] = {
        0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
        0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
        0x1081, 0x0108, 0x3393, 0x221A, 0x56A5, 0x472C, 0x75B7, 0x643E,
        0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED, 0xCB64, 0xF9FF, 0xE876,
        0x2102, 0x308B, 0x0210, 0x1399, 0x6726, 0x76AF, 0x4434, 0x55BD,
        0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E, 0xFAE7, 0xC87C, 0xD9F5,
        0x3183, 0x200A, 0x1291, 0x0318, 0x77A7, 0x662E, 0x54B5, 0x453C,
        0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEF, 0xEA66, 0xD8FD, 0xC974,
        0x4204, 0x538D, 0x6116, 0x709F, 0x0420, 0x15A9, 0x2732, 0x36BB,
        0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868, 0x99E1, 0xAB7A, 0xBAF3,
        0x5285, 0x430C, 0x7197, 0x601E, 0x14A1, 0x0528, 0x37B3, 0x263A,
        0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9, 0x8960, 0xBBFB, 0xAA72,
        0x6306, 0x728F, 0x4014, 0x519D, 0x2522, 0x34AB, 0x0630, 0x17B9,
        0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A, 0xB8E3, 0x8A78, 0x9BF1,
        0x7387, 0x620E, 0x5095, 0x411C, 0x35A3, 0x242A, 0x16B1, 0x0738,
        0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB, 0xA862, 0x9AF9, 0x8B70,
```

```
        0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C, 0xD3A5, 0xE13E, 0xF0B7,
        0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64, 0x5FED, 0x6D76, 0x7CFF,
        0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD, 0xC324, 0xF1BF, 0xE036,
        0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5, 0x4F6C, 0x7DF7, 0x6C7E,
        0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E, 0xF2A7, 0xC03C, 0xD1B5,
        0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66, 0x7EEF, 0x4C74, 0x5DFD,
        0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF, 0xE226, 0xD0BD, 0xC134,
        0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7, 0x6E6E, 0x5CF5, 0x4D7C,
        0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028, 0x91A1, 0xA33A, 0xB2B3,
        0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60, 0x1DE9, 0x2F72, 0x3EFB,
        0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9, 0x8120, 0xB3BB, 0xA232,
        0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1, 0x0D68, 0x3FF3, 0x2E7A,
        0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A, 0xB0A3, 0x8238, 0x93B1,
        0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62, 0x3CEB, 0x0E70, 0x1FF9,
        0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB, 0xA022, 0x92B9, 0x8330,
        0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3, 0x2C6A, 0x1EF1, 0x0F78,
};

unsigned short get_crc16(unsigned char data, unsigned short crc)
{
        unsigned char c;
        c = crc >> 8;
        crc <<= 8;
        crc ^= CRC16_TABLE[data ^ c];
        return crc;
}


unsigned short get_crc16r(unsigned char data, unsigned short crc)
{
        unsigned char c;

        c = crc & 0xff;
        crc >>= 8;
        crc ^= CRC16R_TABLE[data ^ c];
        return crc;
}

unsigned short get_ccitt_crc16(unsigned char data, unsigned short crc)
{
        unsigned char c;
        c = crc >> 8;
        crc <<= 8;
        crc ^= CCITT_CRC16_TABLE[data ^ c];
        return crc;
}

unsigned short get_ccitt_crc16r(unsigned char data, unsigned short crc)
{
        unsigned char c;
```

```
        c = crc & 0xff;
        crc >>= 8;
        crc ^= CCITT_CRC16R_TABLE[data ^ c];
        return crc;
}
```

分类: java

星火spark
关注 - 0
粉丝 - 8

0 0

+加关注

« 上一篇：SAX解析多层嵌套XML

» 下一篇：Linux 的字符串截取很有用。有八种方法。

posted @ 2016-11-03 18:10 星火spark 阅读(5903) 评论(1) 编辑 收藏

评论

#1楼 2018-09-14 15:13 | crxczz

你那个想想，为什么就是想不通，指点下，大神。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

相关博文：

- CRC 校验算法C#实现二 ——8位校验算法
- CRC-8校验算法
- CRC-32 校验算法
- CRC循环冗余校验算法
- java实现CRC16 MODBUS校验算法

最新新闻：

- 共享电助力车为何在三四线城市先“火”了？
- LinkedIn推出视频直播 主打企业发布财报会议等
- 权威消费者报告：特斯拉Model 3被评选最令人满意汽车
- 亚马逊扩张硬件帝国 新收购家庭路由器厂商
- IBM的AI“机器辩手”即将登场 它究竟如何诞生？

» 更多新闻...