

区块链技术深度剖析

红亚太学链系列课程

第2章 区块链编程基础

讲师：王宝成



红亚科技
HONGYATECH

目录

CONTENTS



序列化



字节序



大数运算



C++编程基础

程序员在编写应用程序的时候往往需要将程序的某些数据存储在内存中，然后将其写入某个文件或是将它传输到网络中的另一台计算机上以实现通讯。

这个将程序数据转化成能被存储并传输的格式的过程被称为“序列化”（**Serialization**），而它的逆过程则可被称为“反序列化”（**Deserialization**）。

序列化：将对象变成字节流（二进制）的形式传出去。

反序列化：从字节流恢复成原来的对象。

所谓对象序列化，实际上就是将内存中运行的对象数据直接存入磁盘。对象在内存中是二进制数据，是一个可以直接执行的实体。使用时不需要再经过类的构造，因为存储的对象就是已经构造好的对象。

而反序列化就是要将存储在磁盘中的对象的二进制数据读入内存，并将数据直接转化为对象，并可以用对象来执行功能。

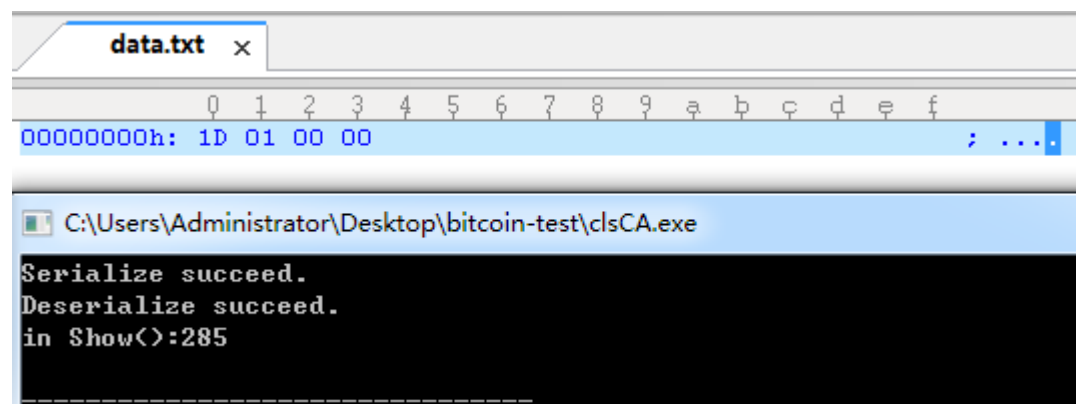
将C++类对象的实例序列化为一串二进制码流是一种较为常用的编程技巧，可以方便地在关系数据库或者文件中存储对象，也可以方便地在网络上传输。

序列化（二进制文件读写）：

- ◆ `FILE * fopen(const char * path,const char * mode);`
参数：
 path：需要打开文件的路径
 mode：文件打开方式—**rb+**、**wb+**（二进制文件读写）
- ◆ `size_t fwrite(const void *buffer, size_t size, size_t count, FILE *file);` [fwrite_struct.cpp](#)
- ◆ `size_t fread(void *buffer, size_t size, size_t count, FILE *file);` [fread_struct.cpp](#)

序列化（二进制类对象）：

- ◆ int 285，内存：0x0000011D，存储：1D010000
- ◆ 将一个类的一个对象序列化到文件



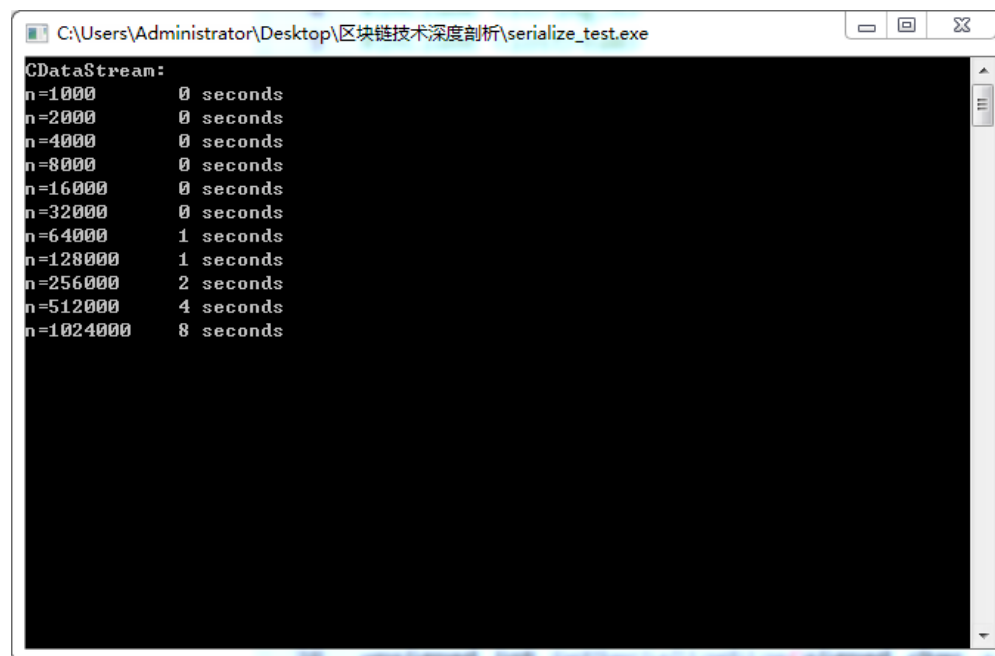
```
data.txt x
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 1D 01 00 00 ; ...

C:\Users\Administrator\Desktop\bitcoin-test\clsCA.exe
Serialize succeed.
Deserialize succeed.
in Show():285
```

- ◆ 使用UltraEdit十六进制编辑模式查看存储的数据文件

比特币源码中的序列化:

◆ [serialize_test.cpp](#)



```
C:\Users\Administrator\Desktop\区块链技术深度剖析\serialize_test.exe
CDataStream:
n=1000      0 seconds
n=2000      0 seconds
n=4000      0 seconds
n=8000      0 seconds
n=16000     0 seconds
n=32000     0 seconds
n=64000     1 seconds
n=128000    1 seconds
n=256000    2 seconds
n=512000    4 seconds
n=1024000   8 seconds
```

序列化对象实例还可以使用工具，例如：[Boost](#)、PB、MFC等



红亚科技
HONGYATECH

目录

CONTENTS



序列化√



字节序

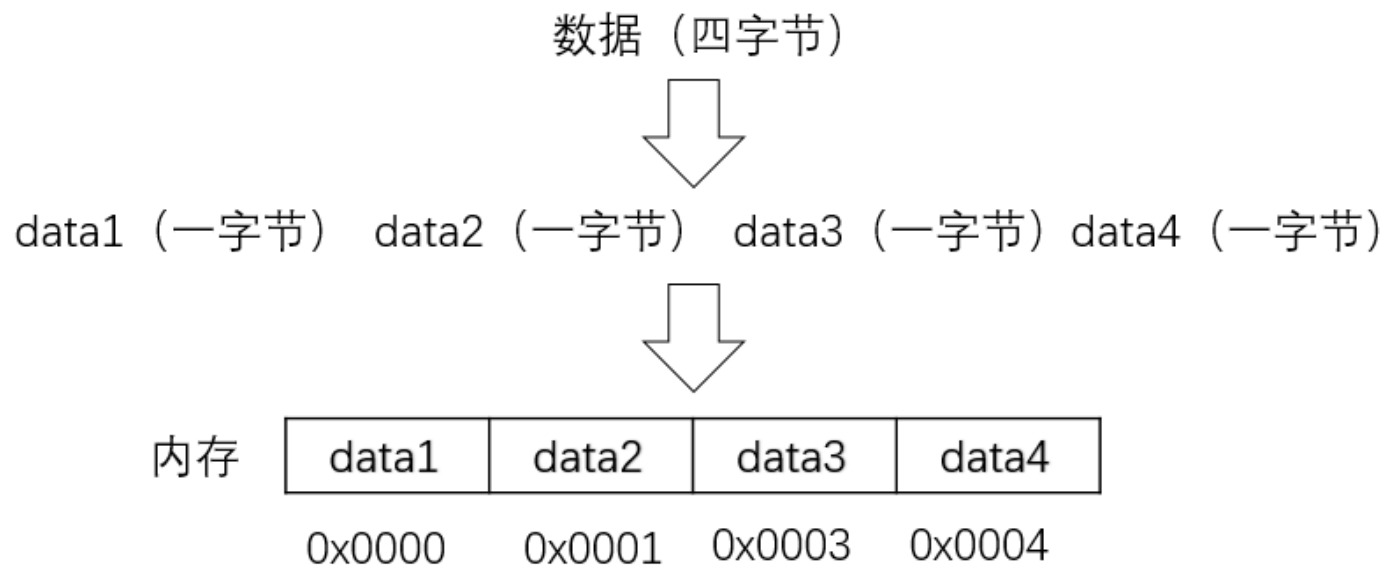


大数运算



C++编程基础

字节序是指多字节数据在计算机内存中存储或者网络传输时各字节的存储顺序。



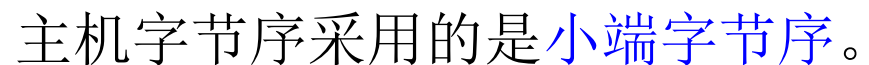
主机字节序，就是主机内存中数据的处理方式，可以分为两种：

- ◆ 小端字节序（little-endian）：将高位数据存储在低内存地址中，低位数据存储在低内存地址中。
- ◆ 大端字节序（big-endian）：将高位数据存储在低内存地址中，低位数据存储在低内存地址中。

小端字节序，更加符合人们的思维形式，因为它遵循“低放低，高放高”的原则。

大端字节序，最直观的字节序。保持了值的原本顺序。

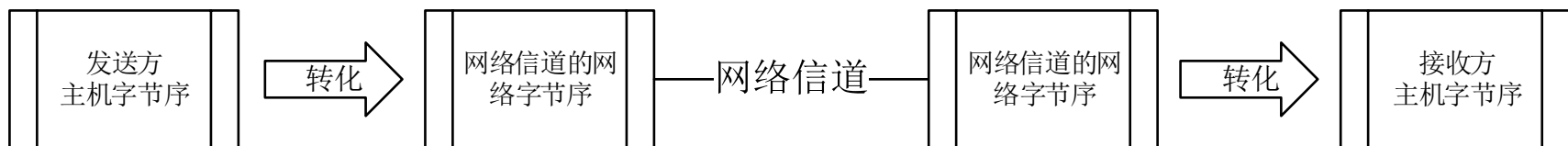
高	低
0x12345678	



网络字节序是TCP/IP中规定好的一种数据表示格式，从而可以保证数据在不同主机之间传输时能够被正确解释，其采用的字节序是**大端字节序**。

比特币网络节点中的区块文件采用的是**小端字节序**。

在网络数据传输过程中，发送方首先将**主机字节序转化为网络字节序**，通过网络信道传输，接收方收到网络字节，将其转化为**主机字节序**。



相关函数如下：

- ◆ htons 把unsigned short类型从主机序转换到网络序
- ◆ htonl 把unsigned long类型从主机序转换到网络序
- ◆ ntohs 把unsigned short类型从网络序转换到主机序
- ◆ ntohl 把unsigned long类型从网络序转换到主机序

程序实例：htonl_demo.cpp

结果截图：

```
主机字节序:78  56  34  12
网络字节序:12  34  56  78
-----
```

创世区块数据如下：

```
GetHash()      = 0x000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
hashMerkleRoot = 0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
txNew.vin[0].scriptSig      = 486604799 4 0x736B6E616220726F662074756F6C69616220646E6F6C
txNew.vout[0].nValue        = 5000000000
txNew.vout[0].scriptPubKey = 0x5F1DF16B2B704C8A578DOBBAF74D385CDE12C11EE50455F3C438EF4C
block.nVersion = 1
block.nTime     = 1231006505
block.nBits     = 0x1d00ffff
block.nNonce    = 2083236893
```

存储转换：

- ◆ 时间戳十进制：1231006505，十六进制：495FAB29
- ◆ 转换为小端字节序：29AB5F49

创世区块存储数据:

00000000	01 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000010	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000020	00 00 00 00 3B A3 ED FD	7A 7B 12 B2 7A C7 2C 3E;fíýz{.²zÇ,>
00000030	67 76 8F 61 7F C8 1B C3	88 8A 51 32 3A 9F B8 AA	gv.a.Ė.Ā^ŠQ2:Ÿ,*
00000040	4B 1E 5E 4A 29 AB 5F 49	FF FF 00 1D 1D AC 2B 7C	K.^J)«_Iÿÿ...¬+
00000050	01 01 00 00 00 01 00 00	00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 FF FF	FF FF 4D 04 FF FF 00 1DÿÿÿÿM.ÿÿ..
00000080	01 04 45 54 68 65 20 54	69 6D 65 73 20 30 33 2F	..EThe Times 03/
00000090	4A 61 6E 2F 32 30 30 39	20 43 68 61 6E 63 65 6C	Jan/2009 Chancel
000000A0	6C 6F 72 20 6F 6E 20 62	72 69 6E 6B 20 6F 66 20	lor on brink of
000000B0	73 65 63 6F 6E 64 20 62	61 69 6C 6F 75 74 20 66	second bailout f
000000C0	6F 72 20 62 61 6E 6B 73	FF FF FF FF 01 00 F2 05	or banksÿÿÿÿ..ó.
000000D0	2A 01 00 00 00 43 41 04	67 8A FD B0 FE 55 48 27	*....CA.gŠÿ° þUH°
000000E0	19 67 F1 A6 71 30 B7 10	5C D6 A8 28 E0 39 09 A6	.gñ q0 . .\Ö'' (à9.!
000000F0	79 62 E0 EA 1F 61 DE B6	49 F6 BC 3F 4C EF 38 C4	ybdê. aB¶Iö%?Lİ8Ä
00000100	F3 55 04 E5 1E C1 12 DE	5C 38 4D F7 BA 0B 8D 57	òU.š.Á.Þ\8M÷°...W
00000110	8A 4C 70 2B 6B F1 1D 5F	AC 00 00 00 00	ŠLp+kñ. _¬....

创世区块存储说明：

```
01000000 - version
0000000000000000000000000000000000000000000000000000000000000000 - prev block
3BA3EDFD7A7B12B27AC72C3E67768F617FC81BC3888A51323A9FB8AA4B1E5E4A - merkle root
29AB5F49 - timestamp
FFFF001D - bits
1DAC2B7C - nonce
01 - number of transactions
01000000 - version
01 - input
0000000000000000000000000000000000000000000000000000000000000000FFFF - prev output
4D - script length
04FFFF001D0104455468652054696D65732030332F4A616E2F32303039204368616E63656C66F72206F6E206272696E66206F666207365636F6E64206261696C6F7574206666F722062616E6B73 - scriptsig
FFFFFFFF - sequence
01 - outputs
00F2052A01000000 - 50 BTC
43 - pk_script length
4104678AFDB0FE5548271967F1A67130B7105CD6A828E03909A67962E0EA1F61DEB649F6BC3F4CEF38C4F35504E51EC112DE5C384DF7BA0B8D578A4C702B6BF11D5FAC - pk_script
00000000 - lock time
```

- ◆ 目标值：0x1D00FFFF，存储：FFFF001D
- ◆ 随机数：2083236893，0x7C2BAC1D，存储：1DAC2B7C
- ◆ 50 BTC：5000000000，0x12A05F200，存储：00F2052A01*



红亚科技
HONGYATECH

目录

CONTENTS



序列化√



字节序√



大数运算



C++编程基础

非对称密码体制**RSA**的安全性依赖于对大素数进行因数分解的耗时性。

RSA对于大数的要求，精度一般是少则数十位，多则几万位。同样，**ECC**安全要求至少是**160**位。

C语言编程中用到的数据类型，即使是最长的**long long**型数据，都不能满足**RSA**对于数据精度的要求。

怎样定义一个长度符合要求的数据来保证**RSA**、**ECC**等密码算法的安全性呢？这样长度的数据又该如何存储？怎样计算呢？

计算机分解一个1000位的大数所需要的时间更是达到了 6^{109} 年。

一般认为，对于当前的计算机水平，RSA选择1024位、ECC选择160位长的密钥就可以保证密码算法的安全性。

因此，为了保证密码算法的安全性，应该将大数计算落实到足够长的位数。

大多数密码算法都需要足够长位数的密钥来保证其安全性。因此，大数计算不仅是学习RSA、ECC的基本前提，也是掌握密码学中其他密码算法必备的技能之一。

大数运算不同于普通运算的关键之处在于大数据的存储方式。

为了满足大数据对于长度的要求，可以选择用字符串存储大数据。

用字符串存储大数据，即可以满足大数据对于存储长度的要求，也因为字符串便于遍历的优点，方便了大数据的按位计算。

比特币源码中采用整型数组来表示大数据（160位、256位）。

```
template<unsigned int BITS>
class base_uint
{
protected:
    enum { WIDTH=BITS/32 };
    unsigned int pn[WIDTH];
```

例如大数存储在内存中的存储方式：

- ◆ 定义一个大数str1 : `char str1[200]="9876543210987";`
- ◆ 因为数字0-9在计算机中的ASCII码为 48-57
- ◆ 即数字0在内存中实际存储的内容为一个8位（最后一位为符号位）二进制数：00110000
- ◆ 所以大 数str1在内存中实际为：

```
00111001  00111000  00110111  00110110  00110101
00110100  00110011  00110010  00110001  00110000
00111001  00111000  00110111|
```

大数运算的实现方法主要有以下几种：

- ◆ 用字符串表示大数。将大数用十进制字符数组表示，然后按照“竖式计算”的思想进行计算。
- ◆ 将大数表示成一个 n 进制数组。 n 的取值越大，数组的大小越小，这样可以缩短运算的时间及空间复杂度，提高算法的效率。



假设在加法中两个操作数都是大于0的。按照“竖式计算”的思想，首先将两操作数低位对齐

然后从最低位开始按“位”相加，当“位”相加的结果大于10时做进位处理（`carry=1`），否则不进位（`carry =0`）。例如两个大数如下：

- ◆ `char str1[200]="9876543210987";`
- ◆ `char str2[200]="1234567890";`
- ◆ `int carry = 0 ; //进位标志`

两个大数加法：

- ◆ 首先将str2前面补上3个0，使得str1与str2位数相同；
- ◆ 接下来从最低位（字符串最后一位）按位相加，即 $\text{str1}[i] + \text{str2}[i] + \text{carry} - 96$
- ◆ 此处的i从字符串长逐渐递减到0（遍历字符串），最后减去96因为数字0-9在计算机中的ascii码为 48-57，减去96得到的是实际中的数值（0-18之间）。
- ◆ 结果大于10将其减10，并将carry置为1，将结果加上48（转换为字符）存入大数result中；

大数乘法也与加法类似，整体思想为大数一从低位到高位分别乘大数二，将结果依次相加，此时进位数值与加减法不同，不只局限于0、1，进位数值随着乘法结果可取为0-8。余下的处理方法与大数加减法类似。

大数乘方运算就是连续调用大数乘法运算来实现的。连续调用可以通过循环或者递归的方式完成大数乘方计算。

大数减法与加法类似，首先比较两个大数的大小，将小的（**str2**）前面补上3个0，使**str1**与**str2**位数相同；

接下来从最低位（字符串最后一位）按位相减，即 $\text{str1}[i] - \text{str2}[i] - \text{carry}$ ，此处的*i*从字符串长逐渐递减到0（遍历字符串）。

得到的结果大于0将其 + 48（转换为字符）存入大数**result**中；得到的结果小于0，将**carry**置为1，将结果加上10（转换为实际大小） + 48（转换为字符）存入大数**result**中。

大数取模运算的运算方法不尽相同，示例代码采用的连续求差的方式实现大数取模运算。

大数一反复与大数二做差，直到结果小于大数二的数值为止，得到最后结果。

大数运算: bignum_demo.cpp

比特币中的256和160位大数:

uint256_operator.cpp

```
大数1为 = 9876543210987
大数2为 = 1234567890
大数1 > 大数2
大数1 + 大数2 = 9877777778877
大数1 - 大数2 = 9875308643097
大数1 * 大数2 = 12193263112482045407430
大数1 MO 大数2 = 90987
999 ^ 10 = 990044880209748209880044990001
计算完毕! 输入任意键退出!
```

[illegible]



红亚科技
HONGYATECH

目录

CONTENTS



序列化√



字节序√



大数运算√



C++编程基础

面向对象程序设计之总结一.ppt

面向对象程序设计之总结二.ppt

面向对象程序设计之总结三.ppt



红亚科技
HONGYATECH

目录

CONTENTS



序列化√



字节序√



大数运算√



C++编程基础√

熟练掌握本章示例代码

阅读比特币源中本章涉及的部分

下载OpenSSL、Boost源代码并编译



红亚科技

H O N G Y A A T E C H

北京红亚华宇科技有限公司（简称：红亚科技），是一家专注于高校信息技术实训教育平台研发的创新型科技公司，公司旗下的信息安全教学平台包括四大安全实训教学系统：网络攻防实训及演练系统、网络攻防靶场实战系统、工控安全实训教学系统，[区块链实训教学系统](#)，目前公司的信息安全实训教学系统在国内已经有400多所高校用户，是国内最大的信息安全实训教育科技公司，2016年开始研究区块链，2018年1月率先推出“红亚太学链”区块链实训教学系统，区块链实训系统涵盖“区块”及“链”的可视化分析与展示、创世区块的生成、P2P网络、交易与脚本、智能合约、“挖矿”等模块，共计100多个实训任务。



网络攻防



网络靶场



工控安全



区块链

THANK YOU VERY MUCH

感谢聆听

