QuantBox行情数据存储方案

作者：伍侃

20150116

行情数据以前用csv文本格式进行保存，太大，使用7z等压缩工具压出来还是很大。并且csv格式解析时比较耗时。

行情数据中会出现大量的重复数据，比如说涨跌停每天都是一样的、交易日也每天都是一样的。如果发现当前数据的某个字段与上一笔一样就跳过，这样是否就能省不少空间呢？

所以一开始做了简化的csv格式，文件的确变小了很多。如：

1,2,3,4,5,6,7

4,,,,,,

5,2,,,,

进一步的，用文本来表示数字还不如用二进制，用二进制就得解决以下问题：如何做字段的分隔，如何表示与上一条数据的某个字段一样。

二进制的保存最后采用的是Google Protocol Buffer。网上可以找到此存储格式的优点，在这就不多说，主要分析下，用来存行情有什么优点。

* Key-Value模式：不需要使用分隔符。对于使用默认值的数据可以不记录。比如说默认值为0/null就不用记录。
* Varint：对于int32一般是占用4个字节，但采用Varint后，很小的数字可以1字节表示。比如小于128的数就可以1字节，每个byte的最高1bit表示后一个byte是否是当前数字的一部分。
* ZigZag编码：对于负数，占用字节就太多了，如果正负数交错表示，这样再结合Varint，负数就能很小的进行表示了。如：0->0,-1->1,1->2,-2->3

以上三个优点再加上自己的差分算法，就能很好的保存数据了，操作方式简述如下：第一笔保存全量，以后的数据都只保存与上一笔变化的量，没有变化就记为0/null,这样就会跳过为0的部分,数据尽量变小。

但实际使用时还是有一些要特性要注意：

* Key不但表示了Field编号，也表示了Field的类型。所以它并不能表示128种编号。所以出现频率高的编号尽量放在1-15，16开始就是使用2字节了。
* 字符串、对象：按Key-Length-Value进行存储，所以当对象中的所有值都是0时，还是至少占用两个字节Key-Length。如：0x12 0x00,所以当对象中所有元素都是0时，应当直接让对象=null.
* Double类型永远是8字节，所以尽量转成int或long

分析后认为数据分两种：

1.本身密集，要通过差分算法转成稀疏的，

2.本身稀疏，直接保存即可

从交易所接收到的行情一直变化来变化去，肯定要用上差分算法，这点是不用怀疑的，关键是哪些数据是稀疏，并且只要直接保存，这点就有争论了。

目前肯定除权除息数据是稀疏的。因为很久很久才有一条，第二天就没了。

静态数据，如涨跌停，每天只更新一次，但CTP行情中每tick都有，先算成密集的。差分后效果很好。

Bar数据，在tick中用来存当天的最高价与最低价，算成密集的，差分后效果也很不错。

配置数据到底是算密集的还是稀疏的的呢？目前划为稀疏的，如果将不同品种的行情存在一个文件中将导致配置数据来回切换是不合适的。

行情中基本都是double类型的数据，如何转成int呢？很显然大家第一步想到的就是利用TickSize，也就是存的数据其实是Price/TickSize，如：100/0.2=500。用这种方法可以将大部分数值转成int

结算价一开始也计划用此方法存，结果发现IF的交易日结算价与交割日结算价不是那么一回事，交易日结算价是1位小数0.2，交割日结算价是2位小数0.01，所以结算价这里转换方法是\*100，然后再除TickSize。(100是SettlementPriceMultiplier)

同理，平均价也是\*100/TickSize. (100是AveragePriceMultiplier)

成交额的变化一定是最小变动金额的整数倍，所以实际值是/(tickSize\* ContractMultiplier)，例如IF是0.2\*300=60

同时保存了TradingDay和ActionDay使用int,如20141125

时间也是一个很关键的地方，如何保存时分秒和毫秒呢？一开始保存的方法两部分HHmmss/fff。做了差分后，第一部分不明显，每秒都更新，变化的值不大，但至少占用两字节。而fff是每次都变，还每次变化500，占用三字节。这种保存方法基本每笔就5字节了。而实际上fff中后面两ff大部分情况下是00。如何简化这部分呢？最后的解决方法是用三部分来表示。HHmm/ssf/ff。HHmm是每分钟才变动一次，ssf每次变动，但一般是变化5，占用两字节，在每分钟才出现一次占用三字。ff由于大部分是0，不占用。

后来发现ssf在IF合约中大量的出现5这个值，所以决定得到ssf后再减去预定义的5，这样变成0后又可少2个字节。

多档行情如何保存也是关键，一档行情就要占用4个编号，5档就占用20了，再加上前面提到的时间、成交量、交易日等信息，光编号就得50多个了，这怎么行。所以很自然就想到使用对象，这样每次只多Key-Length一般两字节。当然在对象中内部还是要注意不能大于15。这样算下来一个对象中最多也只能记三档，好吧，只能用对象的嵌套了。每次记三档，三档不够，再new一个新的三档接着。

对于买卖价的差分算法也有一定的技巧，买卖价每个价位基本是按TickSize相临，所以可以完全省去价位的记录。买卖一与最新价相减，得到差价，这个得原样记录，而买卖一与买卖二等一类的记录时，价差要再减1得到0，因为它们默认就会得到1，不会出现0。

买卖量则是与上一笔中同档位的量进行差分，就一档会变化较大，其它档变动小，所也比较省空间。

有些数据一天就变动一次，我们称为静态数据，这些单独放在了一个对象中。如涨跌停，结算价。

Bar数据也是很关键的地方。从CTP中的数据就有开高低收，这与K线数据正好对应，所以用此数据结构同时记录Bar数据也是没有问题的。

Bar数据的差分如何做呢？按说第一笔的Close与第二笔的Open有很大的概率相等。也就是到第二笔的Open是与上笔的Close做差分还是与上笔的Open做差分。最后还是考虑是与上笔的Open，因为存CTP中TICK数据的机会更多，而Bar数据就算记录，量也没有Tick数据多。

前面介绍了那么多，最核心的地方就是TickSize，它不正确，一切都是空的，为了数据的自解析，TickSize要保存，这个数按double保存肯定不成，最后是\*10000转成int.也就是第一个TickSize存，后面的差分数据TickSize就不存即可。

现在一个数据文件的存储结构就设计得差不多了，我们现在写代码测试下是否可行。

* 把源数据的Tick转成PbTick
* 多个PbTick进行差分
* 顺序保存

读取测试，发现读出来的数据居然是大量数据中的最后一笔。原来是保存的数据没有办法区分每个Tick的分隔，这样一读就读到最后，然后按Key-Value，同样的Key被多次更新，最后更新到了最后一次出现的key.

如何解决呢？当然是Length了。改成Length-PbTick，一个长度，一块Tick即可。

如果此存储方案真的有效，我按日，按合约来存数据，每天就会生成上百个几KB到几MB的小文件，这对于文件分享是很不利的。还好此种文件格式可以直接用copy 1.bin /b + 2.bin /b 3.bin的方式来合并。如果这样合并了，表示我的数据中会出现：快照，差分，差分…快照，差分，差分…。如何知道其中的某一个数据是快照呢？

前面我们已经提到了几个关键参数，tickSize, ContractMultiplier等，这些参数一开始是写在了编解码器中，最后定一个参数配置对象，这些核心的参数放在对象中。约定当这个对象不为空时表示是快照，需要将当前的PbTick直接返回。这种方法对于国外的价格范围不同TickSize也不同进行了支持，只要在这时记录下新tickSize的一次快照即可。

如果你不使用成交金额数据，ContractMultiplier可以不用设置使用默认的1即可，这样成交金额每次变动为TickSize的整数倍，但这样数字太大，所以只要使用成交金额的情况下一定要设置正确合约乘数。比如说IF是300

这种方案的局限。不能随机访问，必需文件加载完才能解析。对于行情交易来说问题不大。用户想用时，至少返回的整天数据，而不是说你只想要其中一小时的我就只返回一小时。过滤由客户端来做。

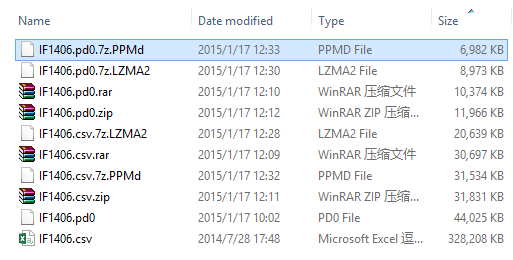
不能直接编辑，只要编辑，得新生成文件。

生成文件后，可以对其再次用专业压缩软件进行压缩，比如说7z,zip。这样文件会小很多。

由于这种行情格式中的数据有它自己的特点，再用不同的压缩算法达到的效果肯定差距很大。所以试验了一下哪种压缩算法比较好。以下都使用最大压缩方式

将csv中的一档行情数据完全转成pd0格式，csv是IF1406从2013年10月21号到2014年6月20号，一共5059003条数据，基本的一档行情。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | csv | Pd0 | csv | Pd0 | 压缩需内存 | 解压需内存 |
| 原始文件 | 320MB | 42.9MB |  |  |  |  |
| 7z/PPMd | 30.7MB | 6.81MB | 42s | 12s | 223MB | 194MB |
| 7z/LZMA2 | 20.1 MB | 8.76 MB | 103s | 31s | 2221MB | 66MB |
| Rar | 29.9MB | 10.1MB | 38s | 9s |  |  |
| Zip | 31.0MB | 11.6MB | 6s | 1s |  |  |



对于纯文本压缩，LZMA2编码明显压缩率高，但耗时时完全无法接受。

对于这种特殊的二进制编码，PPMd又明显好于其它几种的压缩效率，耗时也在可接受的范围。

最后推荐7z->极限压缩->PPMd->192MB是最优的方案。

参考文献

Google Protocol Buffer 的使用和原理

<http://www.ibm.com/developerworks/cn/linux/l-cn-gpb/>

PPM压缩算法的分析与实现

<http://yeye55blog.blog.163.com/blog/static/197241021201110715759577/>