CSON 数据格式定义

1. CSON 定义目的

后端 数据查询引擎(query)的投影，全表检索，内置更新，聚合函数等基础功能，要求在数据查询和修改操作时，对存储的数据能够进行高效的查找和修改。基于这个需求，在 BSON 数据的基础上重新定义一种的新类型的数据格式 CSON，可以满足效率要求。

2. CSON 数据格式

CSONArray 的数据结构定义如下：

<table>
<tr><td>Len</td><td>Num</td><td>Type</td><td>Index0</td><td>Type</td><td>Index1</td><td>Type n</td><td>Index n</td><td>V1</td><td>V2</td><td>V.n</td><td>EOF</td></tr>
<tr><td>4Byte</td><td>4Byte</td><td>1Byte</td><td>Offset0 4Byte</td><td>1Byte</td><td>Offset1 4Byte</td><td>1Byte</td><td>n</td><td>元素 1</td><td>元素 2</td><td>n</td><td>结束符</td></tr>
<tr><td></td><td></td><td colspan="6">索引区</td><td colspan="3">Buffer 区</td><td></td></tr>
</table>

- Len 表示 CSON 数据段的长度，和其他可变长度的类型相同，不包括本身所占用的 4Byte 长度，但包括结束符 EOF 占用的长度
- Num 表示 CSON 数据段中总计元素个数，占用 4Byte。
- Type 表示该元素的类型，占用 1Byte。
- Index 表示该元素的索引或值本身，占用 4Byte。

  如果该元素是简单类型并且是固定长度且长度是不大于 4 四字节的，Index 存放其值。如果元素不满足上面条件，那么 index 存放的是该元素在 BUFFER 区域的相对偏移量。相对偏移量是指对于 Buffer 区域的起始位置而言。
- Value 表示该元素的真实值，不固定长度。数据的前缀一般都包含长度信息。
- EOF 表示 CSON 数据的结束符，占用 1Byte。简单且长度固定的数据类型无 EOF。

嵌套数据示例：

当上图 BSONArray 的 index2 的 type 为 BSONArray 类型时。

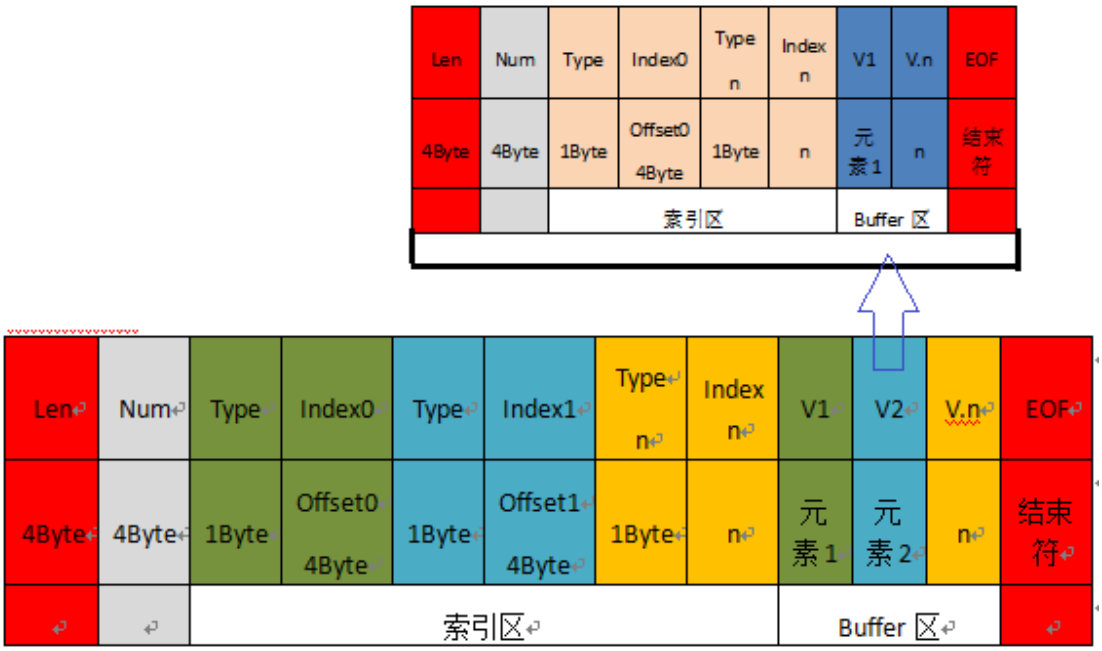
Type = 0x04;

Index = v2 在 buf 区的偏移量：v2 的起始位置 - v1 的起始位置。

V2 = BSONArray(如上图定义)。

当 v2 中的某个元素还是为 BSONArray 类型，数据结构定义方法同此例。

图示如下：



3. CSON 数据类型

element ::=

|	"\x01" double	//Floating point
|	"\x02" string	//UTF-8 string
|	"\x03" document	//Embedded document
|	"\x04" document	//Array
|	"\x05" binary	//Binary data
|	"\x06"	//Undefined — Deprecated
|	"\x07" (byte*12)	//ObjectId
|	"\x08" "\x00"	//Boolean "false"
|	"\x08" "\x01"	//Boolean "true"
|	"\x09" int64	//UTC datetime
|	"\x0A"	//Null value
|	"\x0B" string string	//Regular expression
|	"\x0C" string (byte*12)	//DBPointer — Deprecated
|	"\x0D" string	//JavaScript code
|	"\x0E" string	//Symbol
|	"\x0F" code_w_s	//JavaScript code w/ scope
|	"\x10" int32	//32-bit Integer
|	"\x11" int64	//Timestamp
|	"\x12" int64	//64-bit integer
|	“\x13”	//Decimal
|	“\x14”	//Null Type
|	“\x15”	//Byte
|	“\x16”	//16-bit Interger
|	“\x17”	//Single floating point
|	"\xFF"	//Min key
|	"\x7F"	//Max key

```
string      ::=   int32 (byte*) "\x00"      String
cstring     ::=   (byte*) "\x00"            CString
binary      ::=   int32 subtype (byte*)     Binary
subtype     ::=   "\x00"     Binary / Generic
        |     "\x01"     Function
        |     "\x02"     Binary (Old)
        |     "\x03"     UUID
        |     "\x05"     MD5
        |     "\x80"     User defined

code_w_s    ::=   int32 string document     Code w/ scope
```

红色定义部分为增加的 CSON 数据类型。

Null Type 类型表示该数据不存在，只是为了实现数组结构的数据定义，补齐数组的元素空缺(数组的索引就是 name 编号)。

4. CSON 与 BSON 区别

- CSON 的数据结构假设所有的查询元素都存在，按照数组结构定义，以便直接定位元素，数组的访问索引就是该元素的 name，所以取消了 BSON 的 name 定义。为了区分该元素是否真的存在，增加了 Null Type 类型，读取到此类型表示该元素不存在。
- CSON 的数据类型基本全部接收 BSON 定义的数据类型，除 JavaScript code w/ scope 类型之外全部支持。
- CSON 数据同样支持数据嵌套，CSONArray 类型的子元素可以是 CSONArray 类型。
- 重新定义嵌套元素的类型 BSONArray,都改为 CSONArray 类型。
- 取消 BSON 中的 Embedded document 类型，使用 CSONArray 替代。
- 取消 BSON 中的 CString 类型，全部使用 string 类型替代。因为 CString 无长度信息,在读取时字符串时使用 string 类型会提高部分效率。
- 取消使用的数据类型的 BSON code 暂时保留，以便以后扩展。
- 增加一些需要的数据类型.