#### 1时序数据库

##### **1.1基础**

下面简单介绍一下时序数据库的设计思路：

**1、基本概念**

**时序数据（TSDB）** ：时间序列数据库 （Time Series Database , 简称 TSDB），它将数据按照 时间顺序存储 。TSDB 数据具有时间戳，数据存储量大。

**时序记录（TSL）** ：TSL (Time series log)，是 TSDB 中每条记录的简称。

**FDB管理结构：**管理和记录每个时序数据库使用情况的变量。

**2、数据库的结构**

新建一个时序数据库需要分配一段大于等于4扇区的Flash空间，其存储结构如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| **大小** | 1 sector | 1 sector | SUM-2 sector |
| **区域** | FDB管理结构 | 备份恢复 | 时序记录存储 |

3、新增时序记录：先将时序记录存储到“时序记录存储”区域，然后更新 FDB 管理结构。

4、查询时序记录：查询接口查询到时序记录之后将调用应用模块的回调函数，将时序记录的信息传递给回调函数处理，无需关心时序记录中用户数据的结构。

5、在触发容错或者“FDB管理结构”区域需要执行擦除操作时，系统会使用到“备份恢复”区域。

FDB句柄结构如图2.1。

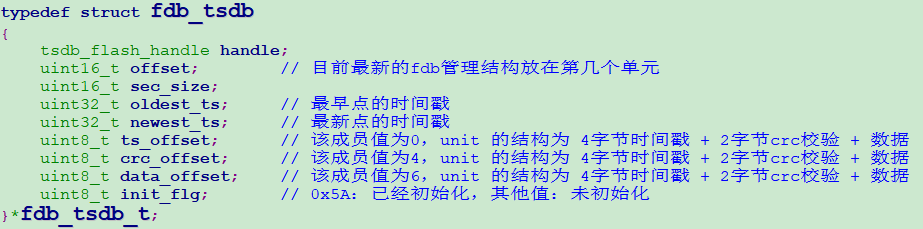


图2.1

**handle**：最新的FDB管理结构，上电后从Flash中读取到并缓存在该成员变量中，每次更新FDB管理结构直接在此基础上修改，然后再同步到Flash中。

**offset**：最新的FDB管理结构存储在“FDB管理结构”区域的第几个单元，更新FDB管理结构之后将其存储在第offset+1个单元。

**sec\_size**：扇区的大小，由宏FDB\_SECTOR\_SIZE控制。

**oldest\_ts**：最旧时序记录的时间戳，增加时序记录时会维护该值。

**newest\_ts**：最新时序记录的时间戳，增加时序记录时会维护该值。

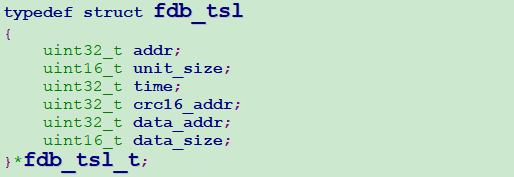
**ts\_offset**：时间戳在每个时序记录中的相对位置，目前为0。

**crc\_offset**：CRC16在每个时序记录中的相对位置，目前为4。

**data\_offset**：用户数据在每个时序记录中的相对位置，目前为6。

**init\_flg**：0x5A：已经初始化，其他值：未初始化；根据该值决定是否可以进行增加时序记录、查询时序记录等操作。

**时序记录的信息结构如下图**



**addr**：时序记录的起始地址。

**unit\_size**：数据单元的大小。

**time**：时序记录的时间戳。

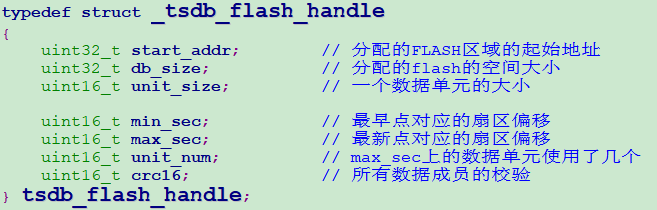
**crc16\_addr**：用户数据的crc16校验的起始地址。

**data\_addr**：用户数据的起始地址。

**data\_size**：用户数据的大小，校验是用。

##### **1.2 FDB管理结构**

一个FDB管理结构的大小为18字节，如图2.2。

图2.2

* 将记录FDB数据存储单元使用情况的变量封装成一个结构体变量；开辟一个扇区的Flash空间来存储这些记录变量（下文统一称其为FDB管理结构），采用增加时序记录就更新的策略。

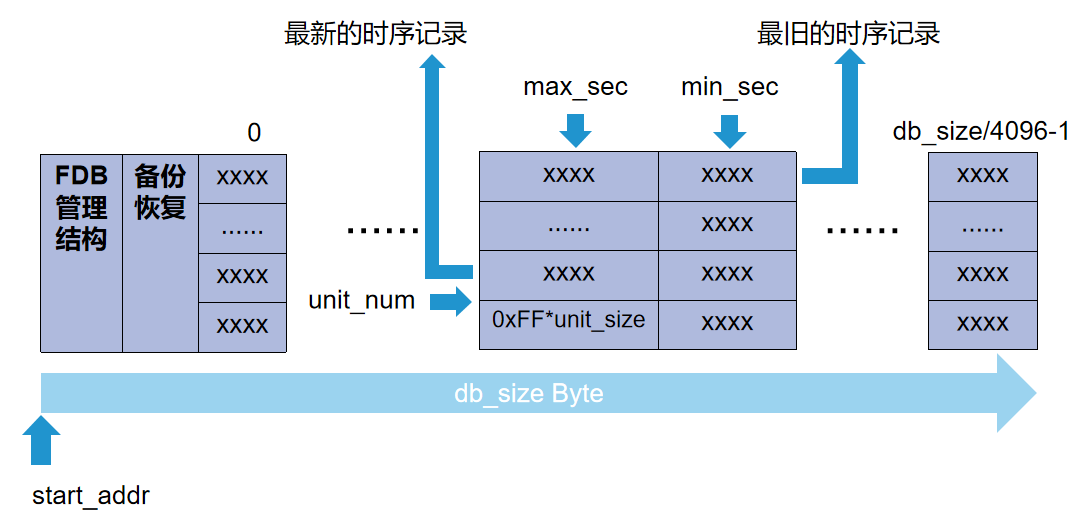


图2.3

图2.3是节点分配的FDB存储单元，会存在一个数据单元无法使用上的情况

**FDB管理结构各个成员的含义**：

**start\_addr**：数据库的起始地址（这个地址必须是扇区对齐）

**db\_size**：整个数据库的大小，包括“FDB管理结构”和“备份恢复”区域

**unit\_size**：数据单元的大小，一个数据单元存储一条时序记录，该大小在新建数据库的时候指定（unit\_size=指定值+6），指定值的范围为[14, 4090]。

**min\_sec**：最旧的时序记录所在的sector，最旧的时序记录在该sector的第一个数据单元

**max\_sec**：最新的时序记录所在的sector

**unit\_num**：max\_sec扇区使用了多少个数据单元，最新的时序记录在该sector的第unit\_num-1个数据单元；新增的时序记录存放在第unit\_num个数据单元；该值必须<=(sector大小/unit\_size)，如果新增时序记录的时候该值等于(sector大小/unit\_size)，则max\_sec需要往后偏移一个sector，然后将新增的时序记录存储在更新后的max\_sec扇区的第0个数据单元，unit\_num置为1,。

**crc16**：所有数据成员的校验码。作用：用于判断FDB管理结构是否是正确的；可能会遇到写入的过程中刚好掉电的情况，导致写入不完整。

###### **1.2.1 FDB管理结构的存储**

为了减少Flash的擦除次数，采用以下两种方式进行优化：

**1、空间换次数**

每个数据库单独分配一个扇区存储FDB 管理结构，则每新增一条时序记录就需要擦除一次该扇区。例如，在一天需要存储 144 条时序记录的应用场景中，一天就需要擦除 144 次，1000 天则需要执行 144000 次擦除操作。虽然这种方式可以减少频繁擦除的次数，但擦除频率依然过高。

**2、利用剩余的空间**

如果一个扇区的 Flash 只存储一个 FDB 管理结构，后续的空间将被浪费。为了减少 Flash 擦除次数，可以将这些空闲空间利用起来。由于未使用的区域仍然为 0xFF，可以将一个扇区的 Flash 划分为 227 个 FDB 管理结构大小的单元（18 \* 227 = 4086 字节）。在更新 FDB 管理结构时，非第一次更新无需擦除操作，只需将新的 FDB 管理结构写入下一个单元。当所有单元都被写满后，才执行一次擦除操作。

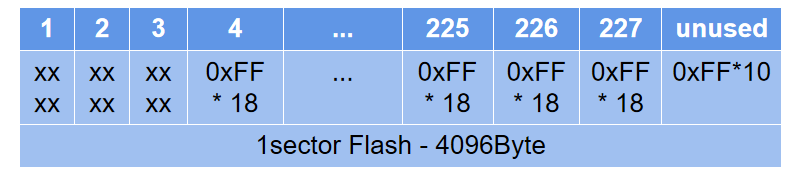


图2.4

图 2.4 中所示的情况表明，此时第 3 个单元中的数据为最新的 FDB 管理结构数据。当需要更新 FDB 管理结构时，数据将直接写入第 4 个单元。如果使用到了第 227 个单元，下一次更新 FDB 管理结构时，就需要先执行扇区擦除操作，然后将新的 FDB 管理结构写入第 0 个单元（这种情况将会触发备份和恢复机制）。

在相同的应用场景下，第一种做法可以将 Flash 擦除次数降低到每天 144 次；而配合上第二种做法则将擦除次数降低至第一种做法的 1/227，即每天擦除 144/227 次。经过 1000 天，擦除次数将为 1000 \*( 144 / 227) = 634 次， 10 年（3660 天）内的擦除次数则为 366 \* 10 \* (144 /227) = 2321 次。

###### **1.2.2备份和恢复机制**

**1、从“FDB管理结构”区恢复**

在写入 FDB 管理结构时，如果恰好发生掉电，可能导致数据没有完整写入（实际上时序记录已经成功存储）。当上电读取 FDB 管理结构时，如果遇到校验失败的 FDB 管理结构，系统会继续向前查找，直到第 1 个单元。如果找到有效的 FDB 管理结构，则将其更新到后续未使用的单元（第一个就校验通过不需要该操作）。例如，假设图 2.4 中第 3 个单元的校验不通过，则会向前查找第 2 个单元。如果第 2 个单元的校验通过，则该单元的数据被视为次新的 FDB 管理结构，并将其值更新为增加一个时序记录后的新值作为最新的FDB管理结构（会提前确认这个假定的时序记录正确存储了才更新FDB管理结构），然后将最新的 FDB 管理结构写入第 4 个单元。

**注意：**理论上最多只会往前查找一个单元。

理论上第1个单元的FDB管理结构必定是完整写入的，具体细节请查看下面第2点。

**2、从“备份恢复“区恢复**

如之前所述，当 FDB 管理结构区域使用到第 227 个单元时，下一次更新 FDB 管理结构时需要先执行擦除操作，然后将新的 FDB 管理结构写入第 1 个单元。

**备份逻辑：** 在执行擦除操作之前，系统会先将最新的 FDB 管理结构写入“备份恢复”区（标记为准备更新），然后才开始擦除原始区并写入新的 FDB 管理结构，最后将“备份恢复”区的数据标记为更新完成。

**前提条件：** 只有在新的时序记录已经写入 Flash 存储单元后，FDB 管理结构才会更新。

**下面是几种可能的情况验证：**

1. 开始备份时掉电： 会丢失一个最新的时序记录。
2. 擦除原始区时掉电： 不会丢失最新的时序记录，因为已经备份。上电后会将备份的数据恢复到原始区。
3. 写入原始区时掉电： 不会丢失数据点，因为已经备份。上电后会将备份恢复到原始区。
4. 无效化备份时掉电： 不会丢失数据点，因为两个区域的 FDB 管理结构相同。上电后将备份无效化，无需恢复到原始区。

**备份恢复区**  
 FDB 管理结构的备份格式如图 2.5 所示，大小为 20 字节。该区域的擦除频率与“FDB 管理结构”区域的擦除频率基本一致（此外，该区域还参与了其他容错机制）。

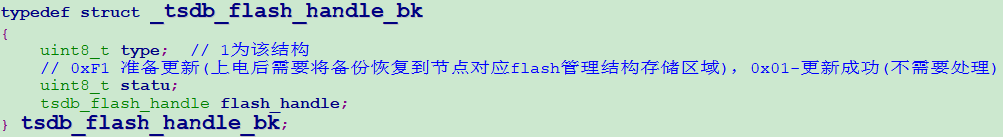


图2.5

数据库初始化（上电后执行一次），如果type字段的值为1：先查看status字段的值是否为0xF1，是则将备份恢复到“FDB管理结构”区域的第1个单元，然后将status字段的值写为0x01（无效化）。

##### **1.3时序记录**

###### **1.3.1存储结构**

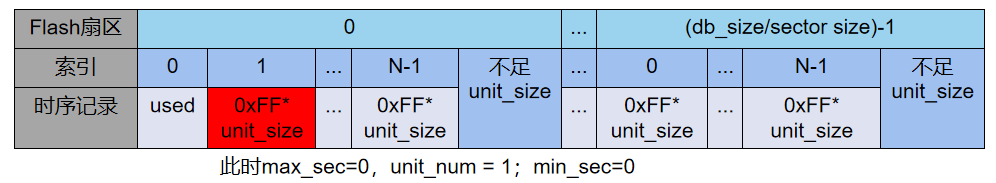
**时序记录的结构如下表**

|  |  |  |
| --- | --- | --- |
| **字段** | **大小** | **说明** |
| 时间戳 | 4字节 | 该值由用户存储数据时指定，如果该值不大于数据库中最新的时序记录的时间戳则该时序记录不能添加到数据库中 |
| CRC16 | 2字节 | 用户数据的校验 |
| 用户数据 | unit\_size-6 | 用户数据可能会小于unit\_size-6, 如果是这种情况，不足的长度补0xFF计算CRC16 |

“时序记录存储”区域的一个扇区分成N个数据单元(N=sector大小/unit\_size)，一个数据单元存储一条时序记录。

###### **1.3.2自修复机制**

并非每次存储时序记录时都需要执行擦除操作，而是只有在使用新扇区时才会进行一次扇区擦除。之后，扇区内的第 1 至 N-1个数据单元可以直接写入，而无需再次擦除。然而，这种机制存在一个潜在问题：由于这些直接写入的位置可能并非全为 0xFF（即，不能将 0 写为 1，例如原值为 0x1F 的位置不能写为 0x3F），这可能导致数据无法正确存储。

图 2.6

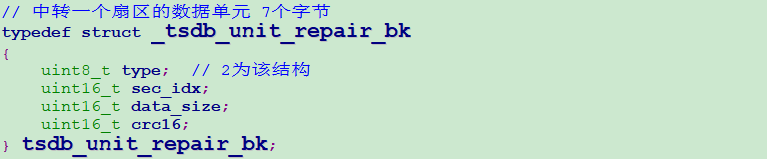
如图 2.6所示，已经存储了 1个时序记录，假设在存储第2 个时序记录的过程中发生了掉电（未完成写入或写入完成但未更新 FDB 管理结构）。上电后，新增的时序记录仍然会存储到第1个数据单元上。如果不处理这种情况，可能会导致第1个数据单元上的时间戳小于第0个数据单元上的时间戳，从而引发读取时序记录时出现问题；并且也会导致用户数据不能正确存储。

解决此问题并不复杂，只需将下一个写入位置修改为全 0xFF。

**具体步骤（上电读到FDB管理结构后执行一次）：**

1. 下个时序记录要存储的位置不是max\_sec扇区的第0个数据单元（因为使用第0单元前会执行擦除操作）并且max\_sec扇区后面未使用的空间非全0xFF”则执行：
2. 将该扇区已经存储的时序记录备份到“备份恢复”区域（保护）；
3. 将准备擦除哪个扇区的信息记录到“备份恢复”区域（保护）；
4. 开始擦除对应的扇区
5. 从“备份恢复”区域将时序记录拷贝回原来的扇区
6. 无效化第(2)步的自修复记录（将crc16字段的值写成0）

**第（1）和（2）步涉及到的结构**

图2.8

如图2.8为自修复记录的结构体，前面所述第（1）步是将时序记录数据拷贝到“备份恢复”区域所在扇区往后偏移sizeof(tsdb\_unit\_repair\_bk)个字节的位置，前面的空间用来存储自修复记录。

**type**：2为存储单元自修复；1为FDB管理结构备份

**sec\_idx**：自修复记录是当前数据库中哪个扇区的

**data\_size**：第（1）步备份的时序记录的大小;

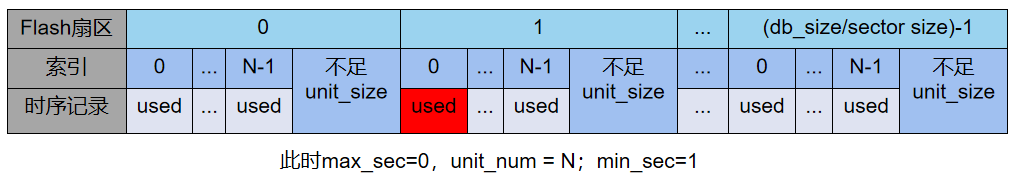
**注意：**data\_size+自修复记录大小必须大于等于扇区大小,data\_size的最大值为unit\_size\*(N-1)，为了确保自修复机制能正常运行，限定数据单元大小必须大于等于自修复记录大小，则用户数据的大小可以配置为[自修复记录大小-6， 扇区大小-6]。

**crc16**：前面所有字段的crc16校验，校验通过，才代表该自修复记录有效。

**上电后，检查“备份恢复”区域（保护机制）**

若 type 字段值为 2，则检查校验， CRC16 校验能通过，表示可能是在擦除 max\_sec 扇区后未将备份数据拷贝回去（也可能是尚未开始擦除，或者数据已经拷贝回去但未无效化自修复记录）。此时，执行第（3）、（4）和（5）步。

###### **1.3.3自修复机制2**



当 max\_sec 扇区的数据单元用完且数据库已完成一个循环（即 (max\_sec + 1) % 扇区总数 = min\_sec）时，写入新的时序记录之前需要先擦除 min\_sec 扇区。如果在执行‘擦除扇区 - 写入时序记录 - 更新 FDB 管理结构’的过程中没有完整执行完，可能会导致最旧的时序记录被清除。

在这种情况下，无需采用复杂的机制进行处理，只需更新 FDB 管理结构，将 min\_sec 向后偏移一个扇区，丢弃对应扇区的时序记录数据。

**具体步骤：**

1. 如果 max\_sec 扇区的数据单元已用完且 (max\_sec + 1) % 扇区总数 = min\_sec，则继续执行；
2. 如果 min\_sec 扇区的第一条时序记录（即最旧的时序记录）校验通过，则继续执行；否则，更新 FDB 管理结构并丢弃 min\_sec 扇区的时序记录；
3. 如果次旧的时序记录校验通过（如果存在次旧记录），且其时间戳大于最旧时序记录的时间戳，则表示不是前述情况，否则更新 FDB 管理结构并丢弃 min\_sec 扇区的时序记录。

**说明：** 需要执行第 3 步的原因是：如果下电之前执行完“擦除扇区 - 写入时序记录”的操作，则无法判断是属于前述的情况；需要比较时间戳的原因是：如果一个扇区只存储一条时序记录，则次旧的时序记录是校验通过的。

##### **1.4 API**

###### **1.4.1初始化数据库**

**bool fdb\_tsdb\_init(fdb\_tsdb\_t db, uint32\_t addr, uint32\_t size, uint16\_t data\_size);**

|  |  |
| --- | --- |
| 参数 | 描述 |
| db | 数据库句柄 |
| addr | 数据库的起始地址，必须扇区地址对齐 |
| size | 数据库的大小，扇区0用来存储管理结构，扇区1用于数据备份中转，扇区2之后才是用于存储时序记录 |
| data\_size | 用户数据的最大长度, 大小范围为[1, FDB\_SECTOR\_SIZE-6]，数据单元大小=用户数据的最大长度+6；一个数据单元存储一条时序记录，每个时序记录包括4个字节时间戳+2字节crc16+L字节用户数据，L<=data\_size |
| 返回 | 是否执行成功 |

**逻辑：**

1. 检查addr是否为扇区地址对齐，检查size是否为扇区大小的整数倍;
2. 检查“备份恢复”区域：FDB管理结构的备份和恢复机制、存储单元的自修复保护机制;
3. 从“FDB管理结构”区域读取FDB管理结构，如果校验通过并且数据库配置（size和data\_size）没有变化，则继承之前的数据库数据，否则重置FDB管理结构、重建数据库;
4. 检查max\_sex扇区的后续数据单元是否全为0xFF，不是则触发存储单元的自修复机制；
5. 执行存储单元的自修复机制2；
6. 如果有存储时序记录，将最新时序记录的时间戳读取出来，缓存在db->newest\_ts；
7. 如果有存储时序记录，将最旧时序记录的时间戳读取出来，缓存在db->oldest\_ts。

###### **1.4.2反初始化数据库**

**bool fdb\_tsdb\_deinit(fdb\_tsdb\_t db);**

|  |  |
| --- | --- |
| 参数 | 描述 |
| db | 数据库句柄 |
| 返回 | 是否执行成功 |

**逻辑：**

1. 检查数据库是否已经初始化；
2. 将FDB管理结构重置，丢弃之前存储的时序记录；
3. 将重置后的FDB管理结构存储到“FDB管理结构”区域的第1个单元。

###### **1.4.3增加时序记录**

**bool fdb\_tsl\_store(fdb\_tsdb\_t db, void \*pin, uint16\_t size, uint32\_t timestamp);**

|  |  |
| --- | --- |
| 参数 | 描述 |
| db | 数据库句柄 |
| pin | 需要存储的用户数据 |
| size | 用户数据的大小 |
| timestamp | 时序记录的时间戳 |
| 返回 | 是否执行成功 |

**逻辑：**

1. 检查数据库是否已经初始化；
2. 检查timestamp是否大于最新时序记录的时间戳；
3. 如果是添加第1条时序记录或者使用新的扇区，先执行擦除操作；
4. 如果是使用新的扇区并且会导致min\_sec扇区滚动，更新最旧时序记录的时间戳（db->oldest\_ts）；
5. 更新FDB管理结构
6. 将时序记录存储到FDB管理结构指向的数据单元
7. 将FDB管理结构同步到Flash

###### **1.4.4查找数据库的时序记录（指定时间段）**

uint32\_t fdb\_query\_tsl\_by\_time(fdb\_tsdb\_t db, uint32\_t from, uint32\_t to, fdb\_tsl\_cb cb, void \*cb\_arg);

|  |  |
| --- | --- |
| 参数 | 描述 |
| db | 数据库句柄 |
| from | 起始时间戳 |
| to | 结束时间戳 |
| cb | 回调函数，每次查询到时序记录时会执行该回调;该参数可为空，为空时只用来查询符合条件的时序记录数量 |
| cb\_arg | 回调函数的参数 |
| 返回 | 处于指定时间段内的时序记录数量 |

**逻辑：**

1. 检查数据库是否已经初始化、是否存储有至少一条时序记录；
2. 如果起始时间戳大于结束时间戳、起始时间戳大于最新时序记录的时间戳、结束时间戳小于最旧时序记录的时间戳，直接返回0；
3. 找到起始时间戳的时序记录位置，没找到则返回起始时间戳往后的第一个时序记录
   1. 起始时间戳小于最旧时序记录的时间戳，返回最旧时序记录的位置
   2. 起始时间戳大于最新时序记录的时间戳，返回空位置
   3. 使用二分法定位到起始时间戳的时序记录位置
4. 如果找到了起始时间戳的时序记录位置，则从该起始位置开始，往后遍历时序记录，直到时序记录的时间戳不在时间段范围、用户终止或者遍历到了最后一条时序记录
   1. 将当前位置的时间戳读取出来
   2. 如果时间戳在时间段范围内，则调用cb函数并传入时序记录的信息+cb\_arg参数
   3. 更新位置信息

###### **1.4.5读取时序记录的用户数据**

uint16\_t fdb\_read\_tsl(fdb\_tsl\_t tsl, void \*pout, uint16\_t outlen, uint8\_t mode)；

|  |  |
| --- | --- |
| 参数 | 描述 |
| tsl | 时序记录的信息 |
| pout | 用于返回读取到的用户数据 |
| outlen | pout指向的缓冲区大小 |
| mode | 模式，0：不校验crc，1：校验crc， |
| 返回 | 0：校验失败，其他值：用户数据的长度 |

**逻辑：**

1. 从tsl->data\_addr读取用户数据，读取outlen个字节；
2. 如果需要校验，先从tsl->crc16\_addr读取crc16，然后计算用户数据的校验，最后比较两者；
3. 如果outlen < 用户数据的最大长度，则用户数据的校验则需要继续读取 [outlen, 用户数据的最大长度-1] 的字节累计校验再比较。