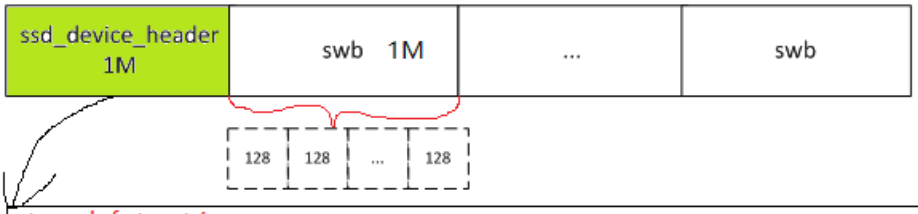
Aerospike磁盘数据存储格式

1、磁盘数据格式




```
typedef struct {
        uint64_t        magic;          // shows we've got the right stuff
        uint64_t        random;         // a random value - good for telling all disks are of the same state
        uint32_t        write_block_size;
        uint32_t        last_evict_void_time;
        uint16_t        version;
        uint16_t        devices_n;// number of devices
        uint32_t        header_length;
        char            namespace[32];// ascii representation of the namespace name, null-terminated
        uint32_t        info_n;         // number of info slices (should be > a reasonable partition count)
        uint32_t        info_stride;    // currently 128 bytes
        uint8_t         info_data[];
} __attribute__((__packed__)) ssd_device_header;
```

注：
记录向swb内存放是以128字节为单位，即记录大小必须以128的倍数存放，不够的后面补0

2、代码分析

```
1.
2. //磁盘头初始化函数
3. ssd_device_header *
4. ssd_init_header(as_namespace *ns)
5. {    //header的大小是1M
6.     ssd_device_header *h = cf_valloc(SSD_DEFAULT_HEADER_LENGTH);
7.
8.     if (! h) {
9.         return 0;
10.    }
11.
12.    memset(h, 0, SSD_DEFAULT_HEADER_LENGTH);
```

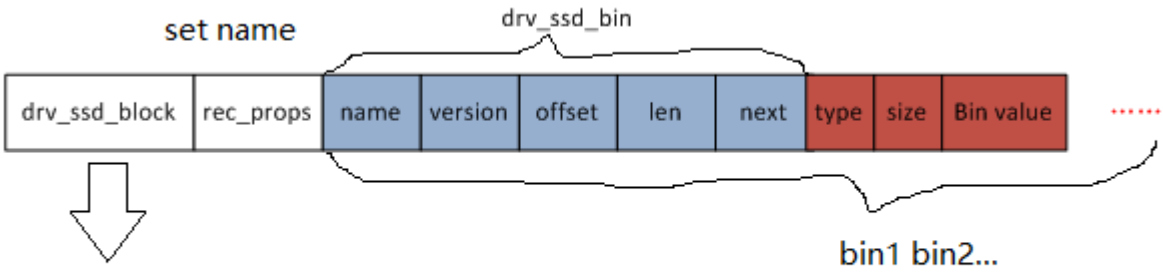
```
13.
14.     h->magic = SSD_HEADER_MAGIC;
15.     h->random = 0;
16.     h->write_block_size = ns->storage_write_block_size;
17.     h->last_evict_void_time = 0;
18.     h->version = SSD_VERSION;
19.     h->devices_n = 0;
20.     h->header_length = SSD_DEFAULT_HEADER_LENGTH;
21.     memset(h->namespace, 0, sizeof(h->namespace));
22.     strcpy(h->namespace, ns->name);
23.     h->info_n = AS_PARTITIONS;
24.     h->info_stride = SSD_HEADER_INFO_STRIDE;
25.
26.     return h;
27. }
```

3、SSD模式下，刷盘是随机的

```
1. //当current_swb写满时，从ssd->swb_free_q队列获取一个空闲的swb
2. ssd_write_bins->swb = swb_get(ssd)->cf_queue_pop(ssd->swb_free_q,
   &swb, CF_QUEUE_NOWAIT)
3. /*
4. 1、而ssd->swb_free_q链表里的swb并不是按磁盘从头到尾的顺序排列的
5. 2、后台线程从脏队列拿出一个刷完后放到swb_free_q队列里
6. */
7. ssd_write_worker->cf_queue_pop(ssd->swb_write_q, &swb, 100)->
8. ssd_flush_swb(ssd, swb)->ssd_post_write->swb_dereference_and_rele
   ase->
9. swb_release->cf_queue_push(swb->ssd->swb_free_q, &swb)
10.
11.
12. //swb和磁盘的关系是1M1M对应的
13. ssd_flush_swb->off_t write_offset = (off_t)WBLOCK_ID_TO_BYTES(ss
   d, swb->wblock_id);
14.             ->lseek(fd, write_offset, SEEK_SET)
15.             ->write(fd, swb->buf, ssd->write_block_size)
16. static inline uint64_t WBLOCK_ID_TO_BYTES(drv_ssd *ssd, uint32_t
    wblock_id) {
17.     return (uint64_t)wblock_id * (uint64_t)ssd->write_block_size;
18. }
```

swb不按照磁盘从小到大进行取，刷写时磁盘可能跳来跳去，即刷写时随机写。对于普通硬盘来说性能是不容乐观的。所以Aerospike官方对于SSD模式也推荐使用SSD盘进行存储数据。

4、写入swb内的记录格式




```
// Per-record metadata on device.
typedef struct drv_ssd_block_s {
  Uint64_t sig;       // deprecated
  Uint32_t magic;
  Uint32_t length;  // total after this field - this struct's pointer + 16
  Cf_digest keyd;
  Uint32_t generation;
  Cf_clock void_time;
  Uint32_t Bins_offset; // offset to bins from data
  Uint32_t n_bins;
  Uint64_t last_update_time;
  Uint8_t   data[];
} __attribute__ ((__packed__)) drv_ssd_block;
```


5、刷写磁盘和写入数据swb的关系

```
1.
2. void
3. ssd_flush_current_swb(drv_ssd *ssd, uint64_t *p_prev_n_writes,
4.         uint32_t *p_prev_size)
5. {
6.     uint64_t n_writes = cf_atomic64_get(ssd->n_wblock_writes);
7.     //ssd->n_wblock_writes表示ssd->swb_write_q队列中有多少个swb，该swb已写满脏数据
8.     // 如果swb_write_q队列中有脏数据的swb，则需要将该swb先刷写完成，这里不能先刷写current_swb
9.     if (n_writes != *p_prev_n_writes) {
10.         *p_prev_n_writes = n_writes;
11.         *p_prev_size = 0;
12.         return;
13.     }
14.     //因为current_swb刷写时，其他线程可能正在向里面写入，所以需要加锁；
15.     //swb_write_q中的swb不需要，因为已写满，其他的线程不能向里写了
16.     pthread_mutex_lock(&ssd->write_lock);
17.     n_writes = cf_atomic64_get(ssd->n_wblock_writes);
18.     // 还需要在锁里面再检查一次。因为可能正好有一个swb放到了swb_write_q里了
```

```
    if (n_writes != *p_prev_n_writes) {
        pthread_mutex_unlock(&ssd->write_lock);
        *p_prev_n_writes = n_writes;
        *p_prev_size = 0;
        return;
    }
    //如果current_swb不为空，那么可以刷。刷前需要将剩下的部分全部置成0清空
    ssd_write_buf *swb = ssd->current_swb;
    if (swb && swb->pos != *p_prev_size) {
        *p_prev_size = swb->pos;
        // Clean the end of the buffer before flushing.
        if (ssd->write_block_size != swb->pos) {
            memset(&swb->buf[swb->pos], 0, ssd->write_block_size - swb->pos);
        }
        //刷写到磁盘：swb->wblock_id*ssd->write_block_size为lseek的偏移，
        //写入大小是ssd->write_block_size
        ssd_flush_swb(ssd, swb);
    }
    pthread_mutex_unlock(&ssd->write_lock);
}
```

```
int
ssd_write_bins(as_storage_rd *rd)
{
    ...
    //在ssd->write_lock锁内进行操作，向里写入脏数据
    //如果current_swb为NULL，则从ssd->swb_free_q拿一个
    pthread_mutex_lock(&ssd->write_lock);
    ssd_write_buf *swb = ssd->current_swb;
    if (! swb) {
        swb = swb_get(ssd);
        ssd->current_swb = swb;
    }
    //如果current_swb空间不够了：（将剩下的清0），将swb放到swb_write_q队列
    //ssd->n_wblock_writes加一，表示队列里多了一个成员
    //从swb_free_q中拿一个当做current_swb
    if (write_size > ssd->write_block_size - swb->pos) {
        if (ssd->write_block_size != swb->pos) {
            // Clean the end of the buffer before pushing to write queue.
            memset(&swb->buf[swb->pos], 0, ssd->write_block_size - swb->pos);
        }
```

```
21.         cf_queue_push(ssd->swb_write_q, &swb);
22.         cf_atomic64_incr(&ssd->n_wblock_writes);
23.         swb = swb_get(ssd);
24.         ssd->current_swb = swb;
25.         ...
26.     }
27.     uint32_t swb_pos = swb->pos;
28.     swb->pos += write_size;
29.     cf_atomic32_incr(&swb->n_writers);
30.     pthread_mutex_unlock(&ssd->write_lock);
31.     ...
32.     //向swb写入
33. }
```

```
1. void *
2. run_ssd_maintenance(void *udata)
3. {
4.     ...
5.     uint64_t prev_n_writes_flush = 0;
6.     uint32_t prev_size_flush = 0;
7.     while (true) {
8.         ...
9.         ssd_flush_current_swb(ssd, &prev_n_writes_flush, &prev_si
   ze_flush);
10.         ...
11.     }
12. }
```

结论:
该函数是刷写current_swb的后台线程。prev_n_writes_flush的入参是0，结合上面2个函数的介绍如果ssd_write_bins函数在写入时，current_swb满了，将他放到swb_write_q队列，那么这里current_swb就不能刷写了，需要等待ssd_write_worker后台线程将swb_write_q队列的脏数据刷写后再刷。从而保证数据一致性。