元数据管理

1. 主要元数据类型

1） MAPPING

a. L2P:

逻辑地址到物理地址的映射，存在于每个 FTL LUN 中:

映射粒度为 4k（BOOT_BLK_POOL 中的 LUN 除外）

使用 L2PP_ID 描述一段 mapping, 长度为 1024 个 mapping，大小为 4k，4M 的 user_data 产生 middle_lun 的 4k 的数据写入。

b. P2L:

物理地址到逻辑地址的映射，目前只存在 USER_LUN 中。其作用为：

当有 UMT 存在时，P2L 只是用于加速重建和 GC；当没有 UMT 存在时，P2L 不仅要用于加速重建和 GC，还要作为 mapping 的 changeLog。

c. UMT:

L2p 的 changeLog，保存最新的 l2p 映射，只存在 USER_LUN 中。UMT 作为可选项，主要用于优化 IO 上啊对最新 mapping 的查询和写入，有额外的内存开销。

2） SPB_DESC_INFO

SPB 的描述信息，包含 SPB 的归属 pool, valid_cnt，Ec_cnt, flags 等状态信息。

3） BLOCK_DESC_INFO

ROOT_BLK_POOL 中描述 BLOCK 的信息，包含 ec_cnt 等信息。

4） LUN_DESC_INFO

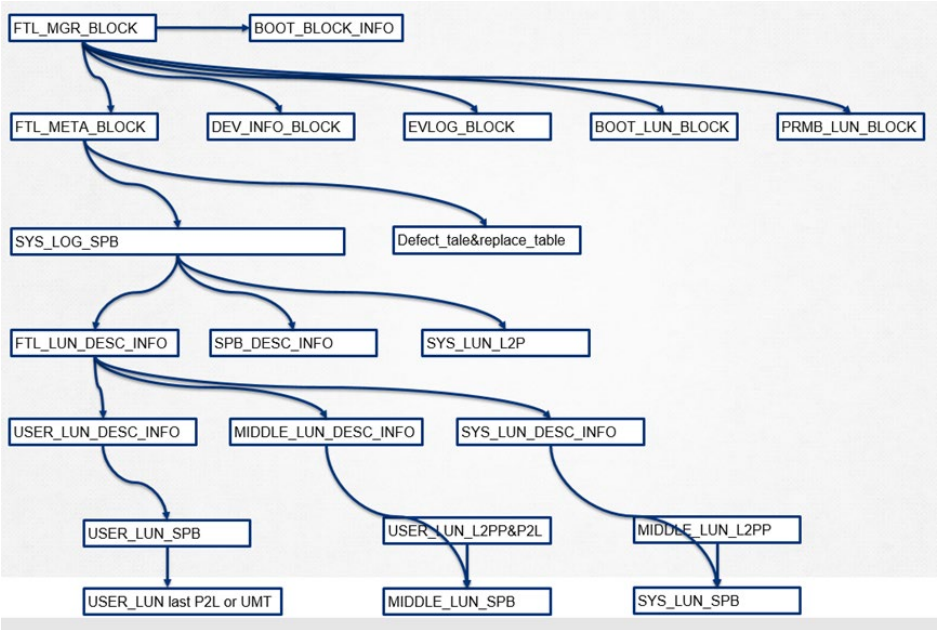
描述每个 LUN 的信息，包含 active SPB 的描述信息, sn, flags 等。

5） defect_bitmap & replace_table & 新增坏块

坏块表信息以及替换表信息；

新增坏块为发生了坏块的 SPB 的 BLOCK 信息。

2. 元数据布局



1） FTL_MGR_BLOCK:

该 BLOCK 为 FTL 所有 meta 的根，位于 ROOT_BLK_POOL 中。该 BLOCK 中的结构体定义：

```
Typedef struct _MGR_BLK_INFO

{

   MGR_BLK_HEADER   header;

   BLK_INFO                ftlMgrBlock;

   BLK_INFO               ftlMetaBlock;

    BLK_INFO                DevInfoBlock;

   BLK_INFO                EvLogBlock;

   BLK_INFO                WkLunBlock;

   U32                          BlockBitmap[];

BOOT_BLK_DESC_INFO   descInfo;
```

```
} MGR_BLK_INFO;

Typedef struct _MGR_BLK_HEADER

{

   U32                        signature;

   U32                          crc;

   U32                          flushId;

   U32                          dataLength;

} MGR_BLK_HEADER;

Typedef struct _BLK_INFO

{

   PDA_T                 writeBlock;

   PDA_T                 mirrorBlock[];

   PDA_T                 spareBlock[];

   U8                       mirrorCnt;

   U8                       spareCnt;

} BLK_INFO;
```

descInfo 中主要记录每个 BLOCK 的一些基本信息，比如 EC 次数等。

各 block 的镜像策略根据每个 BLOCK 自身的需求来确定镜像个数已经 spare 的大小。

当某个 BLOCK 的在读写时发生错误时，可以从 block_bitmap 中重新分配一个新的 block 来进行替换。

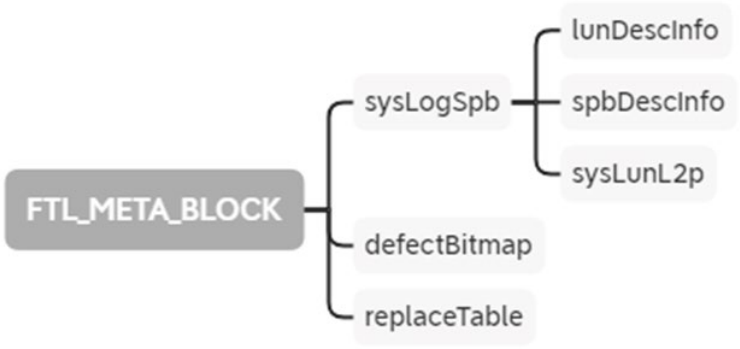
一个 writeBlockk 对应一个或多个 spareBlock。

针对 FTL_MGR_BLOCK, 采用 5 镜像 2 spare 策略，故占用 BLOCK 数：1+5+2 = 8。

2） FTL_META_BLOCK:

存放 FTL 模块根信息的 BLOCK。

该 BLOCK 中存放的信息为：



其中， sysLogSpb 中存放的是一些变化比较频繁的 FTL meta； defectBitmap 为 SPB 的坏块表；replaceTable 为 SPB 的坏块替换表。

针对 FTL_MGR_BLOCK, 采用 3 镜像 2 spare 策略，故占用 BLOCK 数：1+3+2 = 6。

3） DEV_INFO_BLOCK

主要用于存储 device 相关详细的 block:比如 smart 信息等。 采用 3 镜像 2 spare 策略，故占用 BLOCK 数：1+3+2 = 6。

4） WK_LUN_BLOCK

用于存储 well konow LUN 的 data 和描述符信息，以及 data 对应的 mapping 信息也保存到该 BLOCK 上。采用 3 镜像 2 spare 策略，故占用 BLOCK 数：1+3+2 = 6。（按容量）

5） EVLOG_BLOCK

存放 log 的 block，后续考虑可以放到其他的 BLOCK 上去。不采用镜像策略，分配 4 个 block 用于记录 evlog。

6） SYS_LOG

使用 SPB 用于存储 FTL 中的一些频繁变化的数据量相对比较大（与 ROOT_BLK_POOL 比较）的元数据，主要包含：

1. Lun_desc_info: sys_lun， middle_lun 和 user_lun 的描述信息，主要描述信息如下：

```
Typedef struct _LUN_DESC_INFO

{

   U32                  sn;

   U32                  flags;

   ASPB_DESC_T     aspbDesc[];

} LUN_DESC_INFO;

Typedef struct _ASPB_DESC_INFO

{

SpbId_t   spbId;

U32           writePtr;

U32           updatePtr;
} ASPB_DESC_INFO;
```

Sn 记录当前 LUN 的写 sn 号， 所有 LUN 的 sn 统一编码。

Flags 记录当前 LUN 是否是 clean 状态。

AspbDesc 描述当前 LUN 正在使用的 SPB 的信息，同时 ASPB 也是 LUN 重建的对象。其中 writePtr 描述写 page 分配的位置，updatePtr 描述 mapping 更新完成的位置。

2. Spb_desc_info:

描述 SPB 的状态信息：

```
Typedef struct _SPB_DESC_INFO
```

```
{
U32       flags:10;
   U32       validCnt:22;
   U32       readCnt:28;
   U32       subSpb:1;
   U32       spbType:1
   U32       poolId:2
   U32       ecCount;
   ……      ……
} SPB_DESC_INFO;
```

3. SYS_LUN 的 l2p：

即 sys_lun 的 mapping 表，该表常驻内存，需要全量存储到 nand。

7） USER_LUN mapping & MIDDLE_LUN mapping

USER_LUN 的 mapping 包含 P2L 和 L2P， 都作为 data 存储在 MIDDLE_LUN 中；而 MIDDLE_LUN 的 mapping 包含 L2P，作为 data 存储在 SYS_LUN 中。

4. USER_LUN P2L:

该表在没有 enable UMT 时，发挥两个作用：一是用于重建 ASPB 和 GC 时，查找数据是否有效；二是作为最新的 mapping changeLog，在 IO path 上提供最新的 mapping 查询。在 enable UMT 时，只有作用一。

P2L 的组织形式：

LDA0	LDA1	LDA2	LDA3	LDA4	LDA5	LDA6	….	….	LDAn

即在一段连续的 buffer 内记录 LDA 的信息， 而 pda 则是 buffer 数组的 index。Buffer 的长度为 16K，一个 ASPB 至少需要使用一个 16K（所有的

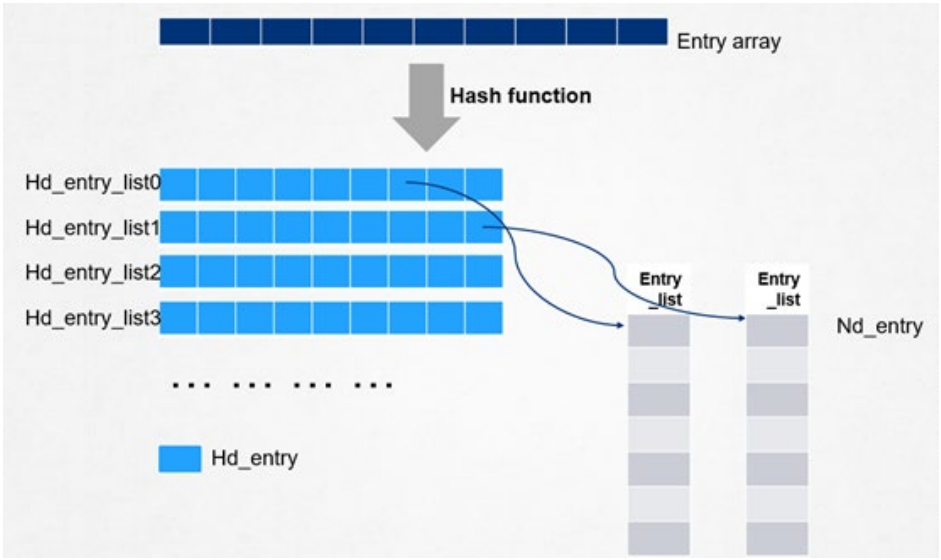
ASPB 额外需要一个或两个 16K 作为冗余）

b. UMT

UMT 为 L2P 的一部分，为当前内存中最新的 L2P。采用 hash 链表结构来进行管理：

Node 的结构为

链结构:



UMT 作为可选项，视当前系统的 memory 多少来开启或者关闭。若用 UMT，则 UMT 代替 P2L 充当 L2P 的 change log，提供 IO path 的增删查改。

c. L2P:

L2P 在内存中也需要进行管理（L2PP），组织为一段连续的 buffer:

PDA0	PDA1	PDA2	PDA3	PDA4	PDA5	PDA6	…	…	PDAn

,而在 LDA 上采用 l2pp_id + offset 的方式，其中 l2pp_id 为 L2P 在整个 LUN 的逻辑空间上的编码。例如：0~4M 为 l2pp 0， 4M~8M 为 l2pp 1。

3. 元数据操作

1） P2L:

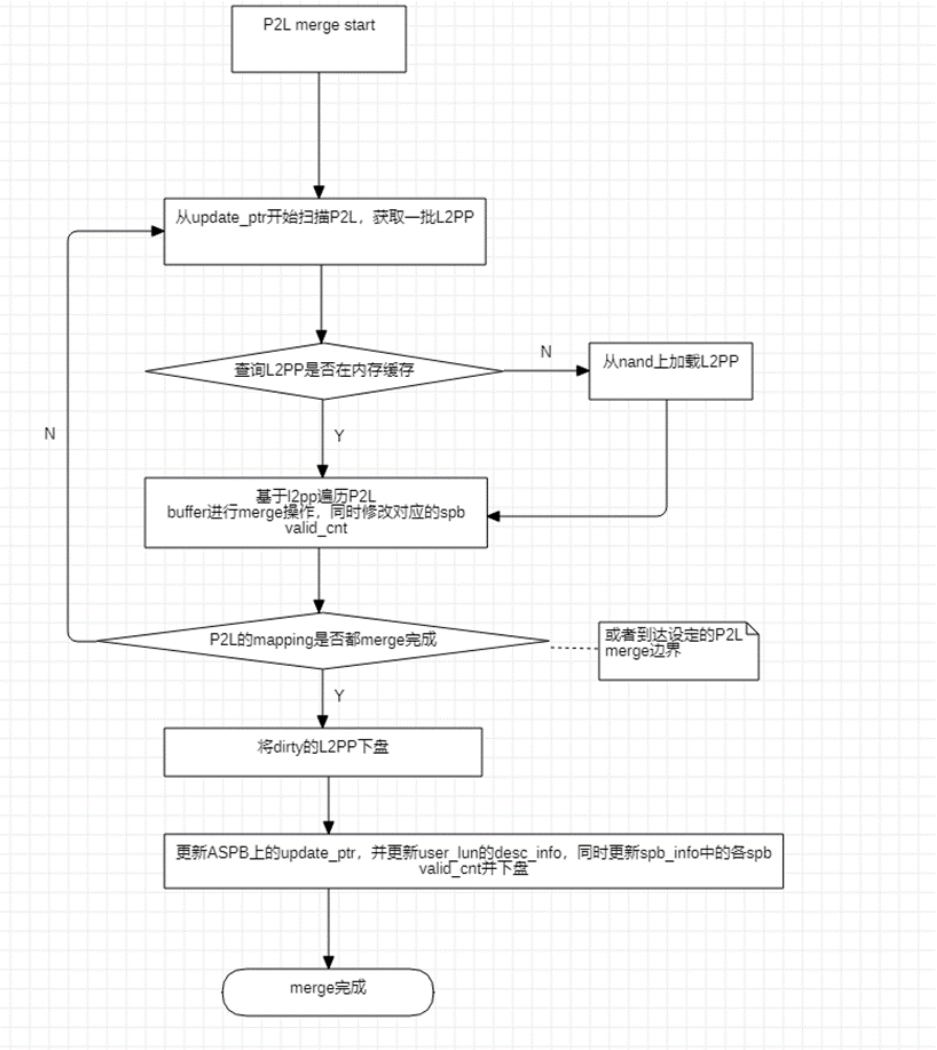
a. 写入：user write 以及 GC write 会将 mapping 写入到 P2L 中。host write 在写入时，需要 check user 的 P2L[A1] 中是否已经存在相同的 LDA。若有，需要将该 LDA 设置为无效 LDA；GC write 在 写入时，需要在 P2L 表上记录 GC 的源 spb_id 信息，用于后续的 merge 操作。

b. 下盘： 当 P2L buffer 写满后，会将 P2L flush 到 nand 保存。

c. Merge: 将 P2L 中的 mapping 更新到 L2P 中的行为

[A1]若搜索引擎支持倒序搜索，则可以采用最佳写入的方式，不需要 modify 之前的 LDA

具体操作流程为:



Merge 的时机：

当 P2L 表写满的时候，只触发当前的 P2L 表 merge；

有 trim update mapping 的时候，需要触发 host P2L 表 merge;

Flush 等流程触发的 force merge。

d. 释放：

当一个 P2L 在即完成了刷盘，又完成了 merge 后，才能将内存进行释放。

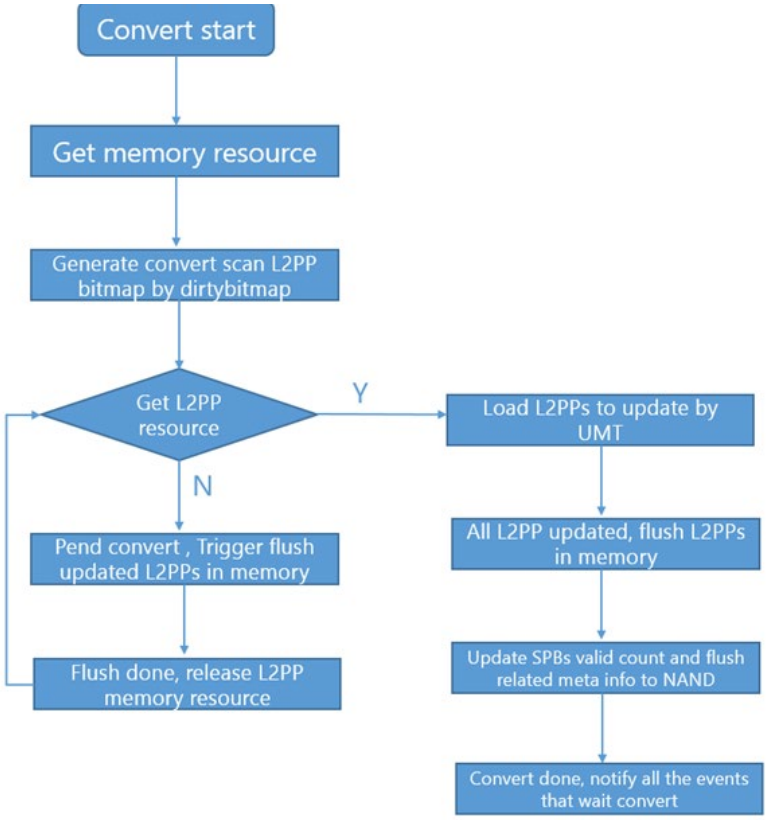
2） UMT

若存在 UMT 时，UMT 作为最新 mapping 的 change log：

a. 写入：

Host write 和 GC write 完成时，往 UMT 中写入最新 mapping。其中，GC write 写入前需要先 check UMT 中是否已经有相同的 LDA 存在，若有则该位置不写入。

b. Merge: 将 UMT 中的 mapping 更新到 L2P 中的操作。



UMT 不区分 mapping 是哪个 ASPB 的，都会一起 update。

Merge 的时机：

当 UMT 资源不足时；

有 trim update mapping 时;

Flush 等流程触发的 force merge。

c. 释放： 在 merge 进行的过程中，已经 merge 完成的那部分 entry 就可以进行释放。

3） LUN_DESC_INFO

a. 当 LUN 分配新的 ASPB 时，需要将新分配的 SPB 信息更新到 LUN_DESC_INFO 中并下盘。

b. 当 ASPB close 时，需要将 ASPB 的信息从 LUN_DESC_INFO 中移除并下盘。

c. 触发 mapping merge 时，需要更新 LUN_DESC_INFO 中的 ASPB 信息，并下盘。

d. 重建 LUN 时，需要将重建后的信息更新到 LUN_DESC_INFO，并下盘。

4) SPB_DESC_INFO

该描述性中包含多个属性字段，不同的流程会针对性的修改不同的属性字段，但不是所有的修改都需要马上下盘，需要视具体流程和修改的属性而定。

5) LUN 状态

Clean: 指与 LUN 相关的所有 meta 都完成下盘并处于一致状态。

比如 USER_LUN, 当其 P2L 全部完成 merge， 并下盘；L2PP 完成下盘；LUN_DESC_INFO 以及 SPB_DESC_INFO 也完成了更新并下盘后，该 LUN 就处于 clean 状态。此时若下电，再上电后就不需要重建任何数据。

Dirty: 指该 LUN 上还有一些 meta 存在脏数据，没有完成下盘。当上电后发现 LUN 是 dirty 的，就需要执行重建操作。

Clean->dirty: LUN 收到写 io；

Dirty->clean: LUN 执行 flush 流程，meta 完成刷盘。

6）SYS_LOG

SPB 空间来源于 SLC_SUB_SPB_POOL 中，动态申请。当 LUN_DESC_INFO|SPB_DESC_INFO|SYS_LUN L2P table 更新数据时，写入 SYS_LOG 的 SPB 当中。

SYS_LOG 根据写入的类型，管理一个 mapping_table。

当 SPB 快满时，触发分配一个新的 SPB；当 SPB 满时，触发切换 SPB 的动作，需要将旧的 SPB 上最新的数据 copy 到新的 spb 当中，然后释放旧的 SPB。同时，将新的 SPB 信息存入 FTL_META_BLOCK 当中。

SYS_LOG 的写都会触发一次镜像写以保证数据的可靠性。

7）Defect_bitmap & replace_table

a. 在开卡阶段，会生成初始扫描的 defect_bitmap 和初始替换的 replace_table， 将其存放在 FTL_META_BLOCK 当中；

b. 在正常运行阶段，读写 IO 失败导致产生新增坏块时，记录新增坏块信息到 FTL_META_BLOCK 上， 等待后台任务来进行 diagnose 和标记坏块。

8）ROOT_BLK_POOL

该 POOL 当中的 block 都预留有 spare block，当前 block 写满后，触发切换 block 的操作，将最新的数据 copy 到新的 block 当中去，然后修改为当前 BLOCK，之前的 BLOCK 修改为 spare block。

4. 元数据重建

在 FW 重建上电时，FTL 需要进行元数据重建操作：

1） 重建 FTL_MGR_BLOCK：

通过扫描指定位置的 ROOT_BLK_POOL 中的 block，找到最新的 FTL_MGR_BLOCK 信息所在的 BLOCK，恢复出其他 META 的 block root 信息。

2） 重建 WK_LUN_BLOCK:

为了尽快相应 host，该 BLOCK 的重建优先级仅次于 1）。通过扫描 WK_LUN_BLOCK 的 data,恢复出 WK_LUN 的 mapping_table 以支持 host 对齐的读写操作。

3） 重建 FTL_META_BLOCK:

通过扫描 FTL_META_BLOCK 所在的几个 block，找到最新的 page 恢复出 ftl_meta，包括 defect_bitmap, replace_table 和新增坏块信息。

4） 重建 SYS_LOG_SPB

通过 FTL_META_BLOCK 的信息可以获取到 sys_log_spb 的 spb_id；

如果有多个 spb 都存在有效数据，先找到最新数据的 spb;

扫描 spb 找到最新写入的 table 的位置；

基于 table 扫描 table 后续新写入的 page,恢复出最新的 table。

5） 重建 SYS_LUN

从 sys_log_spb 中获取到当前 SYS_LUN 的 LUN_DESC_INFO， 若 flag 标记为 clean 状态，则不需要重建；否则需要进行重建：

a. 扫描 SYS_LUN aspb，从 aspb 的 update_ptr 往后扫描，恢复出最新的 mapping，同时在 erase_page 后补充些一定数据量的 pad 数据，更新 aspb 的 wptr 信息。

b. 根据 mapping， 修正 SYS_LUN 中的 spb 的 valid_cnt 信息。

c. 将新的 l2p table 下盘，更新 update_ptr 后，将 LUN_DESC_INFO 下盘。

6） 重建 MIDDLE_LUN

从 sys_log_spb 中获取到当前 MIDDLE_LUN 的 LUN_DESC_INFO， 若 flag 标记为 clean 状态，则不需要重建；否则需要进行重建：

a. 扫描 MIDDLE_LUN aspb，从 aspb 的 update_ptr 往后扫描，恢复出最新的 mapping，同时在 erase_page 后补充些一定数据量的 pad 数据，更新 aspb 的 wptr 信息。

b. 将恢复出来的 mapping(l2p)信息下盘，更新 aspb update_ptr。

c. 将 LUN_DESC_INFO 下盘。

7）重建 USER_LUN

从 sys_log_spb 中获取到当前 MIDDLE_LUN 的 LUN_DESC_INFO， 若 flag 标记为 clean 状态，则不需要重建；否则需要进行重建：

a. 分别扫描 HOST 和 GC 的 aspb，从 update_ptr 往后扫描，获取对应位置的 P2L 表，通过扫描出的 mapping，build P2L 表。若得到的 P2L 表 ptr 小于 update_ptr，则需要从 P2L 表的 ptr 往后扫描来恢复 P2L 表。（存在 P2L 表 merge 但是未下盘的 场景）

b. 同时 aspb 在扫描到 erase_page 后，需要继续补充一定数据量的 pad 数据，更新 aspb 的 wptr 信息。

c. 触发 P2L merge，并进行特殊标记，在该标记下，对 spb_valid count 会有特殊处理。

d. Merge 完成后，更新 aspb update_ptr，将 LUN_DESC_INFO 下盘。