SPDK

Storage Performance Development Kit

读写流程

初始化

```
#ifdef MCAT_SPDK
    {
        cpu_set_t old_set;
        CPU_ZERO(&old_set);
        p_assert(pthread_getaffinity_np(pthread_self(), sizeof(old_set), &old_set) == 0);
        // init spdk
        struct spdk_env_opts opts;
        spdk_env_opts_init(&opts);
        opts.mem_size = get_sys_total_hugepage() * 2; // 2MB per page
        p_info("SPDK Memsize %d", opts.mem_size);
        opts.name = "flash-server";
        opts.core_mask = "ffffffffffffffff";
        p_err_assert(spdk_env_init(&opts) >= 0, "SPDK Init Failed %d", errno);
        p_info("SPDK Init DONE");
        p_assert(0 == spdk_nvme_probe(NULL, NULL, probe_cb, attach_cb, NULL));
        p_assert(pthread_setaffinity_np(pthread_self(), sizeof(old_set), &old_set) == 0);
    }
#endif
```

spdk_env_opts_init 初始化环境参数

spdk_env_init 初始化环境

spdk_nvme_probe 扫描、连接设备，probe_cb和attach_cb是两个回调函数

其中，probe_cb用于扫描设备时的回调，attach_cb用于连接设备时的回调

连接

```
void attach_cb(void *ctx, const struct spdk_nvme_transport_id *trid,
               struct spdk_nvme_ctrlr *ctrlr, const struct spdk_nvme_ctrlr_opts *opts)
{
    p_assert(ctrlr);
    int ns_cnt = spdk_nvme_ctrlr_get_num_ns(ctrlr);
    p_info("ns_cnt %d", ns_cnt);
    for (int i = 1; i <= ns_cnt; i++)
    {
        struct spdk_nvme_ns *ns = spdk_nvme_ctrlr_get_ns(ctrlr, i);
        if (!spdk_nvme_ns_is_active(ns))
            continue;
        printf("  Namespace ID: %d size: %juGB\n", spdk_nvme_ns_get_id(ns),
               spdk_nvme_ns_get_size(ns) / (1 << 30));
        g_ctx.ns_ctrlrs.push_back(ctrlr);
        g_ctx.ns_list.push_back(ns);
        g_ctx.ns_traddr.push_back(trid->traddr);
        ++g_ctx.ns_cnt;
    }
}
```

参数：

ctx：上下文

trid：传输标识符（Transport Identifier），其中含有已连接设备的地址

ctrlr：已连接的设备

opts：选项

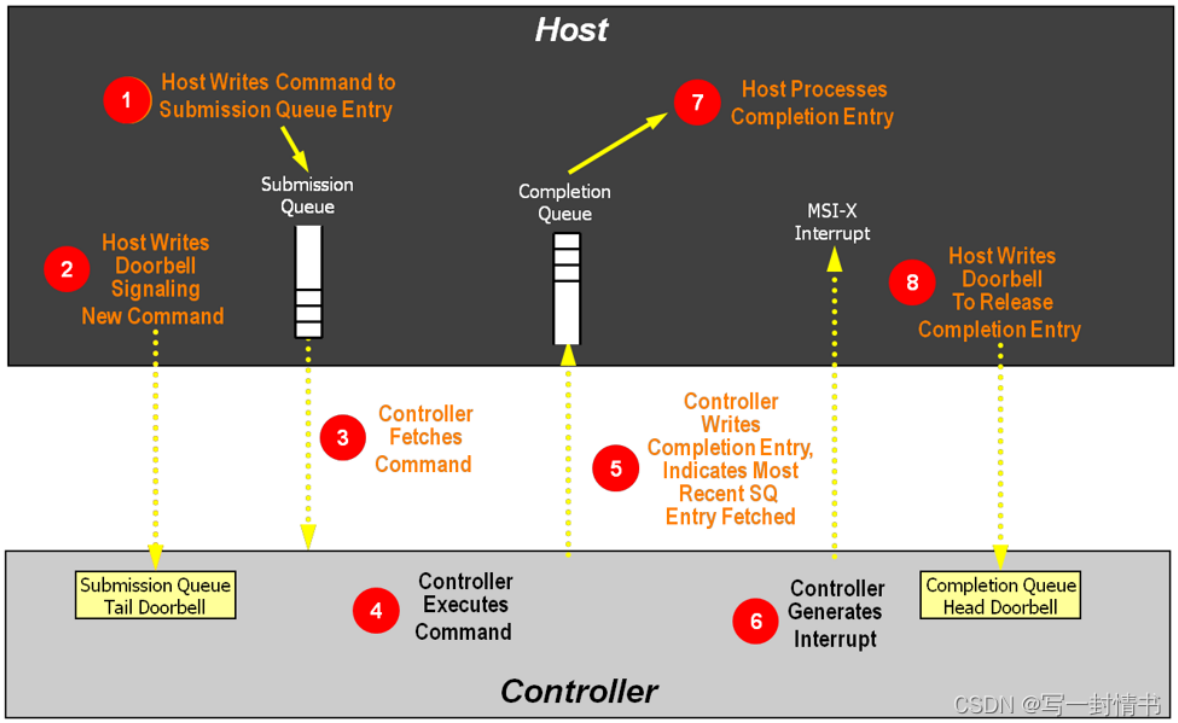
ns：namespace，NVMe上的逻辑单元，可能有一个或者多个，编号从1开始

I/O流程

Qpair：Submission Queue+Completion Queue

SQ和CQ都是环形队列

Qpair可以分类为负责控制命令传输的Admin Qpair和负责I/O命令传输的I/O Qpair



读写操作步骤：

1.使用spdk_nvme_ctrlr_alloc_io_qpair分配qpair

应用程序必须保证只有一个线程提交I/O到一个qpair，保证I/O操作无锁

(2.分配缓冲区)

```
/*
 * Write the data buffer to LBA 0 of this namespace.  "write_complete" and
 *  "&sequence" are specified as the completion callback function and
 *  argument respectively.  write_complete() will be called with the
 *  value of &sequence as a parameter when the write I/O is completed.
 *  This allows users to potentially specify different completion
 *  callback routines for each I/O, as well as pass a unique handle
 *  as an argument so the application knows which I/O has completed.
 *
 * Note that the SPDK NVMe driver will only check for completions
 *  when the application calls spdk_nvme_qpair_process_completions().
 *  It is the responsibility of the application to trigger the polling
 *  process.
 */
rc = spdk_nvme_ns_cmd_write(ns_entry->ns, ns_entry->qpair, sequence.buf,
                            0, /* LBA start */
                            1, /* number of LBAs */
                            write_complete, &sequence, 0);
if (rc != 0) {
    fprintf(stderr, "starting write I/O failed\n");
    exit(1);
}
```

3.spdk_nvme_ns_cmd_write将缓冲区数据写入一个内存空间的一个或多个LBA。write_complete是回调函数，传递I/O完成的具体信息。spdk_nvme_ns_cmd_read从内存空间的一个或多个LBA中读取数据存入缓冲区

4.spdk_nvme_qpair_process_completions处理轮询完成。必须调用这个函数来处理completions

(5.spdk_nvme_ctrlr_free_io_qpair释放qpair)