基于B-Tree和LSM存储引擎之引擎启动篇

作者	时间	QQ技术交流群
perrynzhou@gmail.com	2022/04/20	672152841

存储内核技术交流

微信扫描二维码，关注我的公众号

开源存储问题解答社区:https://github.com/perrynzhou/deep-dive-storage-in-china

- wt引擎初始化是通过wiredtiger_open函数进行，这个函数需要指定数据库目录，顺带初始化连接，创建wt引擎的系统schema和配置。接下来看下wiredtiger_open函数的都做了那些事情，如下是核心代码:

```
// home: 引擎存储的数据库目录
// event_handler:事件处理函数
// config:wt引擎实例的配置参数
// connectionp:数据库操作的实例的二级指针
int wiredtiger_open(const char *home, WT_EVENT_HANDLER *event_handler,
const char *config,WT_CONNECTION **connectionp)
{
        // 定义wt引擎的静态实例
        static const WT_CONNECTION stdc = {.....};

        // 每个引擎实例对应一个 __wt_process, 在该方法内初始化__wt_process
        __wt_library_init();
```

```
        // 初始化 WT_CONNECTION_IMPL 结构，是实例连接的实现方法，WT_CONNECTION指向的也是conn->iface
    __wt_calloc_one(NULL, &conn);
    conn->iface = stdc;
        __wt_connection_init(conn);


        //.解析系统配置模板，根据输入的参数来检查是不是合法
        __wt_config_check(session, WT_CONFIG_REF(session, wiredtiger_open), config, 0);


        // 设置wt引擎的数据目录，如果home为空，则根据环境变量WIREDTIGER_HOME来设定;
        _conn_home(session, home, cfg);

        // 默认配置512个哈希桶来存储的wt的扩展
        __conn_hash_config(session, cfg);
        // 加载应用端加载的扩展
        __conn_load_extensions(session, cfg, true);

        // 设置wt引擎实例的文件系统操作的posix的方法
        __wt_os_posix(session);
        // 检查Wt实例中的posix方法是否设置
        __conn_chk_file_system(session, F_ISSET(conn, WT_CONN_READONLY));

        // wt引擎实例初始化，涉及的步骤相对较多
        __conn_single(session, cfg){
                // 检查wt数据目录下的 WiredTiger 版本号文件是否存在
                __wt_fs_exist(session, WT_WIREDTIGER, &exist);

                // 打开WiredTiger.lock 锁文件
                __wt_open(session, WT_SINGLETHREAD, WT_FS_OPEN_FILE_TYPE_REGULAR,
      is_create || exist ? WT_FS_OPEN_CREATE : 0, &conn->lock_fh);

                // WiredTiger.lock锁文件中写入"WiredTiger lock file"
                __wt_write(session, conn->lock_fh, (wt_off_t)0,strlen(WT_SINGLETHREAD_STRING),WT_SINGLETHREAD_STRING);

                // 创建 WiredTiger 版本号文件
                __wt_open(
      session, WT_WIREDTIGER, WT_FS_OPEN_FILE_TYPE_REGULAR, is_create ? WT_FS_OPEN_CREATE : 0, &fh);

                // 检查系统配置 WiredTiger.turtle、WiredTiger.turtle.set 是否存在，如果存设置WiredTiger.turtle 元数据系统表
                __wt_turtle_exists(WT_SESSION_IMPL *session, bool *existp);

                // 初始化版本号
                __wt_snprintf_len_set(buf, sizeof(buf), &len, "%s\n%s\n",
```

```
WT_WIREDTIGER, WIREDTIGER_VERSION_STRING));
        __wt_write(session, fh, (wt_off_t)0, len, buf);

        }
        // 代码内置的基本配置写入到WiredTiger.basecfg
        __conn_config_file(session, WT_BASECONFIG, false, cfg, i1);

        // 用户输入的配置结合系统的基础配置吸入到文件 WiredTiger.config
        __conn_config_file(session, WT_USERCONFIG, true, cfg, i2);

        /****读取各种配置设置conn掩码 ****/

        // lsm引擎的配置，需要读取lsm_manager.merge和
lsm_manager.worker_thread_max
        __wt_lsm_manager_config(session, cfg);

        // 打开引擎实例
        __wt_connection_open(conn, cfg) {
                // 打开默认的一个Session
                __wt_open_internal_session(conn, "connection", false, 0,
&session);

                // cache初始化
                __wt_cache_create(session, cfg);

                // 根据配置决定是否要初始化事务系统
                __wt_txn_global_init(session, cfg);
        }
        // 用户扩展的加载和初始化
        __conn_builtin_extensions(conn, cfg));
        __conn_load_extensions(session, cfg, false));

        // 日志管理的初始化
        __wt_logmgr_config(session, cfg, false);
        // 写入本实例的基本配置
        __conn_write_base_config(session, cfg);

        // 完成WT_SESSION_IMPL 初始化和更新WiredTiger.wt元数据表的文件
        __wt_metadata_init_base_write_gen(session);
        __wt_metadata_cursor(session, NULL);

        // 设置同步的session
        __wt_backup_open(session);

        // 启动wt引擎的内部线程
        __wt_connection_workers(session, cfg);

}
```

- `wt`内部线程类型在实现代码中已经定义好了，具体如下:

```
#define WT_CONN_SERVER_CAPACITY 0x01u
#define WT_CONN_SERVER_CHECKPOINT 0x02u
#define WT_CONN_SERVER_LOG 0x04u
#define WT_CONN_SERVER_LSM 0x08u
#define WT_CONN_SERVER_STATISTICS 0x10u
#define WT_CONN_SERVER_SWEEP 0x20u
#define WT_CONN_SERVER_TIERED 0x40u
```

- `wt`引擎初始化后会启动一些内部的线程，比如cache线程，checkpoint线程等。启动线程的逻辑在`__wt_connection_workers`函数中，如下是说明该函数的核心逻辑。

```
int __wt_connection_workers(WT_SESSION_IMPL *session, const char *cfg[])
{
        // 启动statistics 线程，收集指标,线程的核心函数是__statlog_server
    __wt_statlog_create(session, cfg));
        // 启动后端存储的线程，这个核心调用的是__tiered_server
    __wt_tiered_storage_create(session, cfg, false);

        // 启动日志子系统初始化
    __wt_logmgr_create(session);

        //启动事务恢复
    __wt_txn_recover(session, cfg);


        // schema元数据追踪的初始化
    __wt_meta_track_init(session);

   // 初始化wt引擎历史存储,主要读取WiredTigerHS.wt
    __wt_hs_open(session, cfg);

         // 启动日志子系统的线程,核心函数 __log_server
    __wt_logmgr_open(session);

        // 启动page的淘汰线程
    __wt_evict_create(session);
        // 启动sweep 线程
    __wt_sweep_create(session);

        // schema对应的文件数据同步线程,核心函数__capacity_server
    __wt_capacity_server_create(session, cfg);

        // 启动检查点线程,核心函数__ckpt_server
    __wt_checkpoint_server_create(session, cfg);

    return (0;
}
```