由于 sqlite 对多进程操作支持效果不太理想，在项目中，为了避免频繁读写 文件数据库带来的性能损耗，我们可以采用操作 sqlite 内存数据库，并将内存数据库定时同步到文件数据库中的方法。

实现思路如下：

1、创建文件数据库；

2、创建内存数据库（文件数据库、内存数据库的内幕表结构需要一致）；

3、在内存数据库中 attach 文件数据库，这样可以保证文件数据库中的内容在内存数据库中可见；

4、对于 insert、select 操作，在内存数据库中完成，对于 delete、update 操作，需要同时访问内存、文件数据库；

5、定时将内存数据库中的内容 flush 到文件数据库。

通过 sqlite 的 cAPI 实现代码如下：

[cpp] view plain copy

```cpp
const char* file_database_path = "/home/tom/test/database/filedb"; //文件数据库存放路径

const char* sql_create_data = "CREATE TABLE testinfo (id TEXT PRIMARY KEY, message TEXT, offset INTEGER, timestamp INTEGER);";
const char* sql_insert_data = "INSERT OR REPLACE INTO MAIN.testinfo VALUES('%s', '%s', %d, %d);";
const char* sql_delete_data = "DELETE FROM MAIN.testinfo WHERE id = '%s'; DELETE FROM filedb.testinfo WHERE id = '%s';"; //删除数据库，需同时删除内存、文件数据库中的内容
const char* sql_update_data = "UPDATE MAIN.testinfo SET message = '%s', offset = %d, timestamp = %d where id = '%s'; UPDATE filedb.testinfo SET message = '%s', offset = %d, timestamp = %d where id = '%s';";//更新数据库，需同时更新内存、文件数据库中的内容
```

```
const char* sql_search_data = "SELECT * FROM MAIN.testinfo WHERE timestamp BETWEEN %d AND %d union SELECT * FROM testdb.testinfo WHERE timestamp BETWEEN %d AND %d;"; //查找数据库，将内存、文件数据库中查找出的内容合并
const char* sql_transfer_data = "INSERT OR REPLACE INTO filedb.testinfo SELECT * FROM testinfo;";   //将内存数据库中的信息同步到文件数据库中
const char* sql_delete_memory_table = "DELETE FROM testinfo;";  //内存数据库中的内容同步结束后，清空


int InsertRecord(DATA_TYPE type, const char* id, const char* message, int offset, int timestamp)
{
    int      rc              =  0;
    char*    errMsg          =  NULL;
    char     sqlcmd[512]     =  {0};
    time_t   insertTimestamp =  0;

    snprintf(sqlcmd, sizeof(sqlcmd), sql_insert_data, id, message, offset, timestamp);
    rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't add record to memory database %s, sqlcmd=%s, err:%s\n", map_data_table[type].data_table_name, sqlcmd, errMsg);
        return -1;
    }

    return 0;
}

int UpdateRecord(DATA_TYPE type, const char* id, const char* message, int offset, int timestamp)
{
    int      rc              = 0;
    char*    errMsg          = NULL;
    char     sqlCmd[512]  = {0};

    snprintf(sqlCmd, sizeof(sqlCmd), sql_update_data, message, offset, timestamp, id, message, offset, timestamp, id);
    rc = sqlite3_exec(memdb, sqlCmd, NULL, NULL, &errMsg);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't update record %s:%s\n", map_data_table[type].data_table_name, errMsg);
        return -1;
    }
```

```
41.
42.     return 0;
43. }
44.
45. int DeleteRecord(DATA_TYPE type, const char* id)
46. {
47.     int      rc              =  0;
48.     char*    errMsg          =  NULL;
49.     char     sqlcmd[512]     =  {0};
50.
51.     snprintf(sqlcmd, sizeof(sqlcmd), sql_delete_data, id,  id);
52.     rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
53.     if (SQLITE_OK != rc) {
54.         fprintf(stderr, "cat't delete record %s:%s\n", map_data_table[type].
    data_table_name, errMsg);
55.         return -1;
56.     }
57.
58.     return 0;
59. }
60.
61. int QueryMessage(DATA_TYPE type, int startTime, int endTime)
62. {
63.     int      rc              = 0;
64.     char     *errMsg         = NULL;
65.     sqlite3  *filedb         = NULL;
66.     char**   pRecord         = NULL;
67.     int      row             = 0;
68.     int      column          = 0;
69.     char     sqlcmd[512]     = {0};
70.
71.     if (type > VEP_NELEMS(map_data_table) || type < 0) {
72.         return -1;
73.     }
74.
75.     rc = sqlite3_open(file_database_path, &filedb);
76.     if (SQLITE_OK != rc) {
77.         fprintf(stderr, "cat't open database:%s\n", sqlite3_errmsg(filedb));

78.         sqlite3_close(filedb);
79.         return -1;
80.     }
81.
```

```
    snprintf(sqlcmd, sizeof(sqlcmd), sql_search_data,  startTime, endTime, startTime, endTime);

    rc = sqlite3_get_table(filedb, sqlcmd, &pRecord, &row, &column, &errMsg);

    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't get table from%s:%s\n", map_data_table[type].data_table_name, errMsg);
        return -1;
    }

    int i;
    printf("row = %d, column = %d\n", row, column);
    for(i = 0; i < 2*column; i++)
    {
        printf("%s ", pRecord[i]);
    }
    printf("\n");

    return 0;
}

//定时调用此函数将内存数据中的内容同步到文件数据库
int Flush(){
    int      i            = 0;
    int      rc           = 0;
    char*    errMsg       = NULL;
    char     sqlcmd[512]  = {0};

    snprintf(sqlcmd, sizeof(sqlcmd), sql_transfer_data);
    rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't transfer memory database %s to file database de:%s\n", map_data_table[i].data_table_name, sqlite3_errmsg(memdb));
        sqlite3_close(memdb);
        return -1;
    }
    snprintf(sqlcmd, sizeof(sqlcmd), sql_delete_memory_table);
    rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);

    return 0;
}

//创建文件数据库
```

```
int CreateDbOnFile()
{
    sqlite3 *db           = NULL;
    int      rc           = 0;
    char*    errMsg       = NULL;
    char     sqlcmd[512]  = {0};
    int      i            = 0;

    rc = sqlite3_open(file_database_path, &db);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't open database:%s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return -1;
    }

    snprintf(sqlcmd, sizeof(sqlcmd), sql_create_data);
    rc = sqlite3_exec(db, sqlcmd, NULL, NULL, &errMsg);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't create file database testinfo:%s\n", errMsg);

        sqlite3_close(db);
        return -1;
    }

    sqlite3_close(db);
    return 0;
}

//创建内存数据库
int CreateDbOnMemery()
{
    int      rc           = 0;
    char*    errMsg       = NULL;
    char     sqlcmd[512]  = {0};
    int      i            = 0;

    rc = sqlite3_open(":memory:", &memdb);
    if (SQLITE_OK != rc) {
        fprintf(stderr, "cat't open database:%s\n", sqlite3_errmsg(memdb));

        sqlite3_close(memdb);
        return -1;
    }

```

```
164.     snprintf(sqlcmd, sizeof(sqlcmd), sql_create_data);
165.     rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
166.     if (SQLITE_OK != rc) {
167.         fprintf(stderr, "cat't create memory database %s\n", errMsg);
168.         sqlite3_close(memdb);
169.         return -1;
170.     }
171. 
172.     return 0;
173. }
174. 
175. //解绑数据库
176. int DetachDb()
177. {
178.     int      rc           =  0;
179.     char*    errMsg       =  NULL;
180.     char     sqlcmd[512]  =  {0};
181. 
182.     snprintf(sqlcmd, sizeof(sqlcmd), "DETACH '%s'", "filedb");
183.     rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
184.     if (SQLITE_OK != rc) {
185.         fprintf(stderr, "detach file database failed:%s:%s\n", file_databas
    e_path, errMsg);
186.         sqlite3_close(memdb);
187.         return -1;
188.     }
189. 
190.     return 0;
191. }
192. 
193. //将文件数据库作为内存数据库的附加数据库
194. int AttachDb()
195. {
196.     int      rc           =  0;
197.     char*    errMsg       =  NULL;
198.     char     sqlcmd[512]  =  {0};
199. 
200.     snprintf(sqlcmd, sizeof(sqlcmd), "ATTACH '%s' AS %s", file_database_pat
    h, "filedb");
201.     rc = sqlite3_exec(memdb, sqlcmd, NULL, NULL, &errMsg);
202.     if (SQLITE_OK != rc) {
203.         fprintf(stderr, "cat't attach database %s:%s\n", file_database_path,
     errMsg);
204.         sqlite3_close(memdb);
```

```
205.         return -1;
206.     }
207. 
208.     return 0;
209. }
210. 
211. //初始化数据库，分别创建文件数据库、内存数据库并把文件数据库 attach 到内存数据库上
212. int InitSqliteDb()
213. {
214.     int retval = 0;
215. 
216.     retval =  CreateDbOnFile();
217.     if (retval != 0) {
218.         return retval;
219.     }
220. 
221.     retval =  CreateDbOnMemery();
222.     if (retval != 0) {
223.         return retval;
224.     }
225. 
226.     retval =  AttachDb();
227.     if (retval != 0) {
228.         return retval;
229.     }
230. 
231.     return 0;
232. }
```