C API to HDFS: libhdfs

C API to HDFS: libhdfs

Libhdfs 是基于 JNI 访问 HDFS 的 C api 。它包括 HDFS 和 MAPRED API 的子集，和官网发布的有所不同：

1. 增加登陆 hdfs 的密码验证功能

2. 增加对作业任务的查询接口

它是预编译的并作为 sos 小组发布 hadoop-v2 一同发布，只适用与 hadoop-v2 和后续版本，hadoop-v1 不支持。

The APIs

Libhdfs 提供的 API 可以在源码头文件里找到，

${HADOOP_HOME}/src/c++/libhdfs/hdfs.h

对外发布则在 ${HADOOP_HOME}/libhdfs/hdfs.h

示例代码（详细可参看 hdfs_test.c，mapred_test.c，hdfs_read.c 和 hdfs_write.c 实例代码）：

```
#include "hdfs.h"

int main(int argc, char **argv) {

    hdfsFS fs = hdfsConnect("default", 0);
    const char* writePath = "/tmp/testfile.txt";
    hdfsFile writeFile = hdfsOpenFile(fs, writePath, O_WRONLY|O_CREAT, 0, 0, 0);
    if(!writeFile) {
          fprintf(stderr, "Failed to open %s for writing!\n", writePath);
          exit(-1);
    }
    char* buffer = "Hello, World!";
    tSize num_written_bytes = hdfsWrite(fs, writeFile, (void*)buffer, strlen(buffer)+1);
    if (hdfsFlush(fs, writeFile)) {
           fprintf(stderr, "Failed to 'flush' %s\n", writePath);
          exit(-1);
    }
   hdfsCloseFile(fs, writeFile);
}
```

怎样链接这个库文件

请参考源码中 hdfs_test.c 的 Makefile 文件

(${HADOOP_HOME}/src/c++/libhdfs/Makefile) or something like: gcc above_sample.c -I${HADOOP_HOME}/src/c++/libhdfs -L${HADOOP_HOME}/libhdfs -lhdfs -o above_sample

常见问题

编译时需正确设置 JAVA_HOME，LIBHDFS_BUILD_DIR 和 LD_LIBRARY_PATH（可参看 Makefile 示例的设置）。

并确认编译机是 64 位系统，若为 32 位系统请将 Makefile 中 CPPFLAGS 的-m64 改为 -m32

运行是需设置 CLASSPATH 和 LD_PRELOAD。CLASSPATH 要指向 hadoop 的 jar，lib 和 conf 路径（可参考 build 里的 test.sh 脚本）

libhdfs is thread safe

libhdfs 是线程安全的，包括后来实现的作业信息查询 API 都是如此。它通过 jni 调用 HDFS 提供的 API，而 HDFS 本身的 API 也是线程安全。

API 介绍：

主要介绍与官方不同的 API，其它 API 可直接参考官方的说明：

http://hadoop.apache.org/core/docs/r0.19.0/api/org/apache/hadoop/fs/FileSystem.html

- **修改过的 API：**

hdfsFS hdfsConnectAsUser(const char* host, tPort port,

const char *user , const char *password);

官方定义为：

hdfsFS hdfsConnectAsUser(const char* host, tPort port,

const char *user , const char **groups, int groups_size);

相对官方 sos 发布的版本有密码验证机制，并去掉了组信息参数。

- **新增的 API：**

mapredJC JobClientInitialize();

新建一个 jobclient，按照 CLASSPATH 设置的 conf 读取配置信息

返回 mapredJC 结构

int JobClientFinalize(mapredJC jc);

析构 mapredJC 结构的

成功返回 0，失败返回-1

MRJobInfo *mapredGetAllJobs(mapredJC jc, int *numItem);

得到当前所有任务的信息，传入 mapredJC 参数，和 numItem 地址

若成功返回 MRJobInfo 数组地址，数组个数由 numItem 传回；

若失败*numItem 置为-1，返回 NULL

若任务为 0，则*numItem 置为 0 放回 NULL

MRJobInfo* mapredGetJob(mapredJC jJobClient, const char *jobid)

得到指定任务的信息，传入 mapredJC 参数和 jobid 字符串

若成功则返回对应此 jobid 的 MRJobInfo 结构地址，注意此结构在堆中分配需要及时析构。

若失败返回 NULL

附相关数据结构说明：

```
    typedef struct   {
        long startTime; /* maybe use time_t */
        int runState; /*RUNNING = 1 SUCCEEDED = 2 FAILED = 3 PREP = 4 KILLED = 5*/
        char jobID[30];     /* the name of the a map-reduce job*/
        char user[30];
    } MRJobInfo;
```

StartTime 是作业启动的时间，单位为秒

RunState 是作业的当前状态

JobID 是作业的字符串表示

User 是启动作业的用户