光伏2.0公共配置文件和软件私有配置文件读取接口说明

此接口只包括如下的两个配置文件:

公共配置文件为GetPvPubFile(PV_PUBCFG_FILE)返回的ini配置文件
私有配置文件为GetPvRunPar(RUNPAR_CFGFILE)返回的ini配置文件

此接口用到了如下两个开源代码:

minIni 一个可移植迷你ini文件解析器（大约900行源码)
jsoncpp json工具库

配合文件监视器接口可实现不用重启程序,配置文件的值动态实时生效

使用方法举例

1.项目代码新引入源文件或库文件说明

- 引入Json的静态库文件libjson_linux-gcc-4.8.5_libmt.a(此库文件是通过开源代码jsoncpp-src-0.5.0编译而成,不同的编译环境名称略有不同)

- 将Json的头文件夹 json 引入到项目（注意此处并不把json文件夹下的文件直接引入项目，因此程序中在包括头文件是加了json文件夹的路径的）
- 引入minIni的源码文件 minIni.h、minGlue.h、minIni.c
- 引入PvDir的源码文件 PvDir.h、PvDir.cpp 或 库文件libPvDir.a和PvDir.h
- 引入PvCfg的源码文件 PvCfg.h、PvCfg.cpp

2.在项目中调用接口的顺序及接口使用说明

(1) 加载配置文件

```
/*************************************************************
  * brief  : 程序开始时调用此接口加载公共配置和软件私有配置文件
  * note   : 在主程序开始时就调用
*************************************************************/
int PvCfgInitialize();
```

(2) 在需要配置值处调用接口取配置值

- 取公共配置的值

```
/*************************************************************************
  * brief  : 获取公共配置"/xxx/pvfs20/id1/pubfile/pub_config.init"的值
  *    par :
  *          val      从配置文件取得的配置值
  *          section  需要取值的域名
```

```
  *          key      需要取值的key名
  *    ret :
  *          0  成功
  *         !0  失败
  * note  :
  *          如果配置文件中存在没有域名的配置值,section也要输入,此时section
  *          的值为"ROOT"
  *
*****************************************************************************/
int GetPvPubCfgVal( std::string &val, const char* section, const char* key );
```

- 取私有配置的值

```
/*****************************************************************************
  * brief : 获取软件私有配置"程序运行路径/程序名小写_config.init"的值
  *    par :
  *          val      从配置文件取得的配置值
  *          section  需要取值的域名
  *          key      需要取值的key名
  *    ret :
  *          0  成功
  *         !0  失败
  * note  :
  *          如果配置文件中存在没有域名的配置值,section也要输入,此时section
  *          的值为"ROOT"
  *
*****************************************************************************/
int GetPvSftCfgVal( std::string &val, const char* section, const char* key );
```

(3) 如果需要解析接口返回值的字符含义

```
/*************************************************************
  * brief : 返回PvCfg接口返回值的字符描述
  * note  : err_val为其他接口的返回值
*************************************************************/
const char* PvCfgErrMsg( int ret);
```

3.在项目中实现配置文件值动态更新方法

需要参考文件监视器接口说明:
1.在文件监视器里添加对公共配置文件和私有配置文件的监视;
2.在文件接收处理程序的处理逻辑里调用重新加载配置文件接口

配置文件重新加载接口如下:

```
/*************************************************************
  * brief : 重新加载公共配置文件的配置
  * note  : 在公共配置文件有修改时调用
*************************************************************/
int PvReLoadPubCfg();
```

```
/*****************************************************************
  * brief  : 重新加载软件私有配置文件的配置
  * note   : 在私有配置文件有修改时调用
*****************************************************************/
int PvReLoadSftCfg();
```

附: 样例代码

文件监视器接收处理线程样例FileReceiver

FileReceiver.h 源码

```
/*****************************************************************
 *
 * @file    FileReceiver.h
 *
 * @brief   FileReceiver头文件
 *
 * @author  fu.sky
 *
 * @date    2022-10-31_17:45:34
 *
 * @version V10.010.000
 *
```

```
 ************************************************************/

#ifndef _FILERECEIVER_H_
#define _FILERECEIVER_H_

#include "IEventManager.h"
#include "json/value.h"

class FileReceiver: public CThread, public IEventManager::IEventHandler
{

private:
    FileReceiver();
    ~FileReceiver();

public:
    static int isRun;
    static FileReceiver* Instance();
private:
    static FileReceiver* mInstance;

public:
    void onAppEvent(
            const char* event,
            IEventManager::EVENT_ACTION action,
            Json::Value& eventAttr
            );

    void start( const int &pre_val = -1 );
```

```
    void stop();

private:
    class FileRcvHandler: public Handler
    {
        public:
            FileRcvHandler( FileReceiver* testFileMo );
            ~FileRcvHandler();
            typedef enum file_test_handler_msg
            {
                MSG_STOCK_FILE,
                MSG_CREAT_FILE,
                MSG_MODIF_FILE,
                MSG_DELET_FILE,
                MSG_MOVTO_FILE,
                MSG_MOVFM_FILE,
                MSG_CLOSER_FILE,
                MSG_NULL
            }FILE_TST_HANDLER_MSG;

            void handleMessage( Message& msg );
        private:
            FileReceiver* mRcver;
    };

    Handler*    mHandler;
    Looper*     mLooper;

private:
```

```
    void threadHandler();

};



#endif//_FILERECEIVER_H_
```

FileReceiver.cpp 源码

```
/*************************************************************
 *
 * @file    FileReceiver.cpp
 *
 * @brief   FileReceiver源文件
 *
 * @author  fu.sky
 *
 * @date    2022-10-31_17:45:34
 *
 * @version V10.010.000
 *
 *************************************************************/

#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <libgen.h>
```

```cpp
#include "FileReceiver.h"
#include <string.h>
#include "PvCfg.h"
#include "CustomOutLog.h"
#include "InfraBase.h"

static pthread_mutex_t  gsMutex = PTHREAD_MUTEX_INITIALIZER;

FileReceiver* FileReceiver::mInstance = NULL;
int FileReceiver::isRun = 0;


FileReceiver::FileRcvHandler::FileRcvHandler( FileReceiver* inRcver )
{
    mRcver = inRcver;
}

FileReceiver::FileRcvHandler::~FileRcvHandler()
{

}

void FileReceiver::FileRcvHandler::handleMessage( Message& msg )
{
    //c_write_log(_DEBUG,"evetAttr:[%s]", msg.mAttr.toStyledString().c_str() );

    std::string tpath  = msg.mAttr["Parameter"]["Dir"].asString();
    std::string tfname = msg.mAttr["Parameter"]["FileName"].asString();
    std::string tfop   = msg.mAttr["Parameter"]["Operation"].asString();
```

```
int ret = 0;

c_write_log(_DEBUG,"[%s][%s][%s]]",
        tpath.c_str(), tfop.c_str(), tfname.c_str() );

switch( msg.mWhat )
{

    case MSG_MOVTO_FILE:
    case MSG_CLOSER_FILE:
        {
            if ( PvCfg::mPubCfgPath ==  tpath
                    && PvCfg::mPubCfgFile == tfname )
            {
                //更新公共配置文件
                ret = PvReLoadPubCfg();

                c_write_log(_DEBUG,"PvReLoadPubCfg run ret=[%s]",
                        PvCfgErrMsg(ret) );
            }
            else if ( PvCfg::mSftCfgPath ==  tpath
                    && PvCfg::mSftCfgFile == tfname )
            {
                //更新软件私有配置文件
                ret = PvReLoadSftCfg();

                c_write_log(_DEBUG,"PvReLoadSftCfg run ret=[%s]",
                        PvCfgErrMsg(ret) );
```

```
                }

            }
            break;
        default:
            break;
    }

    return;
}


FileReceiver::FileReceiver()
{
    mHandler = NULL;
}


FileReceiver::~FileReceiver()
{
    if ( mHandler != NULL )
    {
        delete mHandler;
        mHandler = NULL;
    }
}


void FileReceiver::start( const int &pre_val )
```

```
{
    if ( pre_val != -1 )
    {
        while( 0 == pre_val )
        {
            PauseThreadSleep( 0, 10 );
        }
        c_write_log(_DEBUG,"Wait for the prepend to end!");
    }

    if ( mHandler == NULL )
    {
        mHandler = new FileRcvHandler( this );
    }
    else
    {
        return;
    }

    if ( isAlive() )
    {
        return;
    }

    IEventManager::Initialize()->attachEventHandler(
            STORAGE_EVENT,
            (IEventManager::IEventHandler*) this,
            (IEventManager::HANDLER_FUNC) &FileReceiver::onAppEvent
            );
```

```
    startThread();

    c_write_log(_DEBUG,"FileReceiver::start() Done!");

    return;
}


void FileReceiver::stop()
{
    if ( !isAlive() || mHandler == NULL )
    {
        return;
    }

    //Looper 与 线程释放顺序不能颠倒
    stopThread();
    mLooper->decRef();
    return;
}


void FileReceiver::onAppEvent(
        const char* event,
        IEventManager::EVENT_ACTION action,
        Json::Value& eventAttr
        )
{
```

```
if ( strcmp( event, STORAGE_EVENT) != 0 )
{
    return;
}

Message msg;

//打印json值eventAttr (格式化后输出)
c_write_log(_DEBUG,"eventAttr=[%s]!", eventAttr.toStyledString().c_str() );

std::string tOpName = eventAttr["Parameter"]["Operation"].asString();

c_write_log(_DEBUG,"topName=[%s]!",tOpName.c_str() );

if ( tOpName == STORAGE_FILE_STOCK )
{
    msg.mWhat = FileRcvHandler::MSG_STOCK_FILE;
}
else if ( tOpName == STORAGE_FILE_ADDED )
{
    msg.mWhat = FileRcvHandler::MSG_CREAT_FILE;
}
else if ( tOpName == STORAGE_FILE_MODIFY )
{
    msg.mWhat = FileRcvHandler::MSG_MODIF_FILE;
}
else if ( tOpName == STORAGE_FILE_DELETED )
{
    msg.mWhat = FileRcvHandler::MSG_DELET_FILE;
```

```
    }
    else if ( tOpName == STORAGE_FILE_MOVEDTO )
    {
        msg.mWhat = FileRcvHandler::MSG_MOVTO_FILE;
    }
    else if ( tOpName == STORAGE_FILE_MOVEDFROM )
    {
        msg.mWhat = FileRcvHandler::MSG_MOVFM_FILE;
    }
    else if ( tOpName == STORAGE_FILE_CLOSEWRITE )
    {
        msg.mWhat = FileRcvHandler::MSG_CLOSER_FILE;
    }
    else
    {
        msg.mWhat = FileRcvHandler::MSG_NULL;
    }

    msg.mMetaStr = event;
    msg.mArg1    = action;
    msg.mAttr    = eventAttr;
    msg.setValid( true);
    msg.mTarget  = mHandler;
    msg.mTargetLooper = mLooper;

    mHandler->sendMessage( msg );

    return;
}
```

```
void FileReceiver::threadHandler()
{

    pid_t tid;
    tid = syscall(SYS_gettid);
    c_write_log(_INFO,"thread id[%d]", tid);


    Looper* me = Looper::getLooper();
    mLooper = me;
    mLooper->incRef();
    mLooper->prepare();

    isRun = 1;

    mLooper->Loop();
}

FileReceiver* FileReceiver::Instance()
{
    if ( mInstance == NULL )
    {
        pthread_mutex_lock( &gsMutex );
        if ( mInstance == NULL )
        {
            mInstance = new FileReceiver();
        }
        pthread_mutex_unlock( &gsMutex );
```

```
    }

    return mInstance;
}
```

测试主程序样例

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

#include "IEventManager.h"
#include "FileMonitor.h"
#include "FileReceiver.h"
#include "InfraBase.h"
#include "PvCfg.h"
#include "CustomOutLog.h"


int main()
{

    pid_t tid;
    tid = syscall(SYS_gettid);
    c_write_log(_INFO,"thread id[%d]", tid);
```

```
int ret = 0;

const char* sftCfgFile = GetPvRunPar( RUNPAR_CFGFILE );  //私有配置文件
const char* pubCfgFile = GetPvPubFile( PV_PUBCFG_FILE ); //公共配置文件
const char* sftVerFile = GetPvRunPar( RUNPAR_VERFILE );  //软件版本文件

c_write_log(_DEBUG,"pubCfgFile[%s],sftCfgFile[%s],sftVerFile[%s]",
        pubCfgFile, sftCfgFile, sftVerFile );

//将当前程序版本号用minIni开源方法写入程序版本ini文件
// ini_puts  return: 1:成功   0:失败
ret = ini_puts( "cur_version", "ver_no", "v20.03.050", sftVerFile );
if ( ret != 1 )
{
    c_write_log(_ERROR,"ini_puts(%s)run ret=[%d]", sftVerFile, ret );
    return 1;
}

//加载配置文件
ret = PvCfgInitialize();
if ( ret != PVCFG_SUCCESS )
{
    c_write_log(_DEBUG,"PvCfgInitialize()run ret=[%s]", PvCfgErrMsg(ret) );

    return 1;

}

//启动事件管理线程
```

```
IEventManager::Initialize()->start();

//响应文件监视器的处理线程
FileReceiver::Instance()->start( IEventManager::isRun );

//启动文件监视器线程
FileMonitor::Instance()->start( FileReceiver::isRun );

//将公共配置文件添加到文件监视器中进行监视
FileMonitor::Instance()->addWatch(
        PvCfg::mPubCfgPath.c_str(),          /*路径*/
        (EVENT_MOVE_TO | EVENT_CLOSEWRITE), /*类型*/
        PvCfg::mPubCfgFile.c_str()           /*文件名*/
        );

//将软件私有配置文件添加到文件监视器中进行监视
FileMonitor::Instance()->addWatch(
        PvCfg::mSftCfgPath.c_str(),
        (EVENT_MOVE_TO | EVENT_CLOSEWRITE),
        PvCfg::mSftCfgFile.c_str()
        );


std::string tCfgVal;

//取公共配置文件: 域[INFO] key:faultstate 的值
ret = GetPvPubCfgVal( tCfgVal, "INFO", "faultstate" );

c_write_log(_DEBUG,"GetPvPubCfgVal [INFO]->faultstates=[%s] ret=[%s]",
```

```
            tCfgVal.c_str(), PvCfgErrMsg(ret) );

    //取私有配置文件: 域[ROOT] key:loglevel 的值
    ret = GetPvSftCfgVal( tCfgVal, "ROOT", "loglevel" );
    c_write_log(_DEBUG,"GetPvSftCfgVal [ROOT]->loglevel=[%s] ret=[%s]",
            tCfgVal.c_str(), PvCfgErrMsg(ret) );


    while(1)
    {
        PauseThreadSleep( 10, 0 );

        ret = GetPvPubCfgVal( tCfgVal, "INFO", "faultstate" );
        c_write_log(_DEBUG,"GetPvPubCfgVal [INFO]->faultstates=[%s] ret=[%s]",
                tCfgVal.c_str(), PvCfgErrMsg(ret) );

        ret = GetPvSftCfgVal( tCfgVal, "ROOT", "loglevel" );
        c_write_log(_DEBUG,"GetPvSftCfgVal [ROOT]->loglevel=[%s] ret=[%s]",
                tCfgVal.c_str(), PvCfgErrMsg(ret) );
    }

    return 0;
}
```