线程池：管理一个线程队列和一个任务队列，必要实现信号量

线程管理类：

#pragma once

#include "Threadpool/threadpool.h"

class **QThreadManage**

{

public:

static QThreadManage\* **getInstance**()

{

std::lock\_guard<std::mutex> locker(*m\_mutex\_*);

if(nullptr == m\_pInstance)

{

m\_pInstance = new QThreadManage();

}

return m\_pInstance;

}

static void **releaseInstance**()

{

std::lock\_guard<std::mutex> locker(*m\_mutex\_*);

if(nullptr != m\_pInstance)

{

m\_pInstance->stop();

delete m\_pInstance;

m\_pInstance = nullptr;

}

}

private:

**QThreadManage**() = default;

~**QThreadManage**();

**QThreadManage**(const QThreadManage&) = delete;

QThreadManage& operator=(const QThreadManage&) = delete ;

public:

/\*!

\* \brief initial 初始化

\* \param threads\_num 线程数

\* \return 成功返回0，失败返回错误代码

\*/

zhb::TP\_CODE **initial**(size\_t threads\_num);

/\*!

\* \brief stop 停止

\*/

void **stop**();

/\*!

\* \brief setKey 设置占位符

\* \param key 占位符

\* \return 成功返回0，失败返回错误代码

\*/

zhb::TP\_CODE **setKey**(size\_t key);

/\*!

\* \brief delete\_key 删除占位符

\* \param key 占位符

\* \return 成功返回0，失败返回错误代码

\*/

zhb::TP\_CODE **deleteKey**(size\_t key);

/\*!

\* \brief postNormalLevelTask 占位插入, 普通级别任务

\* \param key 占位符

\* \param f 绑定函数

\* \param args 绑定参数

\* \return 成功返回0，失败返回错误代码

\*/

template<class F, class ...Args>

zhb::TP\_CODE **postNormalLevelTask**(size\_t key, F &&f, Args && ...args)

{

if(m\_pool\_)

{

return m\_pool\_->post\_normal\_task\_with\_key(key, std::move(f), std::move(args)...);

}

return zhb::TP\_CODE::Threadpool\_Not\_Initial;

}

/\*!

\* \brief postHighLevelTask 占位插入, 高级别任务

\* \param key 占位符

\* \param f 绑定函数

\* \param args 绑定参数

\* \return 成功返回0，失败返回错误代码

\*/

template<class F, class ...Args>

zhb::TP\_CODE **postHighLevelTask**(size\_t key, F &&f, Args && ...args)

{

if(m\_pool\_)

{

return m\_pool\_->post\_high\_task\_with\_key(key, std::move(f), std::move(args)...);

}

return zhb::TP\_CODE::Threadpool\_Not\_Initial;

}

/\*!

\* \brief postHighLevelTask 增加任务，优先查找空闲线程

\* \param f 绑定函数

\* \param args 绑定参数

\* \return 成功返回0，失败返回错误代码

\*/

template<class F, class ...Args>

zhb::TP\_CODE **postTask**(F &&f, Args && ...args)

{

if(m\_pool\_)

{

return m\_pool\_->post\_task(std::move(f), std::move(args)...);

}

return zhb::TP\_CODE::Threadpool\_Not\_Initial;

}

private:

static std::mutex m\_mutex\_;

static QThreadManage \*m\_pInstance;

std::unique\_ptr<zhb::threadpool> m\_pool\_{nullptr};

};

实现类：

#include "qThreadManage.h"

std::mutex QThreadManage::m\_mutex\_;

QThreadManage \*QThreadManage::m\_pInstance = nullptr;

QThreadManage::~**QThreadManage**()

{

if(m\_pool\_)

{

stop();

m\_pool\_.reset();

}

}

zhb::TP\_CODE QThreadManage::**initial**(size\_t threads\_num)

{

if(nullptr == m\_pool\_)

{

m\_pool\_ = std::make\_unique<zhb::threadpool>(*threads\_num*);

}

return zhb::TP\_CODE::Threadpool\_Success;

}

void QThreadManage::**stop**()

{

if(m\_pool\_)

{

m\_pool\_->stop();

}

}

zhb::TP\_CODE QThreadManage::**setKey**(size\_t key)

{

if(m\_pool\_)

{

return m\_pool\_->set\_key(key);

}

return zhb::TP\_CODE::Threadpool\_Not\_Initial;

}

zhb::TP\_CODE QThreadManage::**deleteKey**(size\_t key)

{

if(m\_pool\_)

{

return m\_pool\_->delete\_key(key);

}

return zhb::TP\_CODE::Threadpool\_Not\_Initial;

}

使用：单列模式

通过调用getinstan ()产生实例

QThreadManage \*m\_pPool{nullptr}; //线程

m\_pPool = QThreadManage::getInstance(); //产生实例

m\_pPool->initial(8);