MP核心模块设计

资源管理

在24×7hour不断为用户提供服务的服务器系统设计中，资源管理是一个重要的问题，资源管理策略的好坏往往决定了系统的稳定性。

目前服务器系统设计中用的最多也是最有效的策略便是“池”式的资源管理策略，“池”提供了一种资源复用的机制，对于某些资源，创建和销毁资源的开销是很大的，因此在程序运行过程中频繁的创建和释放资源将会带来很大的开销，但是如果对这类资源采用池式的管理方式，即创建资源时从池中申请，释放资源时把资源归还到池中，将会使系统的资源管理策略更高效与实用。

在服务器系统设计中，最重要的池可以分为“内存池”、“线程池”、“对象池”等。其中内存池主要是为了解决程序运行时频繁的小块内存区块的分配和释放带来的内存碎片；线程池是为了避免频繁创建线程和销毁线程时的开销；对象池则是为了使某一类的对象可以被复用。

下面，我们会详细讲解MP服务端得池式设计理念。

CPoolBase抽象类

CPoolBase.h

// 资源池

class CPoolBase

{

public:

CPoolBase()

{

m\_nInit = 10;

};

CPoolBase(int nInit);

virtual ~CPoolBase();

virtual bool InitPool(int nInit);

virtual void ClearPool(); //清空资源池，释放资源池中的所有对象

protected:

int m\_nInit; //资源池中默认初始资源数

CPtrList m\_stackStay; //资源池，池中的元素为void\*，可指向任意类型的资源对象

//纯虚函数，具体的资源对象的创建和释放由派生类定义

virtual LPVOID NewFunc() = 0; //创建新资源对象

virtual void DeleteFunc(LPVOID tp) = 0;//释放资源对象

public:

// 从资源池中获取一个资源，如果资源池已没有可用的资源，则创建一个

inline virtual LPVOID PopOrg();

// 从资源池中获取一个资源，如果资源池已没有可用的资源，则创建一个

// bNewItem指示返回的资源是否是新创建的

inline virtual LPVOID PopOrg(BOOL& bNewItem);

// 归还不再使用的资源（不释放），即把资源加入到资源池中

inline virtual void PushOrg(LPVOID tp);

// 从资源池中删除特定的资源

inline void Remove(LPVOID tp);

};

CPoolBase.cpp

#include "PoolBase.h"

CPoolBase::CPoolBase(int nInit)

{

InitPool(nInit);

}

CPoolBase::~CPoolBase()

{

ClearPool();

}

bool CPoolBase::InitPool(int nInit)

{

m\_nInit = nInit;

for (int i = 0; i < nInit; i++)

{

LPVOID tp = NewFunc(); //创建资源对象

if (NULL != tp)

{

//将刚才创建好的资源对象加入资源池中

m\_stackStay.AddTail(tp);

}

else

{

return FALSE; //创建资源对象失败

}

}

return TRUE;

}

void CPoolBase::ClearPool()

{

POSITION pos = m\_stackStay.GetHeadPosition();

LPVOID tp;

while (NULL != pos)

{

tp = m\_stackStay.GetNext(pos);

DeleteFunc(tp); //释放资源对象

}

m\_stackStay.RemoveAll(); //从资源池中移除所有资源

}

// 从池中申请一个资源对象

LPVOID CPoolBase::PopOrg()

{

LPVOID tp = NULL; //资源对象的指针，类型为void\*

//如果资源池中还有资源

if (0 < m\_stackStay.GetCount())

{

//获的资源对象的指针

tp = m\_stackStay.GetTail();

//资源申请者已获取资源对象，则将该资源对象从资源池中移除

m\_stackStay.RemoveTail();

}

//资源池中已无任何可用资源对象

else

{

tp = NewFunc(); //重新创建一个新的资源对象

}

return tp; //将资源对象的指针返回给申请者

}

// 此函数功能同上，参数bNewItem指示返回的资源对象是从资源池中获取的

//（FALSE）？还是新创建的（TRUE）？

LPVOID CPoolBase::PopOrg(BOOL& bNewItem)

{

LPVOID tp = NULL;

if (0 < m\_stackStay.GetCount())

{

bNewItem = FALSE;

tp = m\_stackStay.GetTail(); m\_stackStay.RemoveTail();

}

else

{

bNewItem = TRUE;

tp = NewFunc();

}

return tp;

}

// 归还资源

void CPoolBase::PushOrg(LPVOID tp)

{

ASSERT(NULL != tp);

m\_stackStay.AddTail(tp); //将资源归还至资源池中统一管理

}

//从资源池中移除某一特定的资源

void CPoolBase::Remove(LPVOID tp)

{

if (NULL == tp)

return;

POSITION pos = m\_stackStay.Find(tp);

if (NULL != pos)

{

DeleteFunc(tp); //释放资源

m\_stackStay.RemoveAt(pos); //从资源池中移除此资源

}

}

CPoolBaseEx.h

class CPoolBaseEx : public CPoolBase

{

public:

CPoolBaseEx();

virtual ~CPoolBaseEx()

{

};

virtual void ClearPool();

virtual void ReinitPool();

protected:

LONG m\_nActionCount; // 已从资源池中获取的资源对象数目

virtual LPVOID NewFunc() = 0;

virtual void DeleteFunc(LPVOID tp) = 0;

public:

inline virtual LPVOID PopOrg();

inline virtual LPVOID PopOrg(BOOL& bNewItem);

inline virtual void PushOrg(LPVOID tp);

inline void Remove(LPVOID tp);

// 获取资源池中总的资源对象数目，由2部分组成：

// 1、资源池中剩余的资源对象数目，此类资源可被外界申请

// 2、已被外界获取的资源对象数目

inline int GetSize();

// 已被外界获取的资源对象数目

inline int GetActionSize();

// 资源池中剩余的资源对象数目

inline int GetStaySize();

};

CPoolBaseEx

CPoolBaseEx::CPoolBaseEx(): m\_nActionCount(0)

{

}

void CPoolBaseEx::ClearPool()

{

CPoolBase::ClearPool();

}

// 重新初始化资源池

void CPoolBaseEx::ReinitPool()

{

int nActionPool = (int)m\_nActionCount; // 已被外界获取的资源对象数目

int nDelayPool = (int)m\_stackStay.GetCount(); // 资源池中剩余的资源对象数目

if ((nActionPool + nDelayPool) > m\_nInit)

{

int nTotalPool = nActionPool + nDelayPool; // 资源对象总数

int nDeletePool= nTotalPool - m\_nInit; // 需要释放的资源对象数目

if (nDelayPool < nDeletePool)

nDeletePool = nDelayPool;

//释放一部分资源对象

for (int i = 0; i < nDeletePool; i++)

{

LPVOID lpData = m\_stackStay.GetTail();

m\_stackStay.RemoveTail();

DeleteFunc(lpData);

}

}

}

LPVOID CPoolBaseEx::PopOrg()

{

LPVOID tp = NULL;

if (0 < m\_stackStay.GetCount())

{

tp = m\_stackStay.GetTail();

m\_stackStay.RemoveTail();

}

else

{

tp = NewFunc();

}

if(NULL != tp)

{

::InterlockedIncrement(&m\_nActionCount); //已获取的资源对象数目加1

}

return tp;

}

LPVOID CPoolBaseEx::PopOrg(BOOL& bNewItem)

{

LPVOID tp = NULL;

if (0 < m\_stackStay.GetCount())

{

bNewItem = FALSE;

tp = m\_stackStay.GetTail();

m\_stackStay.RemoveTail();

}

else

{

bNewItem = TRUE;

tp = NewFunc();

}

if(NULL != tp)

{

::InterlockedIncrement(&m\_nActionCount);

}

return tp;

}

void CPoolBaseEx::PushOrg(LPVOID tp)

{

ASSERT(NULL != tp);

m\_stackStay.AddTail(tp);

::InterlockedDecrement(&m\_nActionCount); // 已获取的资源对象数目减1

}

void CPoolBaseEx::Remove(LPVOID tp)

{

if (NULL == tp)

{

return;

}

CPoolBase::Remove(tp);

}

int CPoolBaseEx::GetSize()

{

return (int)(m\_stackStay.GetCount() + m\_nActionCount);

}

int CPoolBaseEx::GetActionSize()

{

return m\_nActionCount;

}

int CPoolBaseEx::GetStaySize()

{

return (int)(m\_stackStay.GetCount());

}

CPool.h

#include "PoolBase.h"

**//模板参数T是资源对象的类型**

template <class T>

class CPool : public CPoolBaseEx

{

public:

CPool<T>()

{

};

**//在构造CPool<T>类对象时，就创建一部分资源对象并放入资源池中进行统一管//理**

CPool<T>(int nInit)

{

InitPool(nInit);

};

**//销毁CPool<T>对象时，释放资源池的所有资源对象**

virtual ~CPool()

{

ClearPool();

};

//获取一个资源对象

virtual T\* Pop()

{

return (T\*)CPoolBaseEx::PopOrg();

}

//归还一个资源对象

virtual void Push(T\* tp)

{

CPoolBaseEx::PushOrg(tp);

}

protected:

//创建一个新的资源对象，返回其指针

virtual LPVOID NewFunc()

{

return (LPVOID)(new T);

}

//释放一个资源对象

virtual void DeleteFunc(LPVOID tp)

{

delete (T\*)tp;

tp = NULL;

}

};