网络游戏 Server 编程

ONLINE GAME SERVER PROGRAMMING

(韩)韩东勋 著
马晓阳 刘 娟 译
飞思科技产品研发中心 监制

C/C++ 开发专家

网络游戏服务器开发的经典图书。
全面剖析应用开发技术，学有所用。
详细阐释网络游戏服务器编程的精髓。

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

网络游戏 Server 编程

ONLINE GAME SERVER PROGRAMMING

(韩)韩东勋 著
马晓阳 刘 娟 译
飞思科技产品研发中心 监制

C/C++ 开发专家

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

本书向读者展示了网络游戏服务器设计和开发的基础知识及实战案例，首先介绍了 C/C++的基础知识、开发服务器所必需的基础类及通信类；然后说明了 C/C++基础知识中的继承（Inheritance）、重载（Overloading）等概念，以及基础类中的 Log、Memory 类和 Thread 类；接着在通信类中对服务器编程所需的 Winsock 及 IOCP 进行了说明；还介绍了如何运行库和类设计游戏服务器；另外，介绍了其他书籍中不常见到的性能监视器（Performance Monitor）、Packet 管理框架、MiniDump、UDP Hole punching 等内容；最后设计了 PIGU 游戏服务器。

本书适合广大网络游戏开发人员参考学习，同时也可作为高等院校相关专业师生的参考用书。

版权贸易合同登记号 图字：01-2007-3436

图书在版编目（CIP）数据

网络游戏 Server 编程 /（韩）韩东勋著；马晓阳，刘娟译.—北京：电子工业出版社，2009.3

（C/C++开发专家）

书名原文：Online Game Server Programming

ISBN 978-7-121-07644-2

Ⅰ. 网… Ⅱ.①韩…②马…③刘… Ⅲ.游戏－网络服务器－程序设计 Ⅳ.TP368.5

中国版本图书馆 CIP 数据核字（2008）第 167767 号

责任编辑：杨 鸽

印　　刷：北京智力达印刷有限公司

装　　订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱　邮编：100036

开　　本：787×980　1/16　印张：21.5　字数：550.4 千字

印　　次：2009 年 3 月第 1 次印刷

印　　数：4 000 册　　定价：39.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

作 者 序

16年前，父亲曾经送给我一本电脑读物作为初中的入学礼物，记得当时的电脑只是用于游戏和文字处理，而如今电脑已经成了我们生活中必不可少的家电产品，小时候在科幻电影中才能看到的电脑订餐、网络购物也已经成为了现实。不过十余年，电脑技术已经有了惊人的发展和变化。电脑技术的发展可以说是日新月异。在以前，KB已经算是很大的容量，而现在，人们已经不满足于TB的容量了。本书将要介绍的就是历史最为悠久的游戏编程。回顾一下自己的游戏历程：最早接触游戏早在初中时候，印象最深的就是趁父母睡觉时偷偷爬起来玩游戏，可以说当时痴迷于游戏。到了高中，就有了自己设计游戏的想法，最后不顾父母的反对报考了计算机专业。大学毕业之后也就开始步入游戏相关领域，直到现在，笔者还对游戏有着非同一般的热情。因此，谨以此书献给和我一样有着游戏开发热情的广大读者。

本书向读者展示了网络游戏服务器设计和开发的基础知识，以及实战案例。首先介绍了C/C++的基础知识、开发服务器所必需的基础类及通信类；然后说明了C/C++基础知识中的继承（Inheritance）、重载（Overloading）等概念，以及基础类中的Log类、Memory类和Thread类；接着在通信类中对服务器编程所需的Winsock及IOCP进行了说明；还介绍了如何运行库和类设计游戏服务器；另外，还介绍了其他书籍中不常见到的性能监视器（Performance Monitor）、Packet管理框架、MiniDump、UDP Hole punching等内容，最后设计出PIGU游戏服务器。

感谢帮助我和鼓励我的各位朋友，尤其是情报文化社的南武铉组长、李美香女士。

感谢一直以来信任、鼓励我的父母。

最后还要感谢我亲爱的妻子，正是你的关爱和支持，才使我有了今天的收获。

著 者

联系方式

咨询电话：（010）88254160 88254161-67
电子邮件：support@fecit.com.cn
服务网址：http://www.fecit.com.cn http://www.fecit.net
通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

反侵权盗版声明

目 录

第 1 章　基础知识

本章主要介绍网络游戏在线编程的基础知识。

1.1　什么是模块化

1.1.1　为什么要模块化

将所有对象囊括到一个项目中，再通过类进行分离，也是游戏开发的一种方法。不过，最近游戏开发项目的范围逐渐扩展，单一的游戏开发已经无法设计开发更全更好的东西。因此，现在的服务器程序大致分为 3 类：引擎模块、游戏逻辑模块及管理模块。引擎模块的功能主要是网络通信、加密、内存管理等；游戏逻辑模块随着游戏的不同，内容也不同，主要是承担着实际游戏的运行；管理模块主要是服务于游戏，并且根据游戏的状态进行更新或者维护，起到管理的功能。

为什么要将模块划分为引擎模块、游戏逻辑模块及管理模块 3 类进行管理，这样划分管理有什么优点呢？

- 首先，便于源代码的重复调用。不管是开发什么游戏，引擎模块及管理模块基本没有什么变化，因此可以重复调用；也就是说，开发新游戏项目的时候，引擎模块及管理模块都可以重复调用，将重点放在游戏逻辑模块就可以了。
- 其次，便于源代码的共享。大部分的客户端开发人员也都不太精通网络编程，因此，服务器编程人员必须另外制作网络模块，如果通信模块分离为引擎模块的话，就可以很容易地提供模块，并能编写出和服务器通信的源代码。
- 最后，便于源代码的管理。修正细小的定义或者内容的时候，如果模块化准确的话，只需要修正相应模块就可以了；否则，就需要检索出需要修正的部分一一进行处理。

笔者在开发网络游戏的时候就经常进行模块化编程，然后逐渐对上述 3 种模块中的引擎模块及管理模块进行升级，这样，开发新游戏项目的时候就会节省很多时间。

1.1.2 如何进行模块化

那么如何进行模块化呢？可以采用的模块化方法有 LIB、DLL、COM 等，这些方法都有其优点和缺点，游戏开发人员可以根据各自项目选择合适的方法，下面介绍一下采用 LIB 进行模块化的方法，制作 LIB 文件的方法非常简单。

新建一个项目文件，如图 1-1 所示。

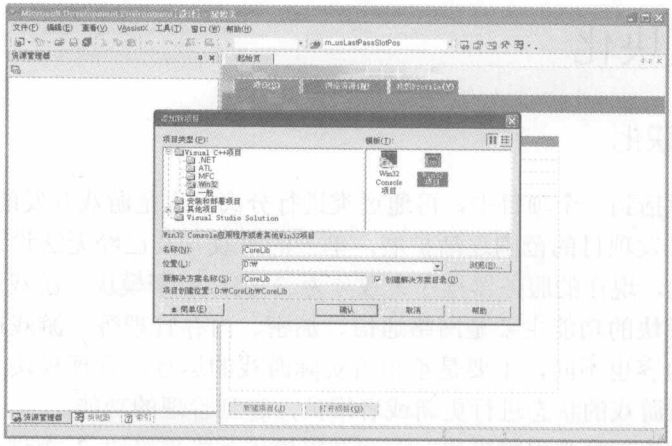

图 1-1　新建项目页面

新建项目之后，选择 LIB 文件（静态库）就可以了。预编译头选项可以根据个人喜好进行选择，本文是选择了预编译头的选项，如图 1-2 所示。设置完 LIB 文件选项之后，就算是新建了一个 LIB 文件，接下来要进行的是基本的引擎模块及管理模块操作。

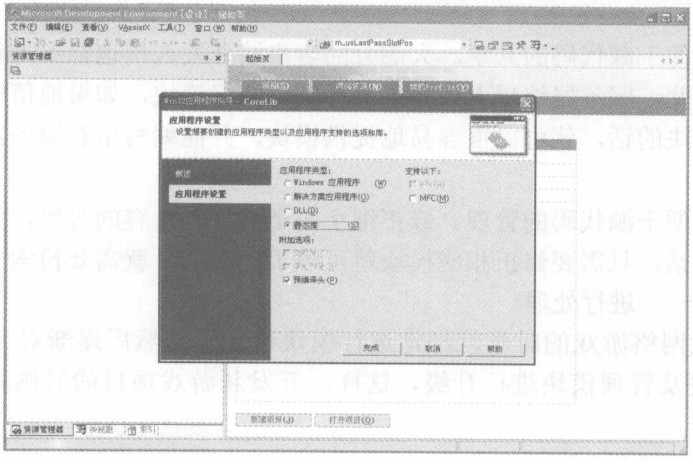

图 1-2　LIB 文件选项页面

1.2　函数指针（Delegate）

前面介绍了如何进行模块化，并制作了新建 LIB 文件的项目，那么接下来就应该在 LIB 文件中添加引擎模块及管理模块，不过，有必要先介绍一下服务器编程中必需的基础知识。首先就是函数指针（Delegate），可能有的开发人员对于模块化的基础知识已经十分了解了，但是进行模块化的话，大家可能会在进行某种操作的时候想要在特定角度得到相应结果。简单地说，函数指针所具有的功能就如同 Windows 编程的事件一样。我们在调用 API 进行编码的时候经常会看到这类函数，Callback 就是最具代表性的一例。为了便于大家的理解，下面通过代码进行一下说明。

编程人员 A 编写了下面的函数。

```
void DelegateSample(void (*fp) (int ) , int data)
 {
printf ("# DelegateSample fuction #\ n") ;
fp(data / 10, data / 100) ;
}
```

然后，编程人员 B 为了使用上面的函数，编写了下面的代码。

```
void Callback(int result1, int result2)
 {
printf ("# result1 : %d, result2 : %d", result1, result2) ;
}
void main(void)
{
DelegateSample(Callback, 500) ;
}
```

编程人员 B 调用了函数 DelegateSample，该函数可以将编程人员 A 编写的数据除以 10 及 100 后得到的值返回为结果，通过不是返回值或参数的函数得到结果，这里使用的就是函数指针，C/C++相关书籍对此有更为详细的介绍。现在可能还看不出模块化有什么不同的效果，不过这与使用 C++继承的 virtual 都是模块化所必需的基础知识。

```
#include <stdio.h>
void DelegateSample(void (*fp) (int , int )/ *函数指针声明*/ , int data)
 {
printf ("# DelegateSample fuction #\ n") ; fp(data / 10, data / 100) ;
}
void Callback(int result1, int result2) //函数指针 Callback 函数
 {
printf ("# result1 : %d, result2 : %d", result1, result2) ;
}
```

```
void main(void)
{
DelegateSample(Callback, 500) ;
}
```

1.3 Windows函数的使用

1.3.1 为什么要使用 Windows 函数

在进行操作的时候，一般都是使用 int、char 等 ANSI 标准函数，但是，在进行游戏服务器编程的时候，使用更多的是由 Windows 函数构成的 Windows API 函数。大部分 ANSI 标准函数和 Windows 函数的使用是相似的，为了减少由于函数类型转换导致的错误，最好以 Windows 函数为基础进行操作。下面针对 Windows 函数和 ANSI 标准函数做一下比较说明，如表 1-1 所示。

表 1-1 Windows 函数和 ANSI 标准函数的比较及说明

Windows 函数	ANSI 标准函数	说　明
BOOL	bool	TRUE/FALSE 变量
BYTE	signed char	8bit 变量
DWORD	unsigned long	32bit 整数变量
DWORD_PTR	unsigned long	32bit 整数指针变量
FLOAT	float	32bit 实数变量
INT	int	32bit 整数变量
INT32	int	32bit 整数变量
INT64	_int64	64bit 整数变量
CHAR	char	8bit 变量
LPCSTR	const char*	8bit 字符串变量
LPCTSTR	char* 或者 wchar*	因 MultiByte、Unicode 而不同
VOID	void	

如表 1-1 所示的都是最常用的变量，本书后面所介绍的案例都是参照以上函数进行制作的。大家可能会对表中的“LPCTSTR”感到奇怪，分开来写的话就是“CONST TCHAR*”，后面会进行详细说明（更加详细的 Windows 函数可以参照 MSDN 中名为 Windows Data Types 的节）。

1.3.2 常用函数的使用

除了 Windows 函数之外，常用的内存相关函数大多也是由 Windows 函数提供的，

下面介绍几种有代表性的常用函数，如表 1-2 所示。

表 1-2　Windows 函数和 ANSI 标准函数的比较及说明

Windows	ANSI	说　明
FillMemory	memset	用于指定大小内存的数据填充
ZeroMemory	memset	用于指定大小内存的清零初始化
CopyMemory	memcpy	用于指定大小内存的复制
MoveMemory	memmove	用于指定大小内存的复制

函数原型

VOID FillMemory(PVOID Destination, SIZE_T Length, BYTE Fill)	
PVOID Destination	添加数值的内存
SIZE_T Length	添加数值的内存大小
BYTE Fill	要添加的 1Byte 值

VOIDZeroMemory(PVOIDDestination, SIZE_T Length)	
PVOID Destincation	进行初始化的内存
SIZE_T Length	进行初始化的内存大小

VOIDCopyMemory(PVOIDDestination, CONST VOID* Source, SIZE_T Length)	
PVOID Destination	进行复制的内存
CONST VOID* Source	进行复制的原文件内存
SIZE_T Length	进行复制的内存大小

VOIDMoveMemory(PVOIDDestination, CONST VOID* Source, SIZE_T Length)	
PVOID Destination	进行复制的内存
CONST VOID* Source	进行复制的原文件内存
SIZE_T Length	进行复制的内存大小

通过上面介绍的函数可以了解到 SIZE_T 类型的参数，在 32bit 环境下没有什么区别，不过在如今超过 64bit 的环境下，这种函数就显得比较重要了。一般而言，SIZE_T 都是声明为 unsigned int，调用内存相关函数的使用时经常会看到 SIZE_T 类型的变量。也就是说，SIZE_T 类型的变量主要用于表现内存地址。

一般的 32bit 环境下是 32bit，64bit 环境下是 64bit。这里需要注意的是，如果随意地将 SIZE_T 视为 32bit 进行编码的话，以后再想将其转换为 64bit 时就会出现问题。

1.3.3 TCHAR 的使用

可以说，ANSI 标准类型的 CHAR 函数是大家比较熟悉的。这种环境也被称为 MultiByte 环境，这种函数也便于处理英文或字符为 8bit 的环境。当然，也可以在 CHAR 中使用韩文、中文等其他语言，但是需要采用几种简便方法。如果想要将游戏出口至国外的话，为了将字符串代码导致的问题最小化，最好通过 Unicode 处理字符串，TCHAR 正是应对一部分 MultiByte 形态项目的产物。根据想要在编译选项中选择的对象，TCHAR 声明的变量可以自动转换为 MultiByte 形态和 Unicode 形态进行使用，如图 1-3 所示。TCHAR 看似非常简单、方便，不过有些地方也需要特别注意，下面介绍 4 个需要注意的事项。

图 1-3 编译选项中修改 MultiByte、Unicode 的界面

第一，在直接利用 CopyMemory 等函数处理 TCHAR 类型数据的时候要十分小心。选项不同，sizeof(TCHAR)的大小会出现错误，必须编写如下源代码。

```
TCHAR SampleData[64] ;
Zero(SampleData, 64 * sizeof (TCHAR)) ;
```

在上述源代码中，如果没有*sizeof(TCHAR)的话，在 MultiByte 环境中没有什么问题，但是在 Unicode 环境中，大小会出现问题，并影响到数据。也就是说，在 MultiByte 环境中，内存可以正常初始化，但是在 Unicode 环境中，会出现仅初始化一半的现象。这也是非常重要的问题，所以最好能编写源代码。

第二，进行指针运算的时候需要注意。由于 CHAR 函数是 1Byte，习惯性地进行

指针+1、–1 运算的时候，理所当然地每 1Byte 进行移动。但是，对于 TCHAR 的 TCHAR 环境而言，+1 的话，实际上是 2Byte 移动。简单举例说明一下：

```
TCHAR SampleData[64] = {1,2,3,4,5,6,7,8,9,0};
TCHAR *pPointer = SampleData + 6;
```

如上所述，编写源代码的话，pPointer 为 MultiByte，或者是 Unicode，或者是值 7。也就是说，Unicode 环境下是 2Byte，所以进行+6 运算的话，想成是 4 就是错误的。进行指针运算的时候，+、–随着前面变量类型而变化。第一项和第二项原来是经常出错的重要事项，下面再介绍一个样本源代码。

```
#include <tchar .h>
#include <window.h> void _t main(void) {
TCHAR SampleData [10] = {1,2,3,4,5,6,7,8,9,0};
TCHAR *pPointer = SampleData + 6;
BYTE Data[64] = {0,};
CopyMemory(Data , *pPointer , 2 * sizeof (TCHAR)) ;
}
```

如前所述，在 CopyMemory 中，如果不进行 sizeof(TCHAR)的话，在 Unicode 环境中就很难复制实际所需的长度。因此，在上面的源代码中，Data 变量中存在 7、8 的值。

第三，必须修改原来使用的字符串函数。也就是说，必须改变原有的 strcpy、strcmp 等函数，代表性的函数如表 1-3 所示。

表 1-3　TCHAR 使用的字符串函数

TCHAR	CHAR	说　明
_tcscpy	strcpy	用于复制字符串
_tcsncpy	strncpy	用于复制指定字符串的长度
_tcscmp	strcmp	比较字符串
_tprintf	printf	将字符串输出到界面
_stprintf	stprintf	将指定格式的字符串输入到指定的缓冲区
_sntprintf	sntprintf	按照指定的长度将指定格式的字符串输入到指定的缓冲区

函数原型

TCHAR* _tcscpy(TCHAR*st rDesti nation, CONST TCHAR*strSource)	
RETURN	strDestination 的指针
TCHAR*strDestination	将要复制字符串的内存
TCHAR*strSource	源文件字符串

TCHAR* _tcsncpy(TCHAR*strDestination, CONST TCHAR*strSource, SIZE_T count)	
RETURN	strDestination 的指针
TCHAR*strDestination	将要复制字符串的内存
CONST TCHAR*strSource	源文件字符串
SIZE_T count	将要复制的字符串的长度

INT _tcscmp(CONST TCHAR*st ri ng1, CONST TCHAR*st ring2)	
RETURN	在值为 0 的情况下相同、非 0 的情况下不相同的字符串
CONST TCHAR*string1	要比较的字符串 1
CONST TCHAR*string2	要比较的字符串 2

INT _tprintf(CONST TCHAR*format , [argument])	
RETURN	输出的字符串的个数
CONST TCHAR*format	格式化字符串

INT _stprintf(TCHAR*buffer, CONST TCHAR*format , [argument])	
RETURN	输出的字符串的个数
TCHAR*buffer	输入字符串的内存
CONST TCHAR*format	格式化字符串

INT_sntprintf(TCHAR*buffer, SIZE_T count , CONST TCHAR*format , [argument])	
RETURN	输出的字符串的个数
TCHAR*buffer	输入字符串的内存
SIZE_T count	将要输入的字符串的个数
CONST TCHAR*format	格式化字符串

第四，使用的大部分字符串必须通过“_T("")”进行组合。

```
_t printf ("TEST STRING\ n") ; //错误
_t printf (_T("TEST STRING\ n")) ; //成功
```

_T("")主要是使相应字符串按照选项修改为 MultiByte 或者 Unicode 形态。可以制作支持 MultiByte 环境及 Unicode 环境的服务器引擎、管理模块。下面介绍一下制作引擎模块需要了解的知识——类和继承。