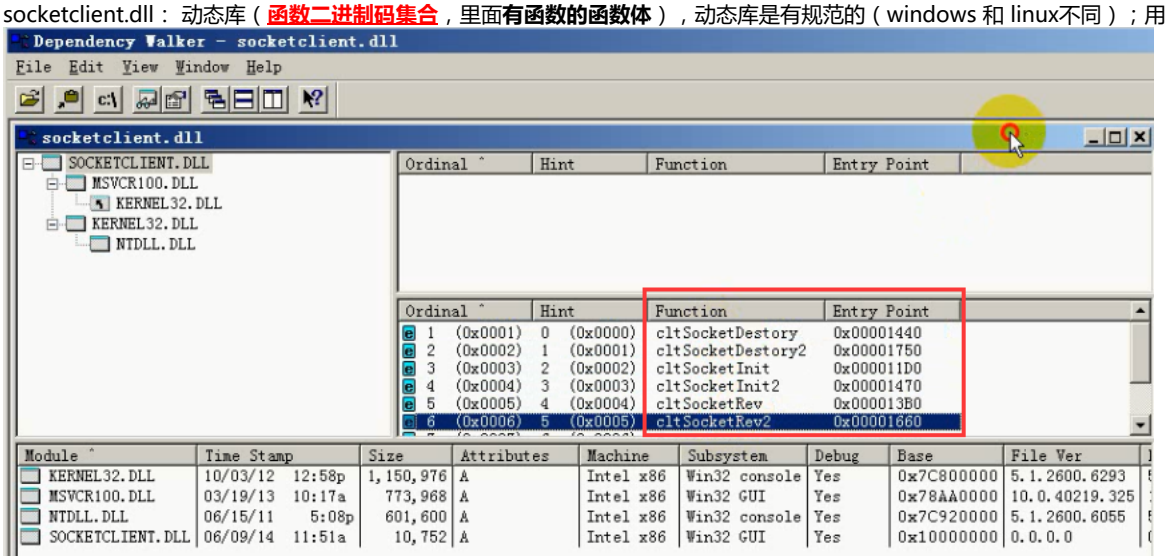
《传智播客 C语言就业班》第五讲 动态链接库dll

大的工程一般按照模块进行划分，可以把各模块封装成动态库；动态库里面其实就是一个个函数；
动态库的优点：降低耦合，便于维护（可以保持应用程序（动态库业务调用测试.c）不变，然后通过修改dll，对各业务模块内部功能进行修改）

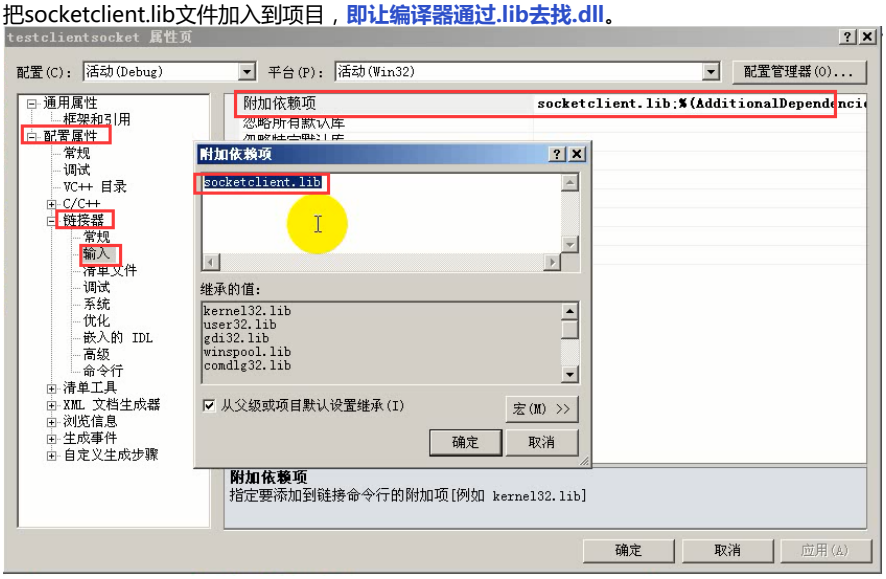
动态库部分

socketclient.lib： **是对dll资源描述文件，描述socketclient.dll有哪些函数，以及具体的函数入口地址** ；.lib来找dll，即函数的入口地址；
socketclient.dll： 动态库（**函数二进制码集合**，里面**有函数的函数体**），动态库是有规范的（windows 和 linux不同）；用Depends工具显示如下：



使用例子test.c程序来测试dll
注意： .lib .dll和test.c要放在同一目录下（不要把.lib和.dll放在Debug里面）；

把socketclient.lib文件加入到项目，**即让编译器通过.lib去找.dll**。



动态库测试框架搭建

从socketclientdll.h入手，把定义函数的.h文件 改成 调用这些函数的.c测试框架，F10调试函数被调用后，返回值是否正常。

socketclientdll.h

```
#ifndef _INC_Demo01_H
#define _INC_Demo01_H

#ifdef __cplusplus
extern "C" {
#endif

    //-----------------第一套api接口---Begin-------------------------------//
    //客户端初始化 获取handle上下
    int cltSocketInit(void **handle /*out*/);

    //客户端发报文
    int cltSocketSend(void *handle /*in*/, unsigned char *buf /*in*/,  int buflen /*in*/);

    //客户端收报文
    int cltSocketRev(void *handle /*in*/, unsigned char *buf /*in*/, int *buflen /*in out*/);

    //客户端释放资源
    int cltSocketDestory(void *handle/*in*/);
    //-----------------第一套api接口---End---------------------------------//
```

动态库业务调用测试.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "stdlib.h"
#include "string.h"
#include "stdio.h"
#include "socketclientdll.h"
void main()
{
	int ret = 0;
	void *handle = NULL;

	unsigned char buf[1024];
	int buflen = 10;
	unsigned char out[1024]; //注意：如果保证收到的消息没有烫烫烫，这里改成={0}即可
	int outlen = 0;
	strcpy(buf, "abcd123456789");
	//客户端初始化 获取handle上下
	ret = cltSocketInit(&handle/*out*/);
	if (ret !=0)
	{
		printf("func cltSocketInit() err :%d \n", ret);
		return ;
	}
	//客户端发报文
	ret = cltSocketSend(handle /*in*/, buf /*in*/, buflen /*in*/);
	if (ret != 0)
	{
		printf("func cltSocketSend() err :%d \n", ret);
		return;
	}
	//客户端收报文
	ret = cltSocketRev(handle /*in*/, out /*in*/, &outlen /*in out*/);
	if (ret != 0)
	{
		printf("func cltSocketRev() err :%d \n", ret);
		return;
	}
	printf("out:%s \n", out);
	//客户端释放资源
	ret = cltSocketDestory(handle/*in*/);
	if (ret != 0)
	{
		printf("func cltSocketDestory() err :%d \n", ret);
		return;
	}
}
```

输出：

注意：动态库的开发和设计，与普通函数的开发和设计有点不同。底层库提供的**是一种机制**，该什么功能就是什么功能，不是针对某一个人的某一种具体开发策略.....（动态库没有义务多加一个字节的'\0'，该发多长发多长！）

动态库的制作

1、做一个自己的动态库，首先按如下选项选择，之后在mysocketclient.c 打桩

（1）第一套api函数

```
/*
```

```
下面定义了一套socket客户端发送报文接受报文的api接口
请写出这套接口api的调用方法
*/
#ifndef _INC_Demo01_H
#define _INC_Demo01_H
#ifdef __cplusplus
extern "C" {
#endif

    //------------------第一套api接口---Begin--------------------------------//
    //客户端初始化 获取handle上下
    int cltSocketInit(void **handle /*out*/);

    //客户端发报文
    int cltSocketSend(void *handle /*in*/, unsigned char *buf /*in*/,  int buflen /*in*/);

    //客户端收报文
    int cltSocketRev(void *handle /*in*/, unsigned char *buf /*in*/, int *buflen /*in out*/);

    //客户端释放资源
    int cltSocketDestory(void *handle/*in*/);
    //------------------第一套api接口---End----------------------------------//
```

（2）第二套api函数

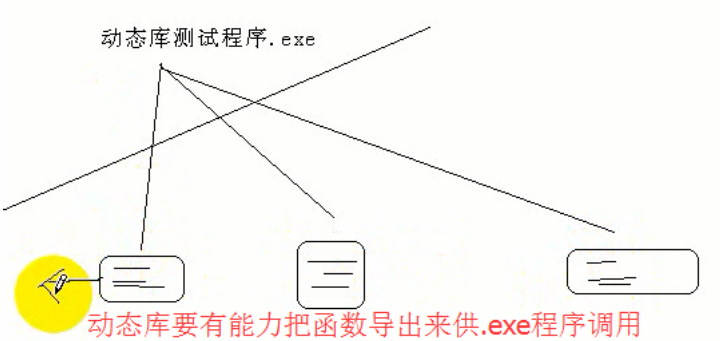
```
//------------------第二套api接口---Begin--------------------------------//
    int cltSocketInit2(void **handle);
    //客户端发报文
    int cltSocketSend2(void *handle, unsigned char *buf,  int buflen);
    //客户端收报文
    int cltSocketRev2(void *handle, unsigned char **buf, int *buflen);
    int cltSocketRev2_Free(unsigned char **buf);
    //客户端释放资源
    int cltSocketDestory2(void **handle);
//------------------第二套api接口---End----------------------------------//
```

2、动态库mysocketclient.c 内部添加return 0;

```
int cltSocketInit(void **handle /*out*/)
{
    return 0;
}

 //客户端发报文
int cltSocketSend(void *handle /*in*/, unsigned char *buf /*in*/,  int buflen /*in*/)
{
    return 0;
}

 //客户端收报文
int cltSocketRev(void *handle /*in*/, unsigned char *buf /*in*/, int *buflen /*in out*/)
{
    return 0;
}

//客户端释放资源
 int cltSocketDestory(void *handle/*in*/)
{
    return 0;
}
```

3、编写动态库.dll的代码，如printf("func cltSocketInit() begin\n")来进行测试。

注意：



注意：在每个函数实现之前添加一行代码__declspec(dllexport)**表示把函数按动态库的标准方式导出来，能让其他模块进行调用！**

```
#define  _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

__declspec(dllexport)
int cltSocketInit(void **handle /*out*/)
{
    printf("func cltSocketInit() begin\n");
    printf("func cltSocketInit() End\n");
    return 0;
}
```

在动态库里面分配内存

另注：

- **1、也可以在模块定义（DEF）文件中列出导出函数，不过这样做常常引起更多的麻烦！**
- **2、如果VS下生成了dll，却没有生成相应的lib，问题在于没有写__declspec(dllexport)。也从侧面证明了必须写__declspec(dllexport)才能把函数按动态库的标准方式导出来。**

```
//客户端发报文
__declspec(dllexport)
int cltSocketSend(void *handle /*in*/, unsigned char *buf /*in*/,  int buflen /*in*/)
{
    return 0;
}

//客户端收报文
__declspec(dllexport)
```

链接：【好文】Win32环境下动态链接库(DLL)编程原理

4.后续还要把Debug下生成的mysocketclient.lib和mysocketclient.dll放在动态库业务调用测试.c目录下，把之前的旧socketclient.lib和旧socketclient.dll删除，然后再在VS下修改链接器-输入-附加依赖项为mysocketclient.lib即可。

5、通过测试案例的工程来调试动态库的工程（打开两个VS工程，可以一边写动态库的源码，一边进行调试；调试程序中，按f11，就可以通过“动态库业务调用测试.c”这个动态库测试案例，直接调用到动态库工程mysocketclient里面的代码，**便于一边开发一边调试**！）
动态库源码生成的xxx.lib和xxx.dll（在Debug下）复制到**动态库测试框架**文件夹下（和.c在同一文件夹下而不是Debug），修改动态库测试框架项目的附加依赖项为xxx.lib。

动态库核心代码开发设计——关于socketclient

```
typedef struct _SCK_HANDLE
{
    char version[16];
    char ip[128];
    int port;
    unsigned char *buf;
    int buflen;
}SCK_HANDLE;
//动态库内部的数据类型，不想让测试程序（上层应用知道）
//数据类型的封装
//“下划线”是一种命名约定。一般表明这是内部使用的类型或变量，不希望用户使用（指库的使用者）。
```

Init()功能：在于分配内存，供上下文使用；另外返回handle。
Init()实现：

```
//客户端初始化 获取handle上下文
__declspec(dllexport)
int cltSocketInit(void **handle /*out*/)
{
    int ret = 0;
    SCK_HANDLE *sh = NULL;
    sh = (SCK_HANDLE*)malloc(sizeof(SCK_HANDLE));
    if (sh == NULL)
    {
        ret = -1;
        printf("func cltSocketInit() error: %d, malloc error...", ret);
        return ret;
    }
    memset(sh, 0, sizeof(SCK_HANDLE)); //注意使用memset初始化内存！把指针所指想的内存空间都赋成0
    strcpy(sh->ip, "192.168.0.28");
    sh->port = 88;
    *handle = sh;
    return ret;
}
```

Send()实现：

```
//客户端发报文
__declspec(dllexport)
int cltSocketSend(void *handle /*in*/, unsigned char *buf /*in*/, int buflen /*in*/)
{
    int ret = 0;
    SCK_HANDLE *sh = NULL;
    if (handle == NULL || buf == NULL)
    {
        ret = -1;
        printf("func cltSocketSend() error: %d, handle == NULL || buf == NULL...", ret);
        return ret;
    }
    sh = (SCK_HANDLE *)handle;
    sh->buf = (char*)malloc(buflen * sizeof(unsigned char));
    if (sh->buf == NULL)
    {
        ret = -2;
        printf("func cltSocketSend() error: %d, (buflen:%d)", ret, buflen);
        r
return ret;
    }
    memcpy(sh->buf, buf, buflen);
    sh->buflen = buflen;
    return ret;
}
```

注意：别人传过来的内存空间的数据有可能不是C风格字符串，而是二进制码，因此按内存块进行拷贝：memcpy(sh->buf, buf, buflen)

Recv()实现：

```
//客户端收报文
__declspec(dllexport)
int cltSocketRev(void *handle /*in*/, unsigned char *buf /*in*/, int *buflen /*in out*/)
{
    int ret = 0;
    SCK_HANDLE *sh = NULL;
    if (handle == NULL || buf == NULL || buflen == NULL)
    {
        ret = -1;
        printf("func cltSocketSend() error: %d, handle == NULL || buf == NULL...", ret);
        return ret;
    }
    sh = (SCK_HANDLE *)handle;
    memcpy(buf, sh->buf, sh->buflen);
    *buflen = sh->buflen;
    if (sh->buf != NULL)
    {
        free(sh->buf);
        sh->buf = NULL; //把状态回到原始
        sh->buflen = 0;
    }
    return ret;
}
```

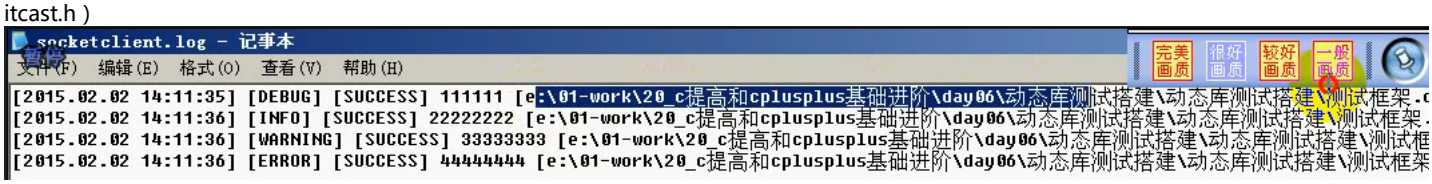
SocketDestory实现：

```
//客户端释放资源
__declspec(dllexport)
int cltSocketDestory(void *handle/*in*/)
{
    int ret = 0;
    SCK_HANDLE *sh = NULL;
    if (handle == NULL)
    {
        ret = -1;
        printf("func cltSocketDestory() error: %d", ret);
        return ret;
    }
    sh = (SCK_HANDLE *)handle;
    if (sh->buf != NULL)
    {
        free(sh->buf);
        sh->buf = NULL; //把状态回到原始
        sh->buflen = 0;
    }
    free(sh);
    return ret;
}
```

另注：
handle句柄：一般是资源、内存的首地址，或资源的代表；
handle：运行的上下文环境，即dll中API共同使用的内存空间；上面程序中，发报文时内容保存在dll的内存空间 —— 用到malloc；

动态库开发日志添加

（重点看 《video超适合自学的C++基础视频·C语言提高day6第10讲 / 传智播客第4期C语言提高第7天第14讲》 动态库_日志集成功能，itcast.c和itcast.h ）



打印日志ITCAST_LOG()
__FILE__ 编译器内置宏，编译时会把该.c文件的文件名替换__FILE__；

```
16
17  __declspec(dllexport)
18 int cltSocketInit(void **handle /*out*/)
19  {
20      int     ret = 0;
21      SCK_HANDLE *hdl = NULL;
22
23      hdl = (SCK_HANDLE *)malloc(sizeof(SCK_HANDLE));
24      if (hdl == NULL)
25      {                                   把printf替换成打印日志的函数
26          ret = -1;
27                                          printf("func cltSocketInit() err:%d \n", ret);
28          ITCAST_LOG(__FILE__, __LINE__, LogLevel[4], 0, "func cltSocketInit() err:%d \n", ret);
29          return ret;
30      }
31      memset(hdl, 0, sizeof(SCK_HANDLE)); //把指针所指向的内存空间 赋值成 0;
32
33      strcpy(hdl->ip, "192.168.6.254");
34      hdl->port = 8081;
```

注意：
1、现在Win7及以上已经不定义“WIN32”这个宏了，正确的做法是前面加下划线改成_WIN32，如下；

2、需要自己手动在c盘下新建itcast文件夹，详见itcast.c内代码，如下；

```
static int ITCAST_Error_OpenFile(int* pf)
{
    char    fileName[1024];

    memset(fileName, 0, sizeof(fileName));
#ifdef _WIN32
    sprintf(fileName, "c:\\itcast\\%s",ITCAST_DEBUG_FILE_);
#else
    //sprintf(fileName, "%s/log/%s", getenv("HOME"), ITCAST_DEBUG_FILE_);
#endif
```

结构体内套结构体的问题

```
//自定义数据类型——也是固定大小内存块的别名，是一个模子
typedef struct _AdvTeacher
{
    char name[64];
    int age;
    struct _AdvTeacher *pAdvTeacher; //正确，定义结构体变量后会分配个字节内存
    struct _AdvTeacher myAdvTeacher; //错误！结构体套结构体，无法确定内存大小
}AdvTeacher;
void main()
{
}
```

附：应用程序怎样找到DLL文件

如果应用程序使用LoadLibrary显式链接，那么在这个函数的参数中可以指定DLL文件的完整路径。如果不指定路径，或是进行隐式链接，Windows将遵循下面的搜索顺序来定位DLL：

1．包含EXE文件的目录，
2．进程的当前工作目录，
3．Windows系统目录，
4．Windows目录，
5．列在Path环境变量中的一系列目录。

这里有一个很容易发生错误的陷阱。如果你使用VC＋＋进行项目开发，并且为DLL模块专门创建了一个项目，然后将生成的DLL文件拷贝到系统目录下，从 应用程序中调用DLL模块。到目前为止，一切正常。接下来对DLL模块做了一些修改后重新生成了新的DLL文件，但你忘记将新的DLL文件拷贝到系统目录 下。下一次当你运行应用程序时，它仍加载了老版本的DLL文件，这可要当心！

动态库malloc分配的内存，必须调用动态库定义的释放函数来释放掉，不能用free直接释放，在release中会宕掉！

查内存泄漏

Linux C编程内存泄漏检测工具：**mtrace**
最简单的是**memwatch**，要在项目属性 -> C/C++ -> 预处理器 ->预处理器定义下添加两个宏MEMWATCH和MW_STDIO。
原理：通过宏定义，把系统的malloc函数改写为自己的函数，如下：
#define malloc(n) mwMalloc(n, __FILE__, __LINE__)