KCBP

（Kingdom Core Business Platform）

程序员手册

深圳市金证科技股份有限公司

二〇〇三年九月二十六日

文档信息

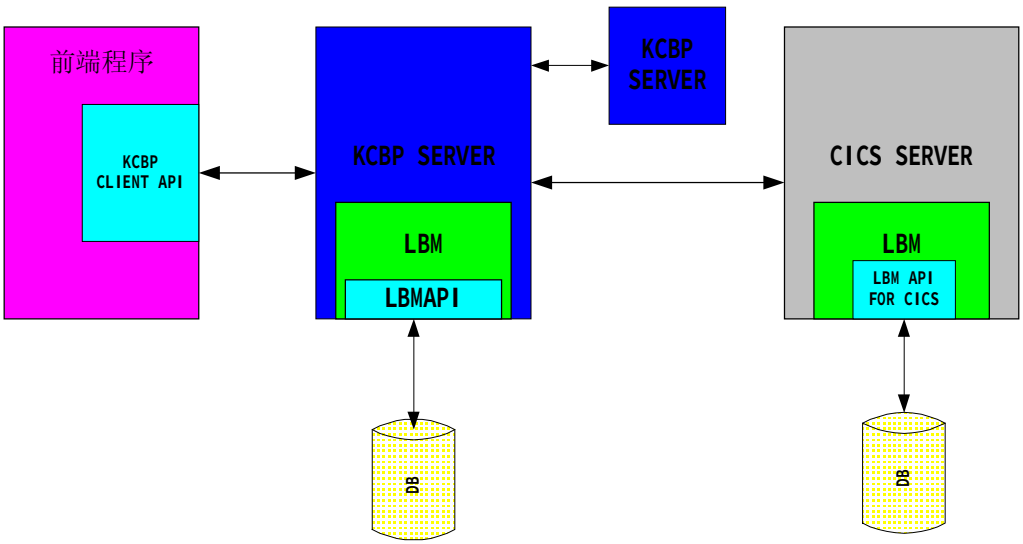
项目名称	KCBP 系统
标题	KCBP 程序员手册
类别	文档
子类别	
摘要	
当前版本	
日期	2003.9.26
作者	杜玉巍
文档拥有者	
送交人员	
文件	KCBP 程序员手册.DOC

修改历史			
版本	日期	修改人	摘要
V0.1	2002/04/01	杜玉巍	初稿
V0.5	2003/01/18	丁光博	修订
V1.0	2003/05/20	杜玉巍	1.0 版本
V1.1	2003/09/26	杜玉巍	增加发布订阅 API 说明和范例程序

目录

1. 简介

1.1 KCBP API 与 KCBP 系统



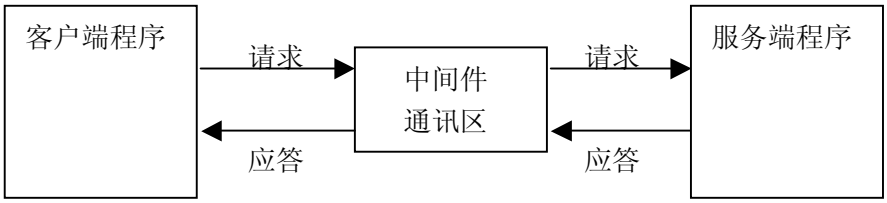
说明：

- 用户程序分为前端和后端两类，前端程序主要负责客户交互，后端程序负责事务处理。
- 前端程序通过 KCBP Client Api 提供的函数调用后端程序提供的服务功能；KCBP Client API 有两种：C 接口和 C++类接口，都以动态连接库的形式存在。KCBP CLIENT API 可以运行在 WIN 平台、AIX、LINUX 等平台。在 WIN 平台上，它的名称是 KCBPCLI.DLL。在 UNIX 平台上，它的名称是 KCBPCli.so，头文件名称是 KCBPCli.hpp 和 KCBPCli.h。
- 前端程序采用按名调用方式调用服务端服务程序，所谓按名调用，是指在 KCBP 系统中，每个服务端程序都有一个唯一的名称，通过这个名称，客户端可以通过 KCBP 系统访问服务端服务程序。
- 服务端服务程序简称 LBM(它是 Loadbale business module 的缩写)。LBM 使用 LBMAPI(KCBP 服务端 API)与 KCBP 系统进行通讯；LBM 是由 C 或 C++写的程序，它采用 ESQL/C 方式访问数据库。
- LBMAPI 以动态库形式存在，在 AIX 平台上，它的名称是 liblbmapi.so，在 WIN 平台上，它的名称是 LBMAPI.DLL。LBMAPI 的头文件名称是 Lbmapi.h。
- 为了增加后端系统的可移植性，我们还封装了 CICS 和 TUXEDO 等系统使用的 LBMAPI，这样，使用 LBMAPI 写的服务程序，可以不用修改，只需重新编译和连

接，便可以运行在多种中间件系统上。

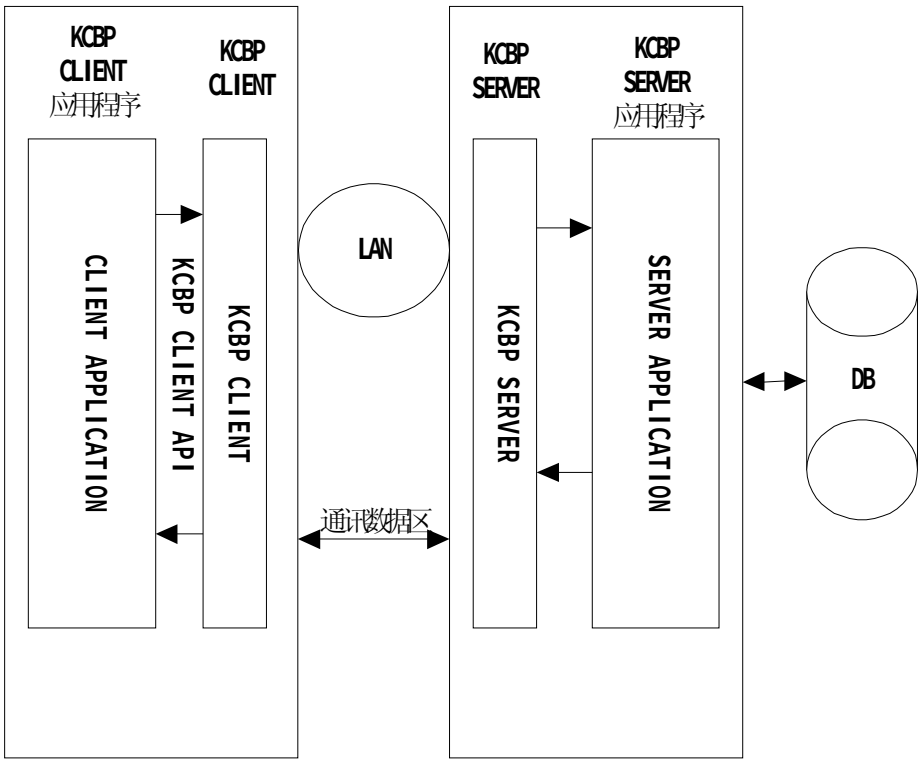
1.2 KCBP 应用程序编程思维模型

客户端程序与服务端程序通过中间件通讯区交换请求及应答，思维模型非常简单。



对应用程序开发者来讲，不用关心跨系统、跨平台、分布式等复杂操作，系统中的复杂操作由中间件系统处理。

1.3 KCBP 应用程序结构图



1.4 KCBP API 编程 HELLO WORLD!

下面是使用 KCBP API 编写的 HELLO, WORLD!例子程序。

客户端：

```
#include <stdio.h>
#include "KCBPCli.hpp"
int main()
{
	char szInfo[20];
	CKCBPCli *pKCBPCli;
	pKCBPCli =new CKCBPCli();                                  /* 实例化接口类*/
	if( pKCBPCli == NULL) return 1;
	pKCBPCli->ConnectServer( "KCBP1", "TEST", "TEST" );        /* 连接 KCBP SERVER */
	pKCBPCli->CallProgramAndCommit( "HELLO" );                 /* 调用 HELLO 服务程序 */
	pKCBPCli->GetValue( "MESSAGE",szInfo, sizeof(szInfo)-1 );  /* 取 MESSAGE */
	szInfo[sizeof(szInfo)-1]=0x0;
	delete pKCBPCli;
	printf("%s\n",szInfo);                                     /* 输出信息 */
	return 0;
}
```

服务器端：

```
#include <stdlib.h>
#include "lbmapi.h"
void* HelloWorld(void *pCA)                    /* pCA 传送通讯区参数 */
{
	LBMHANDLE hHandle;                         /* 声明 LBMHANDLE */
	if(!(hHandle= KCBP_Init(pCA)) )            /* 初始化 LBMAPI，获得 LBMHANDLE */
	{ /* 失败 */
		return NULL;
	}
	KCBP_BeginWrite(hHandle);                  /* 初始化通讯缓冲区 */
	KCBP_SetValue(hHandle, "MESSAGE","Hello, world!");   /* 设置返回信息 */
	KCBP_Exit(hHandle);                        /* LBM 结束*/
}
```

2. KCBP CLIENT API

2.1 CLIENT API 通用调用流程

1. 调用 ConnectServer 接口函数
2. 使用 SetValue 接口函数设置输入的条件(可选)
3. 调用 CallProgramAndCommit 接口函数
4. 使用 GetValue 接口函数或 RsOpen 接口函数得到输出的结果
5. 如果继续发送请求，先调用 BeginWrite 清空公共缓冲区，再重复 STEP 2-STEP5
6. 调用 DisConnect 断开连接

2.2 CLIENT API 函数分类

连接	设置通讯选项		SetConnectOption	
	查询通讯选项		GetConnectOption	
	连接		ConnectServer	
	断开		DisConnect	
服务调用	异步		暂不支持	
	同步		SERVER 端确认	**CallProgramAndCommit**
			CLIENT 端确认	**CallProgram**
				Commit
				RollBack
	订阅		KCBPCLI_Subscribe KCBPCLI_Unsubscribe KCBPCLI_ReceivePublication	
	发布		KCBPCLI_RegisterPublisher KCBPCLI_DeregisterPublisher KCBPCLI_Publish	
数据传输	初始化		BeginWrite	
	获取传输长度		GetCommLen	
	0 维表		设置值	SetValue
			获取值	GetValue
	2 维表	写	创建结果集	RsCreate
			增加结果集	RsNewTable
			增加行	RsAddRow
			设当前行各列值	RsSetCol，RsSetColByName
			保存当前行	RsSaveRow
		读	打开表、关闭表	RsOpen，RsClose
			取表名称	RsGetCursorName
			取表列名表	RsGetColNames
			取表列名	RsGetColName
			后续表查询	RsMore
			获取表行数	RsGetTableRowNum，RsGetRowNum
			获取表列数	RsGetTableColNum，RsGetColNum

			读取一行	RsFetchRow
			读当前行的列值	RsGetCol，RsGetColByName
错误处理	取错误码及错误信息		GetErr	
	取错误码		GetErrorCode	
	取错误信息		GetErrorMsg	
其它	设置调用超时		SetCliTimeOut	
	查询当前版本号		GetVersion	
SQL 风格 API 函数	连接		SQLConnect	
	断开		SQLDisconnect	
	调用服务程序		SQLExecute	
	查结果列数		SQLNumResultCols	
	取结果集名称		SQLGetCursorName	
	取列名表		SQLGetColNames	
	读一行		SQLFetch	
	查询后续结果集		SQLMoreResults	
	关闭结果集		SQLCloseCursor	
	结束交易		SQLEndTran	

2.3 CLIENT API 头文件

2.3.1 KCBPCLI.H

```
#ifndef _KCBPCLI_H
#define _KCBPCLI_H
#include <time.h>
#ifdef WIN32
    #ifdef KCBPCLI_EXPORTS
        #define KCBPCLI_API __declspec(dllexport)
    #else
        #define KCBPCLI_API __declspec(dllimport)
    #endif
    #define KCBPCLISTDCALL __stdcall /* ensure stcall calling convention on NT */
#else
    #define KCBPCLI_API
    #define KCBPCLISTDCALL      /* leave blank for other systems */
```

```
#endif


/* Transact option values  */

#define  KCBP_COMMIT   9

#define  KCBP_ROLLBACK 10


#define  UNKNOWN_ROWS  0xffffffff


#define KCBP_SERVERNAME_MAX 32

#define KCBP_DESCRIPTION_MAX32


#if (defined(KCBP_AIX) && defined(__xlC__))
    #pragma options align=packed
#else
    #pragma pack(1)
#endif


typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX+1];
    int  nProtocal;
    char szAddress[KCBP_DESCRIPTION_MAX+1];
    int  nPort;
    char szSendQName[KCBP_DESCRIPTION_MAX+1];
    char szReceiveQName[KCBP_DESCRIPTION_MAX+1];
    char szReserved[KCBP_DESCRIPTION_MAX+1];
}tagKCBPConnectOption;
```

```
#define KCBP_MSGID_LEN 32

#define KCBP_CORRID_LEN32

#define KCBP_USERID_LEN12

#define KCBP_PUBLISHERID_LEN12

#define KCBP_SERIAL_LEN 26

#define KCBP_REPLAYTO_LEN   64


#define KCBP_PSFLAG_PERSISTENT       1

#define KCBP_PSFLAG_NOPERSISTENT 2

#define KCBP_PSFLAG_REDELIVERED      4

#define KCBP_PSFLAG_NOREDELIVERED    8

#define KCBP_PSFLAG_SYSTEM       16

#define KCBP_PSFLAG_USER         32


/*****************************************************************************/

/* callback notification function definition.                                */

/*****************************************************************************/

typedef void (KCBPCLI_Callback_t)(void *);

typedef KCBPCLI_Callback_t * KCBPCLI_Notify_t;


/* control parameters to publish/subscribe queue primitives */

typedef struct

{

    int nFlags;                 /* indicates which of the values are set , inlcude type, mode,
redelivered flag*/

    char szId[KCBP_SERIAL_LEN+1];         /* pub/sub serial identifer */

    char szMsgId[KCBP_MSGID_LEN+1];       /* id of message before which to queue */

    char szCorrId[KCBP_CORRID_LEN+1];     /* correlation id used to identify message */

    int nExpiry;                /* subscribe message duration time, unit with second */
```

```
    int nPriority;              /* publish priority */
    time_t tTimeStamp;          /* pub/sub timestamp*/
    KCBPCLI_Notify_t lpfnCallback;       /* callback function pointer*/
}tagKCBPPSControl;

#if (defined(KCBP_AIX) && defined(__xlC__))
    #pragma options align=reset
#elif defined(KCBP_SOL)
    #pragma pack(4)
#else
    #pragma pack()
#endif

typedef void * KCBPCLIHANDLE;

#ifdef __cplusplus
extern "C" {
#endif

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Init(KCBPCLIHANDLE * hHandle) ;
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Exit(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetVersion(KCBPCLIHANDLE hHandle, int *pnVersion) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SetConnectOption(KCBPCLIHANDLE hHandle,
tagKCBPConnectOption stKCBPConnection) ;
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetConnectOption(KCBPCLIHANDLE hHandle,
tagKCBPConnectOption *pstKCBPConnection) ;
```

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_ConnectServer(KCBPCLIHANDLE hHandle, char *ServerName, char *UserName, char *Password ) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_DisConnect(KCBPCLIHANDLE hHandle) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_BeginWrite(KCBPCLIHANDLE hHandle) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_CallProgramAndCommit(KCBPCLIHANDLE hHandle, char *ProgramName) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_CallProgram(KCBPCLIHANDLE hHandle, char *ProgramName) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Commit(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RollBack(KCBPCLIHANDLE hHandle) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetValue(KCBPCLIHANDLE hHandle, char *KeyName, char *Vlu, int Len) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SetValue(KCBPCLIHANDLE hHandle, char *KeyName, char *Vlu ) ;


/*rs*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsCreate(KCBPCLIHANDLE hHandle, char *Name, int ColNum,char *pColInfo) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsNewTable(KCBPCLIHANDLE hHandle, char *Name, int ColNum,char *pColInfo) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsAddRow(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsSaveRow(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsSetCol(KCBPCLIHANDLE hHandle, int Col, char *Vlu) ;

/*KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsSetCol(KCBPCLIHANDLE hHandle, char *Name, char *Vlu) ;*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsSetColByName(KCBPCLIHANDLE hHandle, char *Name, char *Vlu) ;
```

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsOpen(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsMore(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsClose(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetCursorName(KCBPCLIHANDLE hHandle, char *
pszCursorName, int nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetColNames(KCBPCLIHANDLE hHandle, char *pszInfo, int
nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetColName(KCBPCLIHANDLE hHandle, int nCol, char *pszName,
int nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsFetchRow(KCBPCLIHANDLE hHandle) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetCol(KCBPCLIHANDLE hHandle, int Col, char *Vlu) ;

/*KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetCol(KCBPCLIHANDLE hHandle, char *KeyName, char
*Vlu) ;*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetColByName(KCBPCLIHANDLE hHandle, char *KeyName, char
*Vlu) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetRowNum(KCBPCLIHANDLE hHandle, int *pnRows ) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetColNum(KCBPCLIHANDLE hHandle, int *pnCols ) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetTableRowNum(KCBPCLIHANDLE hHandle, int nt, int
*pnRows) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetTableColNum(KCBPCLIHANDLE hHandle, int nt, int
*pnCols) ;


/*misc*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetErr(KCBPCLIHANDLE hHandle, int *pErrCode,char *ErrMsg) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetErrorCode(KCBPCLIHANDLE hHandle, int *pnErrno) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetErrorMsg(KCBPCLIHANDLE hHandle, char *szError) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetCommLen(KCBPCLIHANDLE hHandle, int *pnLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SetCliTimeOut(KCBPCLIHANDLE hHandle, int TimeOut) ;
```

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SetTrace(KCBPCLIHANDLE hHandle, int nMode) ;


/*SQL-Liked*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLConnect(KCBPCLIHANDLE hHandle, char *ServerName, char *UserName, char *Password) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLDisconnect(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLExecute(KCBPCLIHANDLE hHandle, char * szProgramName) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLNumResultCols(KCBPCLIHANDLE hHandle, int *pnresultcols) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLGetCursorName(KCBPCLIHANDLE hHandle, char * pszCursorName, int nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLGetColNames(KCBPCLIHANDLE hHandle, char * szTableInfo, int nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLGetColName(KCBPCLIHANDLE hHandle, int nCol, char * szTableInfo, int nLen) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLFetch(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLMoreResults(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLCloseCursor(KCBPCLIHANDLE hHandle) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SQLEndTran(KCBPCLIHANDLE hHandle, int nType ) ;


/*pub/sub*/

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Subscribe(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszTopicExpr, char *pszFilterExpr) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Unsubscribe(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_ReceivePublication(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszData, int nDataLen) ;


KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RegisterPublisher(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszTopicExpr) ;

KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_DeregisterPublisher(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl) ;
```

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Publish(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl,
char *pszTopicExpr, char *pszData, int nDataLen) ;

#ifdef __cplusplus

}

#endif

#endif
```

2.3.2 KCBPCLI.HPP

```
#ifndef _KCBPCLI_HPP
#define _KCBPCLI_HPP

#include "KCBPCli.h"

class KCBPCLI_API CKCBPCli
{
public:
    void    *m_pKCBPCliBase;
public:
    CKCBPCli(void);
    ~CKCBPCli();

    int GetVersion(int *pnVersion);

    int SetConnectOption( tagKCBPConnectOption stKCBPConnection) ;
    int GetConnectOption( tagKCBPConnectOption *pstKCBPConnection) ;

    int ConnectServer(char *ServerName, char *UserName, char *Password );
    int DisConnect();

    int BeginWrite();

    int CallProgramAndCommit(char *ProgramName);

    int CallProgram(char *ProgramName);
    int Commit();
    int RollBack();

    int GetValue( char *KeyName, char *Vlu, int Len=0 );
    int SetValue( char *KeyName, char *Vlu );
```

```
/*rs*/
int RsCreate(char *Name, int ColNum,char *pColInfo);
int RsNewTable(char *Name, int ColNum,char *pColInfo);

int RsAddRow();
int RsSaveRow();
int RsSetCol(int Col, char *Vlu);
int RsSetCol(char *Name, char *Vlu);
int RsSetColByName(char *Name, char *Vlu);

int RsOpen();
int RsMore();
int RsClose();
int RsGetCursorName(char * pszCursorName, int nLen);
int RsGetColNames(char *pszInfo, int nLen);
int RsGetColName(int nCol, char *pszName, int nLen);
int RsFetchRow();

int RsGetCol(int Col, char *Vlu);
int RsGetCol(char *KeyName, char *Vlu);
int RsGetColByName(char *KeyName, char *Vlu);

int RsGetRowNum(int *pnRows );
int RsGetColNum(int *pnCols );
int RsGetTableRowNum(int nt, int *pnRows);
int RsGetTableColNum(int nt, int *pnCols);

/*misc*/
int GetErr(int *pErrCode,char *ErrMsg);
int GetErrorCode(int *pnErrno);
int GetErrorMsg(char *szError);

int GetCommLen(int *pnLen);
int SetCliTimeOut(int TimeOut);

/*SQL-Liked*/
int SQLConnect(char *ServerName, char *UserName, char *Password);
int SQLDisconnect();
int SQLExecute(char * szProgramName);
int SQLNumResultCols(int *pnresultcols);
int SQLGetCursorName(char * pszCursorName, int nLen);
int SQLGetColNames(char * szTableInfo, int nLen);
int SQLGetColName(int nCol, char * szTableInfo, int nLen);
int SQLFetch();
```

```
    int SQLMoreResults();
    int SQLCloseCursor();
    int SQLEndTran(int nType );
};
#endif
```

2.4 KCBP CLIENT API 函数说明

注意：

- **API** 描述主要针对 **C++**接口，另外，也给出 **C** 独有接口的描述。**C++**接口具有基本功能，**C** 接口比 **C++**接口的功能丰富，比如 **C** 接口具有发布/订阅 **API**。
- **INPUT** 表示输入参数，**OUTPUT** 表示输出参数，**INPUT/OUTPUT** 既是输入又是输出，**IGNORE** 表示该项无实际意义，**Reserved** 表示该项用法保留。
- 函数返回值为 **0** 时，表示调用成功，否则返回值为错误代码。
- **C++**接口和 **C** 接口混用时，**C++**接口实例指针可以作为 **C** 接口的 **Handle** 参数。

2.4.1 设置连接选项

函数原型： int SetConnectOption(tagKCBPConnectOption stKCBPConnection) ;

输入参数：

参数名称	参数说明	用法
tagKCBPConnectOption stKCBPConnection	连接选项结构	Input

返　　回：0 表示成功，其它失败，返回信息由返回代码决定。

返回码定义：

用法说明：

调用 ConnectServer 时，API 按照 ServerName 查找服务端的连接参数，当用户设置了连接选项时，客户端使用该选项中的参数连接服务端，否则在 KCBPCli.ini 中查找参数。

连接选项结构定义如下：

```
#if defined(KCBP_AIX)
#pragma options align=packed
#else
#pragma pack(push, 1)
#endif

typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX+1];/*用户自定义的 KCBP 服务器名称*/
    int    nProtocal;       /*协议类型，0 表示使用 TCP*/
```

```
    char szAddress[KCBP_DESCRIPTION_MAX+1]; /*服务端 IP*/
    int    nPort;/*服务端端口号*/
    char szSendQName[KCBP_DESCRIPTION_MAX+1];/*发送队列名称，由服务端指定*/
    char szReceiveQName[KCBP_DESCRIPTION_MAX+1];/*接收队列名称，由服务端指定*/
    char szReserved[KCBP_DESCRIPTION_MAX+1];/*保留*/
}tagKCBPConnectOption;

#if defined(KCBP_AIX)
#pragma options align=reset
#else
#pragma pack(pop)
#endif
```

在 AIX 上，编译时注意用宏 KCBP_AIX 对齐结构。

例子：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "KCBPCli.hpp"
int main(int argc,char *argv[])
{
    CKCBPCli *pKCBPCli=new CKCBPCli();
    if(!pKCBPCli) return 1;

    tagKCBPConnectOption stKCBPConnection;
    memset(&stKCBPConnection, 0 , sizeof(stKCBPConnection));
    strcpy(stKCBPConnection.szServerName, "KCBP01");
    stKCBPConnection.nProtocal = 0;
    strcpy(stKCBPConnection.szAddress, "192.168.54.2");
    stKCBPConnection.nPort = 22000;
    strcpy(stKCBPConnection.szSendQName, "req1");
    strcpy(stKCBPConnection.szReceiveQName, "ans1");
    if(pKCBPCli->SetConnectOption( stKCBPConnection ) )
    {
        delete pKCBPCli;
        return 2;
    }
    if(pKCBPCli->SQLConnect("KCBP01","KCXP00","888888"))
    {
        delete pKCBPCli;
        return 3;
    }
```

```
    pKCBPCli->SQLDisconnect();
    delete pKCBPCli;
    return 0;
}
```

2.4.2 查询连接选项

函数原型：int GetConnectOption(tagKCBPConnectOption *pstKCBPConnection) ;
输入参数：

参数名称	参数说明	用法
tagKCBPConnectOption *pstKCBPConnection	返回的连接选项结构	Output

返　　回：0 表示成功，其它失败，返回信息由返回代码决定。
返回码定义：
用法说明：

2.4.3 连接 KCBP SERVER

函数原型：int **ConnectServer**(char *ServerName, char *UserName, char *Password);
输入参数：

参数名称	参数说明	用法
char *ServerName	KCBP 节点名	Input
char *UserName	KCBP 用户号	Input
char *Password	KCBP 密码（密文）	Input

返　　回：0 表示成功，其它失败，返回信息由返回代码决定。
返回码定义：
用法说明：

2.4.4 断开与 KCBP SERVER 的连接

函数原型：int DisConnect()
输入参数：无
返　　回：0 成功，其它失败，返回信息由返回代码决定。
用法说明：

2.4.5 同步调用一个服务程序（提交方式）

函数原型：int **CallProgramAndCommit**(char *prg);
参数说明：

参数名称	参数说明	用法
char *prg	服务程序名	Input

返　　回：返回 **0** 表示成功，其它失败。
用法说明：客户端程序等待此服务程序返回后再继续执行。

2.4.6 同步调用一个服务程序（不提交方式）

函数原型：int **CallProgram** (char *prg);
参数说明：

参数名称	参数说明	用法
char *prg	服务程序名	Input

返　　回：返回 **0** 表示成功，其它失败。
用法说明：同步调用另一个服务程序，Server 端不提交，需要由客户程序调用 Commit 提交。参数 prg 为该服务程序名称，等待此服务程序返回后再继续执行。

2.4.7 提交事务

函数原型：int **Commit**();
参数说明：无
返　　回：返回 **0** 表示成功，其它失败。
用法说明：只适用于 CallProgram。

2.4.8 回滚事务

函数原型：int **RollBack**();
参数说明：无
返　　回：返回 **0** 表示成功，其它失败。
用法说明：只适用于 CallProgram。

2.4.9 订阅

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Subscribe(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszTopicExpr, char *pszFilterExpr) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagKCBPPSControl *pstPSCtl	控制项	Input/Output
char *pszTopicExpr	订阅主题	Input
char *pszFilterExpr	过滤条件，保留	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Output
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Output
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Output
int nExpiry	订期，以秒为单位	Input
int nPriority	消息的优先级别	Ignore

time_t tTimeStamp	受理时间	Output
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针	Reserved

返　　回：返回 **0** 表示成功，**100** 表示订阅已受理，但尚无该主题，其它失败。
用法说明：订阅消息。

2.4.10 查询订阅消息

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_ReceivePublication(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszData, int nDataLen) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagKCBPPSControl *pstPSCtl	控制项	Input
char *pszData	存放消息的数据区	Output
int nDataLen	数据区长度	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Input
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Input
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Input
int nExpiry	查询超时，以秒为单位	Input
int nPriority	消息的优先级别	Ignore
time_t tTimeStamp	受理时间	Ignore
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针，保留	Ignore

返　　回：返回 **0** 表示受到一条消息，该消息存放在 **pszData** 中，其它失败。
用法说明：查询是否收到订阅消息。

2.4.11 取消订阅

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Unsubscribe(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagKCBPPSControl *pstPSCtl	控制项	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Input
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Ignore

Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Ignore
int nExpiry	订期，以秒为单位	Ignore
int nPriority	消息的优先级别	Ignore
time_t tTimeStamp	受理时间	Ignore
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针	Ignore

返　　回：返回 **0** 表示成功，其它失败。
用法说明：取消订阅消息。

2.4.12 登记发布主题

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RegisterPublisher(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszTopicExpr) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagKCBPPSControl *pstPSCtl	控制项	Input/Output
char *pszTopicExpr	发布主题	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Output
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Ignore
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Ignore
int nExpiry	有效期，以秒为单位	Input
int nPriority	消息的优先级别	Ignore
time_t tTimeStamp	受理时间	Output
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针	Reserved

返　　回：返回 **0** 表示成功，其它失败。
用法说明：声明发布主题。

2.4.13 发布

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Publish(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl, char *pszTopicExpr, char *pszData, int nDataLen) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagKCBPPSControl *pstPSCtl	控制项	Input
char *pszTopicExpr	发布主题	Input
char *pszData	存放消息的数据区	Input
int nDataLen	数据区长度,小于 32K	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Input
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Ignore
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Ignore
int nExpiry	有效期，以秒为单位	Input
int nPriority	消息的优先级别	Input
time_t tTimeStamp	受理时间	Ignore
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针	Reserved

返　　回：返回 **0** 表示成功，其它失败。
用法说明：按主题发布消息。

2.4.14 取消发布主题

函数原型：
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_DeregisterPublisher(KCBPCLIHANDLE hHandle, tagKCBPPSControl *pstPSCtl) ;
参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
TagKCBPPSControl *pstPSCtl	控制项	Input

tagKCBPPSControl *pstPSCtl 各项参数说明：

参数名称	参数说明	用法
int nFlags	标志	Reserved
char szId[KCBP_SERIAL_LEN+1]	受理号	Input
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Ignore
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号	Ignore
int nExpiry	有效期，以秒为单位	Ignore
int nPriority	消息的优先级别	Ignore
time_t tTimeStamp	受理时间	Ignore
KCBPCLI_Notify_t lpfnCallback	消息处理函数指针	Reserved

返　　回：返回 **0** 表示成功，其它失败。
用法说明：取消发布主题。

2.4.15 清除公共数据区

函数原型：int **BeginWrite**();
参数说明：无
返　　回：0 成功，其它失败
用法说明：表示开始写通信用的公共数据区，它的真正作用是清除该公共数据区。注意，如果重新开始写通信用的公共数据区(比如在错误处理时)，应该再次调用 BeginWrite 函数，这样

可以清除原来的内容。

2.4.16 根据键名(KeyName)设置键值(Value)

函数原型：int **SetValue**(char *KeyName, char *Value);

输入参数：

参数名称	参数说明	用法
char *KeyName	关键字	Input
char *Value	关键字值	Input

返　　回：0 成功，其它失败

用法说明：

此函数通过其参数 KeyName 指定的关键字来存储字符串值 Value，可以通过 GetValue 函数并使用相同的关键字来获取设置的字符串值。

如果 KeyName 为空字符串，则设置整个公共数据区代表的字符串。

关键字 KeyName 是任意定义的，可以在程序规划时确定，或由服务程序的程序员和客户程序的程序员事先约定。

SetValue 函数和 GetValue 函数是 KCBP 用于传递单值(0 维结构)的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。注意，传递的值只能是字符串。

SetValue 函数最好在 RsCreate 函数之前使用。

2.4.17 根据键名(KeyName)取键值(Value)

函数原型：int GetValue(char *KeyName, char *Value, int nLen=0);

输入参数：

参数名称	参数说明	用法
Char *KeyName	关键字	Input
Char *Value	关键字值	Input
Int nLen	Value 缓冲区长度	Input

返　　回：0 成功，其它失败

用法说明：

根据键名(KeyName)取键值(Vlu)。

此函数通过其参数 KeyName 指定的关键字来获取通过 SetValue 函数来设置的字符串值。

如果 GetValue 的参数指定的关键字并没有值，则返回空字符串。

如果 KeyName 为空字符串，则返回整个公共数据区代表的字符串。

关键字 KeyName 是任意定义的，可以在程序规划时确定，或由服务程序的程序员和客户程序的程序员事先约定。

SetValue 函数和 GetValue 函数是 KCBP 用于传递单值(0 维结构)的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。注意，传递的值只能是字符串。

GetValue 函数最好在 RsOpen 函数之前使用。

注意，如果不设置 nLen，调用 GetValue 函数容易引起 C 的越界错误。nLen 可以限定获取不超过指定长度的字符串。

2.4.18 创建结果集

函数原型：int RsCreate(char *Name, int ColNum,char *pColInfo);

输入参数：

参数名称	参数说明	用法
Char *Name	结果集名称	Input
int ColNum	指定结果集的列数	Input
Char *pColInfo	结果集列名称表	Input

返　　回：0 成功，其它失败

用法说明：

RsCreate 函数和 RsOpen 函数是 KCBP 传递二维结构的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。

如果是客户机到服务器方向，数据量总长建议不要超过 32K，每行不要超过 4K。

RsCreate 用于创建第 1 个结果集。

结果集列名称表格式如”col1,col2,col3”。

2.4.19 增加结果集

函数原型：int RsNewTable(char *Name, int ColNum,char *pColInfo);

输入参数：

参数名称	参数说明	用法
Char *Name	结果集名称	Input
int ColNum	指定结果集的列数	Input
Char *pColInfo	结果集列名称表	Input

返　　回：0 成功，其它失败

用法说明：

RsCreate 用于创建第 1 个结果集，RsNewTable 用于增加后续结果集。

2.4.20 使公共数据区的结果集增加一行

函数原型：int **RsAddRow**();

输入参数：无

返　　回：0 成功，其它失败

用法说明：

2.4.21 根据列号设置结果集中当前行的某一列值

函数原型：int **RsSetCol**(int Col, char *Vlu);

参数说明：

参数名称	参数说明	用法
int Col	列序号，从 1 开始编号	Input
char *Vlu	列值	Input

返　　回：0 成功，其它失败

用法说明：

2.4.22 根据列名设置结果集中当前行的某一列值

函数原型：int RsSetCol(char *Name, char *Vlu);

int RsSetColByName(char *Name, char *Vlu);

参数说明：

参数名称	参数说明	用法
char *Name	列名称	Input
char *Vlu	列值	Input

返　　回：0 成功，其它失败

用法说明：

列名称在 RsCreate 及 RsNewTable 时通过列名表设置。

RsSetCol、RsSetColByName 两个函数做用相同。RsSetColByName 便于 C 程序的移植。

2.4.23 在公共数据区的结果集中存储当前行

函数原型：int **RsSaveRow**();

输入参数：无

返　　回：0 成功，其它失败

用法说明：

2.4.24 打开结果集

函数原型：int **RsOpen**();

输入参数：无

返　　回：

0 表示打开成功，并且可以确定结果集的行数；

100 表示打开成功，但不能确定结果集的行数，KCBP 大查询采用该种方式返回数据；

其它失败。

用法说明：

返回值 0 时，可以用 RsGetTableColNum、RsGetTableRowNum 等函数取表的行、列数。

返回值 100 时，结果集的行数不确定。不能用 RsGetTableColNum、RsGetTableRowNum 等函数取表的行、列数。这时，可以用 RsGetColNum 取当前结果集列数。如需确定是否有后续结果集，需要用 RsMore 查询。

2.4.25 获取当前结果集名称

函数原型：int RsGetCursorName(char * pszCursorName, int nLen);

参数说明：

参数名称	参数说明	用法
PszCursorName	结果集名称	output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，名称放在 pszCursorName 中；其它失败。

用法说明：结果集名称是在 RsCreate 或 RsNewTable 时设定的。

2.4.26 获取当前结果集的全部列名称

函数原型：int RsGetColNames(char *pszInfo, int nLen);

参数说明：

参数名称	参数说明	用法
PszInfo	列名表	output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，结果集列名称表放在 pszInfo 中；其它失败。

用法说明：结果列名表是在 RsCreate 或 RsNewTable 时设定的。

2.4.27 获取当前结果指定列的名称

函数原型：int RsGetColName(int nColIndex, char *pszInfo, int nLen);

参数说明：

参数名称	参数说明	用法
NcolIndex	列序号	Input
PszInfo	列名	output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，列名放在 pszInfo 中；其它失败。

用法说明：结果列名是在 RsCreate 或 RsNewTable 时通过列名表设定的。

2.4.28 获取公共数据区的当前结果集的行数

函数原型：int RsGetRowNum(int *nRows);

参数说明：

参数名称	参数说明	用法
int *nRows	结果集行数指针	output

返　　回：0 成功，行数放在 nRows 中；其它失败。

用法说明：

2.4.29 获取公共数据区的当前结果集的列数

函数原型：int RsGetColNum(int *nCols);

参数说明：

参数名称	参数说明	用法
Int *nCols	结果集列数	output

返　　回：0 成功，列数放在 nCols 中；其它失败。

用法说明：

2.4.30 获取公共数据区的指定结果集的行数

函数原型：int RsGetTableRowNum(int nt, int *nRows);

参数说明：

参数名称	参数说明	用法
Int nt	结果集编号，从 1 开始	input
int *nRows	结果集行数	output

返　　回：0 成功，行数放在 nRows 中；其它失败。

用法说明：

2.4.31 获取公共数据区的指定结果集的列数

函数原型：int RsGetTableColNum(int nt, int *nCols);
参数说明：

参数名称	参数说明	用法
Int nt	结果集编号，从 1 开始	input
int *nCols	结果集列数	output

返　　回：0 成功，列数放在 nCols 中；其它失败。
用法说明：

2.4.32 从公共数据区的结果集中依序获取一行，作为当前行

函数原型：int **RsFetchRow**();
输入参数：无
返　　回：0 成功，其它失败
用法说明：

2.4.33 从结果集的当前行的某一列取值（根据列序号）

函数原型：int **RsGetCol**(int Col, char *Vlu);
参数说明：

参数名称	参数说明	用法
int Col	列序号，从 1 开始编号	Input
char *Vlu	列值，指向存储列值的缓冲区	Output

返　　回：0 成功，其它失败
用法说明：

2.4.34 从结果集的当前行的某一列取值(根据列名)

函数原型：int **RsGetCol** (char *ColName, char *Vlu);
int **RsGetColByName**(char *ColName, char *Vlu);
参数说明：

参数名称	参数说明	用法
const char *ColName	列名，由服务程序的 RsSetColNameList 函数设置。	Input
char *Vlu	列值，指向存储列值的缓冲区	Output

返　　回：0 成功，其它失败
用法说明：RsGetColByName 与 RsGetCol 功能相同。RsGetColByName 函数便于 C 程序移植。

2.4.35 查询后续结果集

函数原型：int **RsMore**();

输入参数：无
返　　回：0 有后续结果集，其它无
用法说明：
用 RsOpen 结果集后，如果返回 100，说明是一个大查询，这时重复调用 RsFetchRow，当 RsFetchRow 返回非 0 时，意味着当前结果集已经结束，这时需要用 RsMore 查询是否有后续结果集，如果有，继续 RsFetchRow 操作。

2.4.36 关闭结果集

函数原型： int RsClose();
参数说明：无
返　　回：0 成功，其它失败。
用法说明：RsClose 通知 Server 端停止发送结果集数据并清除结果集占用的存储资源。

2.4.37 获取公共数据区当前长度

函数原型：int **GetCommLen**(int *nLen);
参数说明：无
返　　回：0 成功，公共数据区当前长度放在 nLen 中；其它失败。
用法说明：

2.4.38 设置调用超时时间

函数原型：int SetCliTimeOut(int TimeOut);
参数说明：

参数名称	参数说明	用法
int TimeOut	调用超时，单位秒	Input

返　　回：0 成功，其它失败。
用法说明：

2.4.39 返回错误码和错误信息

函数原型：int GetErr(int *ErrCode,char *ErrMsg);
参数说明：

参数名称	参数说明	用法
int *ErrCode	错误码	Output
char *ErrMsg	错误信息	Output

返　　回：0 成功，其它失败
用法说明：此调用用于同步或异步调用失败时，返回错误信息，该种失败是由 KCBP 返回的，与业务程序无关。

2.4.40 返回错误码

函数原型：int GetErrCode(int *ErrCode);

参数说明：

参数名称	参数说明	用法
int *ErrCode	错误码	Output

返　　回：0 成功，其它失败

用法说明：此调用用于同步或异步调用失败时，返回错误信息，该种失败是由 KCBP 返回的，与业务程序无关。ErrCode 为 0 表示无错误。

2.4.41 返回错误信息

函数原型：int GetErrorMsg(char *szError);

参数说明：

参数名称	参数说明	用法
char *ErrMsg	错误信息	Output

返　　回：0 成功，其它失败。

用法说明：此调用用于同步或异步调用失败时，返回错误信息，该种失败是由 KCBP 返回的，与业务程序无关。返回空串无错误信息。

2.4.42 查询 API 当前版本号

函数原型：int GetVersion(int *pnVersion);

参数说明：

参数名称	参数说明	用法
int *pnVersion	版本号	Output

返　　回：0 成功，其它失败。

用法说明：

2.4.43 SQL 风格函数

SQLConnect 与 ConnectServer 相同；
SQLDisconnect 与 Disconnect 相同；
SQLExecute 与 CallProgramAndCommit 相似；
SQLNumResultCols 与 RsGetColNum 相同；
SQLGetCursorName 与 RsGetCursorName 相同；
SQLGetColNames 与 RsGetColNames 相同；
SQLGetColName 与 RsGetColName 相同；
SQLFetch 与 RsFetchRow 相同;
SQLMoreResults 与 RsMore 相同;
SQLCloseCursor 与 RsClose 相同
SQLEndTran 包含了 Commit 和 Rollback 的操作，通过参数控制。

注意：SQL 风格函数有以下几点特殊之处：

- 没有 RsOpen，SQLNumResultCols 隐含打开结果集。
- SQLEndTran 包含了 Commit 和 Rollback 的操作，通过参数控制。

3. KCBP 服务端编程接口 LBMAPI

LBMAPI 是 KCBP 服务端提供的 C 语言调用接口。
业务程序是 LBM，它用 ESQL/C 方式编写，它通过 LBMAPI 访问 KCBP。
LBMAPI 通过 void *pCA 与 KCBP 传递通讯信息。
一般来讲，LBMAPI 调用成功返回 0，失败返回非 0。

3.1 服务端程序调用流程

3.1.1 服务端程序通用流程

1. 调用 **KCBP_Init** 初始化
2. 获取输入参数
3. 业务处理**(ESQL/C** 方式访问数据库**)**
4. 返回结果
5. 调用 **KCBP_Exit** 退出程序

3.1.2 获得输入参数

1. 获取 **0** 维表形式的输入参数**:**使用 **KCBP_GetValue()**逐个获取参数
2. 获取 **2** 维表形式的输入参数**:**先用 **KCBP_RsOpen()** 打开输入结果集**,**再用 **KCBP_RsFetchRow()**将请求逐行读出**,**每读出 **1** 行，再用 **KCBP_RsGetCol()**获取各个输入参数。

3.1.3 返回结果

3.1.3.1 使用 0 维表返回结果

1. 使用 **KCBP_GetValue** 函数获取输入的条件
2. 调用 **KCBP_BeginWrite** 清空输出缓冲区
3. 使用 **KCBP_SetValue** 函数
4. 重复步骤 **3**，直到设置完成所有的输出结果

3.1.3.2 使用 2 维表返回结果

1. 使用 **KCBP_GetValue** 函数获取输入的条件
2. 重复 **Step1** 直到获得全部输入参数

3. 调用 **KCBP_BeginWrite** 函数清空通讯缓冲区
4. 使用 **KCBP_RsCreate()**函数创建第 **1** 个结果集
5. 使用 **KCBP_RsAddRow()**增加结果列数
6. 使用 **KCBP_RsSetCol()**设置列内容
7. 使用 **KCBP_RsSaveRow()**保存结果集到缓冲区
8. 重复步骤 **5-7**，直到设置完成第该结果集
9. 如果有后续结果集，使用 **KCBP_NewTable()**函数创建后续结果集，否则退出
10. 重复步骤 **8**

3.2 LBMAPI 分类

初始化、退出	初始化		KCBP_Init	
	结束		KCBP_Exit	
	终止		KCBP_Abort	
数据传输	初始化		KCBP_BeginWrite	
	获取传输长度		KCBP_GetCommLen	
	0 维表		设置值	KCBP_SetValue
			获取值	KCBP_GetValue KCBP_GetValueN
	2 维表	写	创建结果集	KCBP_RsCreate
			增加结果集	KCBP_RsNewTable
			增加行	KCBP_RsAddRow
			设当前行各列值	KCBP_RsSetCol ， KCBP_RsSetColByName
			保存当前行	KCBP_RsSaveRow
		读	打开表	KCBP_RsOpen
			关闭表	KCBP_RsClose
			取表名称	KCBP_RsGetCursorName
			取表列名表	KCBP_RsGetColNames
			取表列名	KCBP_RsGetColName
			后续表查询	KCBP_RsMore
			获取表行数	KCBP_RsGetTableRowNum KCBP_RsGetRowNum
			获取表列数	KCBP_RsGetTableColNum KCBP_RsGetColNum
			读取一行	KCBP_RsFetchRow
			读当前行的列值	KCBP_RsGetCol KCBP_RsGetColN KCBP_RsGetColByName KCBP_RsGetColByNameN
服务调用	异步	暂不支持		
	同步	系统内	单个服务调用	**KCBP_CallProgram**
			多个服务调用	**KCBP_CallProgramExt**
		系统间	单个服务调用	**KCBP_CallProgramSys**
			多个服务调用	**KCBP_CallProgramSysExt**
事务控制	提交事务		**KCBP_Commit**	
	回滚事务		**KCBP_RollBack**	
	异常中止事务		**KCBP_Abend**	
内存操作	分配内存		KCBP_Malloc	

	分配局部内存	KCBP_MallocLocal
	分配共享内存	KCBP_MallocShared
	释放内存	KCBP_Free
公共数据区操作	写公共数据区	KCBP_SaveToCwa
	读公共数据区	KCBP_LoadFromCwa
	公共数据区加锁	KCBP_LockCwa
	公共数据区解锁	KCBP_UnLockCwa
其它	输出调试信息	KCBP_PrintStatus
	读 KCBP 系统参数	KCBP_GetSystemParam
	安全 Sleep	KCBP_Sleep

3.3 LBMAPI 头文件

下面服务端编程接口的头文件 Lbmapi.h：

```
/*
KCBP LBM API for C, Version 1.0
Copyright (c) 2002 SZKingdom Corp. All rights reserved.
Original by Mr. Yuwei Du, 20020611
*/

#ifndef _LBMAPI_H
#define _LBMAPI_H

#if defined (WIN32)
    #define LBMSTDCALL __stdcall      /* ensure stcall calling convention on NT */
#else
    #define LBMSTDCALL                  /* leave blank for other systems */
#endif

#ifndef __cplusplus
    #define LBMSTDCALL
    typedef int _bool;
    #define _false 0
    #define _true 1
    #ifdef false
        #undef false
    #endif
    #ifdef true
        #undef true
    #endif
    #define bool _bool
    #define false _false
    #define true _true
```

```
#endif

#ifdef __cplusplus
extern "C" {
#endif

#if defined(LBMAPI_EXPORTS)
	#define LBMAPI_API __declspec(dllexport)
#elif defined(LBMAPIDLL)
	#define LBMAPI_API __declspec(dllimport)
#else
	#define LBMAPI_API
#endif

typedef void * LBMHANDLE;

#define KCBP_PARAM_NODE			0
#define KCBP_PARAM_CLIENT_MAC		1
#define KCBP_PARAM_CONNECTION_ID	2
#define KCBP_PARAM_SERIAL		3

extern LBMAPI_API	LBMHANDLE LBMSTDCALL KCBP_Init( void * pCA) ;
extern LBMAPI_API	int LBMSTDCALL KCBP_Exit( LBMHANDLE hHandle) ;
extern LBMAPI_API	int LBMSTDCALL KCBP_Abort(LBMHANDLE hHandle, int nErrno, int
nLevel, char * szErrMsg ) ;

extern LBMAPI_API	int LBMSTDCALL KCBP_GetCommLen( LBMHANDLE hHandle, int
*pnLen) ;

extern LBMAPI_API	int LBMSTDCALL KCBP_GetValue( LBMHANDLE hHandle, char *
KeyName, char * Value ) ;
extern LBMAPI_API	int LBMSTDCALL KCBP_GetValueN( LBMHANDLE hHandle, char *
KeyName, char * Value, int Num ) ;

extern LBMAPI_API	int LBMSTDCALL KCBP_BeginWrite( LBMHANDLE hHandle) ;

extern LBMAPI_API	int LBMSTDCALL KCBP_SetValue( LBMHANDLE hHandle, char *
KeyName, char * Value ) ;

extern LBMAPI_API	int LBMSTDCALL KCBP_RsCreate( LBMHANDLE hHandle, char *Name,
int ColNum, char * TableInfo) ;
extern LBMAPI_API	int LBMSTDCALL KCBP_RsNewTable( LBMHANDLE hHandle, char
*Name, int ColNum, char * TableInfo) ;
extern LBMAPI_API	int LBMSTDCALL KCBP_RsAddRow( LBMHANDLE hHandle) ;
```

```
extern LBMAPI_API  int LBMSTDCALL KCBP_RsSetCol( LBMHANDLE hHandle, int Col, char * Value) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsSetColByName( LBMHANDLE hHandle, char * Name, char * Value) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsSaveRow( LBMHANDLE hHandle) ;

extern LBMAPI_API  int LBMSTDCALL KCBP_RsOpen( LBMHANDLE hHandle) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetCursorName(LBMHANDLE hHandle,char * pszCursorName, int nLen);
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColNames(LBMHANDLE hHandle,char *pszInfo, int nLen);
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColName(LBMHANDLE hHandle,int nColIndex, char *pszInfo, int nLen);
extern LBMAPI_API  int LBMSTDCALL KCBP_RsFetchRow( LBMHANDLE hHandle) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetCol( LBMHANDLE hHandle, int Col, char * Value) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColN( LBMHANDLE hHandle, int Col, char * Value, int Num) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColByName( LBMHANDLE hHandle, char * Name, char * Value) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColByNameN( LBMHANDLE hHandle, char * Name, char * Value, int Num) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetRowNum( LBMHANDLE hHandle, int *pnRows) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetColNum( LBMHANDLE hHandle, int *pnCols) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetTableRowNum( LBMHANDLE hHandle, int nTableIndex, int *pnRows) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsGetTableColNum( LBMHANDLE hHandle, int nTableIndex, int *pnCols) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsMore( LBMHANDLE hHandle) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RsClose( LBMHANDLE hHandle) ;

extern LBMAPI_API  int LBMSTDCALL KCBP_Commit( LBMHANDLE hHandle) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_RollBack( LBMHANDLE hHandle) ;
extern LBMAPI_API  int LBMSTDCALL KCBP_Abend( LBMHANDLE hHandle, char * AbendCode) ;

extern LBMAPI_API  int LBMSTDCALL KCBP_PrintStatus( LBMHANDLE hHandle, char * statusbuf, ... ) ;

extern LBMAPI_API  int LBMSTDCALL KCBP_GetSystemParam( LBMHANDLE hHandle, int nParam, char *pszBuffer, int nBufSize) ;
```

```
extern LBMAPI_API    int LBMSTDCALL KCBP_Sleep( LBMHANDLE hHandle, int nSeconds) ;

extern LBMAPI_API    void * LBMSTDCALL KCBP_Malloc( LBMHANDLE hHandle, int nBytes,
bool bShared) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_Free( LBMHANDLE hHandle, char * pMem) ;
#define KCBP_MallocLocal( hHanddle, nBytes) KCBP_Malloc( hHandle, nBytes, 0)
#define KCBP_MallocShared( hHandle, nBytes) KCBP_Malloc( hHandle, nBytes, 1)

extern LBMAPI_API    int LBMSTDCALL KCBP_SaveToCwa( LBMHANDLE hHandle, int nPos,
void * pMem, int len) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_LoadFromCwa( LBMHANDLE hHandle, int
nPos, void * pMem, int len) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_LockCwa( LBMHANDLE hHandle, int nPos,
int nLen) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_UnLockCwa( LBMHANDLE hHandle, int nPos,
int nLen ) ;

extern LBMAPI_API    int LBMSTDCALL KCBP_CallProgram( LBMHANDLE hHandle, char *
prg) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_CallProgramSys( LBMHANDLE hHandle, char
* prg, char * sys_id) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_CallProgramExt( LBMHANDLE hHandle, char
* prg) ;
extern LBMAPI_API    int LBMSTDCALL KCBP_CallProgramSysExt( LBMHANDLE hHandle,
char * prg, char * sys_id) ;

#ifdef __cplusplus
}
#endif

#endif
```

3.4 LBMAPI 函数说明

参数用法说明：**INPUT** 表示输入参数，**OUTPUT** 表示输出参数，**INPUT/OUTPUT** 既是输入又是输出。

3.4.1 初始化 KCBP 环境

函数原型：extern LBMAPI_API　LBMHANDLE LBMSTDCALL KCBP_Init(void * pCA) ;
输入参数：

参数名称	参数说明	用法
void * pCA	KCBP 通讯区指针	Input

返　　回：NULL 失败，其它成功。

用法说明：返回值 NULL，表示初始化失败，应调用 KCBP_Exit 退出。

3.4.2 退出 KCBP 程序

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_Exit(LBMHANDLE hHandle) ;
输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败
用法说明：退出 KCBP 服务程序，调用该函数。该函数与 KCBP_Init 配对使用。

3.4.3 终止 KCBP 程序

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_Abort(LBMHANDLE hHandle, int nErrno, int nLevel, char * szErrMsg) ;
输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Int nErrno	错误号	Input
Int nLevel	错误级别	Input
char * szErrMsg	错误信息	Input

返　　回：0 成功，其它失败
用法说明：终止 KCBP 服务程序。该函数一般用在 KCBP_Init 失败时返回错误信息。

3.4.4 清除公共数据区

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_BeginWrite(LBMHANDLE hHandle) ;
参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败
用法说明：表示开始写通信用的公共数据区，它的真正作用是清除该公共数据区。注意，如果重新开始写通信用的公共数据区(比如在错误处理时)，应该再次调用 BeginWrite 函数，这样可以清除原来的内容。

3.4.5 取通讯区当前长度

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_GetCommLen(LBMHANDLE hHandle, int *pnLen) ;
参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
int *pnLen	通讯区长度指针	Output

返　　回：0 成功，长度放在 pnLen 中，其它失败

用法说明：获取公共数据区当前数据长度

3.4.6 根据键名(KeyName)设置键值(Value)

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_SetValue(LBMHANDLE hHandle, char * KeyName, char * Vlu) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *KeyName	关键字	Input
char *Value	关键字值	Input

返　　回：0 成功，其它失败

用法说明：

此函数通过其参数 KeyName 指定的关键字来存储字符串值 Value，可以通过 GetValue 函数并使用相同的关键字来获取设置的字符串值。

如果 KeyName 为空字符串，则设置整个公共数据区代表的字符串。

关键字 KeyName 是任意定义的，可以在程序规划时确定，或由服务程序的程序员和客户程序的程序员事先约定。

SetValue 函数和 GetValue 函数是 KCBP 用于传递单值(0 维结构)的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。注意，传递的值只能是字符串。

SetValue 函数最好在 RsCreate 函数之前使用。

3.4.7 根据键名(KeyName)取键值(Value)

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_GetValue(LBMHANDLE hHandle, char * KeyName, char * Value) ;

extern LBMAPI_API　int LBMSTDCALL KCBP_GetValueN(LBMHANDLE hHandle, char * KeyName, char * Value, int Num) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Char *KeyName	关键字	Input
Char *Value	关键字值	Input
Int nLen	Value 缓冲区长度	Input

返　　回：0 成功，其它失败

用法说明：

根据键名(KeyName)取键值(Vlu)。

此函数通过其参数 KeyName 指定的关键字来获取通过 SetValue 函数来设置的字符串值。

如果 GetValue 的参数指定的关键字并没有值，则返回空字符串。

如果 KeyName 为空字符串，则返回整个公共数据区代表的字符串。

关键字 KeyName 是任意定义的，可以在程序规划时确定，或由服务程序的程序员和客户程序的程序员事先约定。

SetValue 函数和 GetValue 函数是 KCBP 用于传递单值(0 维结构)的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。注意，传递的值只能是字符串。

GetValue 函数最好在 RsOpen 函数之前使用。

注意，如果不设置 nLen，调用 GetValue 函数容易引起 C 的越界错误。nLen 可以限定获取不超过指定长度的字符串。

KCBP_GetValue 与 **KCBP_GetValueN** 的差别在于 **N** 指定了输出 **buffer** 长度，可防止越界。

3.4.8 创建结果集

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsCreate(LBMHANDLE hHandle, char *Name, int ColNum, char * pColInfo) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Char *Name	结果集名称	Input
int ColNum	指定结果集的列数	Input
Char *pColInfo	结果集列名称表	Input

返　　回：0 成功，其它失败

用法说明：

RsCreate 函数和 RsOpen 函数是 KCBP 传递二维结构的标准方法。其方向既可以是服务器到客户机，也可以是客户机到服务器。

KCBP 系统限制行长，每行不要超过 4K。

RsCreate 用于创建第 1 个结果集。

结果集列名称表格式如”col1,col2,col3”。

3.4.9 增加结果集

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsNewTable(LBMHANDLE hHandle, char *Name, int ColNum, char * pColInfo) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Char *Name	结果集名称	Input
int ColNum	指定结果集的列数	Input
Char *pColInfo	结果集列名称表	Input

返　　回：0 成功，其它失败

用法说明：

RsCreate 用于创建第 1 个结果集，RsNewTable 用于增加后续结果集。

3.4.10 使公共数据区的结果集增加一行

函数原型： extern LBMAPI_API int LBMSTDCALL KCBP_RsAddRow(LBMHANDLE hHandle) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：

3.4.11 根据列号设置结果集中当前行的某一列值

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_RsSetCol(LBMHANDLE hHandle, int Col, char * Value) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
int Col	列序号，从 1 开始编号	Input
char *Value	列值	Input

返　　回：0 成功，其它失败

用法说明：

3.4.12 根据列名设置结果集中当前行的某一列值

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_RsSetColByName(LBMHANDLE hHandle, char * Name, char * Value) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *Name	列名称	Input
char *Value	列值	Input

返　　回：0 成功，其它失败

用法说明：

列名称在 **KCBP_**RsCreate 及 **KCBP_**RsNewTable 时通过列名表设置。

3.4.13 在公共数据区的结果集中存储当前行

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_RsSaveRow(LBMHANDLE hHandle) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：

3.4.14 打开结果集

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_RsOpen(LBMHANDLE hHandle) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：

0 表示打开成功，并且可以确定结果集的行数；

100 表示打开成功，但不能确定结果集的行数，KCBP 大查询采用该种方式返回数据；其它失败。

用法说明：

返回值 0 时，可以用 RsGetTableColNum、RsGetTableRowNum 等函数取表的行、列数。返回值 100 时，结果集的行数不确定。不能用 RsGetTableColNum、RsGetTableRowNum 等函数取表的行、列数。这时，可以用 RsGetColNum 取当前结果集列数。如需确定是否有后续结果集，需要用 RsMore 查询。

3.4.15 获取当前结果集名称

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetCursorName(LBMHANDLE hHandle,char * pszCursorName, int nLen);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
PszCursorName	结果集名称	output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，名称放在 pszCursorName 中；其它失败。

用法说明：结果集名称是在 RsCreate 或 RsNewTable 时设定的。

3.4.16 获取当前结果集的全部列名称

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColNames(LBMHANDLE hHandle,char *pszInfo, int nLen);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
PszInfo	列名表	Output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，结果集列名称表放在 pszInfo 中；其它失败。

用法说明：结果列名表是在 RsCreate 或 RsNewTable 时设定的。

3.4.17 获取当前结果指定列的名称

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColName(LBMHANDLE hHandle,int nColIndex, char *pszInfo, int nLen);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
NcolIndex	列序号	Input
PszInfo	列名	Output
Nlen	输出缓冲区长度	Input

返　　回：0 成功，列名放在 pszInfo 中；其它失败。

用法说明：结果列名是在 RsCreate 或 RsNewTable 时通过列名表设定的。

3.4.18 获取公共数据区的当前结果集的行数

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetRowNum(LBMHANDLE hHandle, int *pnRows) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
int *pnRows	结果集行数指针	output

返　　回：0 成功，行数放在 nRows 中；其它失败。

用法说明：

3.4.19 获取公共数据区的当前结果集的列数

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColNum(LBMHANDLE hHandle, int *pnCols) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Int *pnCols	结果集列数	output

返　　回：0 成功，列数放在 nCols 中；其它失败。

用法说明：

3.4.20 获取公共数据区的指定结果集的行数

函数原型：

extern LBMAPI_API int LBMSTDCALL KCBP_RsGetTableRowNum(LBMHANDLE hHandle, int nTableIndex, int *pnRows) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Int nt	结果集编号，从 1 开始	input
int *pnRows	结果集行数	output

返　　回：0 成功，行数放在 nRows 中；其它失败。

用法说明：

3.4.21 获取公共数据区的指定结果集的列数

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetTableColNum(LBMHANDLE hHandle, int nTableIndex, int *pnCols) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Int nt	结果集编号，从 1 开始	input
int *pnCols	结果集列数	output

返　　回：0 成功，列数放在 nCols 中；其它失败。

用法说明：

3.4.22 从公共数据区的结果集中依序获取一行，作为当前行

函数原型： extern LBMAPI_API int LBMSTDCALL KCBP_RsFetchRow(LBMHANDLE hHandle) ;

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：

3.4.23 从结果集的当前行的某一列取值（根据列序号）

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetCol(LBMHANDLE hHandle, int Col, char * Value) ;

extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColN(LBMHANDLE hHandle, int Col, char * Value, int Num) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
int Col	列序号，从 1 开始编号	Input
char *Value	列值，指向存储列值的缓冲区	Output
int Num	输出缓冲区长度	Input

返　　回：0 成功，其它失败

用法说明：

3.4.24 从结果集的当前行的某一列取值(根据列名)

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColByName(LBMHANDLE hHandle, char * Name, char * Value) ;

extern LBMAPI_API int LBMSTDCALL KCBP_RsGetColByNameN(LBMHANDLE hHandle, char * Name, char * Value, int Num) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
const char *ColName	列名，由服务程序的 RsSetColNameList 函数设置。	Input
char *Value	列值，指向存储列值的缓冲区	Output
int Num	输出缓冲区长度	Input

返　　回：0 成功，其它失败

用法说明：KCBP_RsGetColByName 与 KCBP_RsGetColByNameN 区别在于 N 限定最大输出。

3.4.25 查询后续结果集

函数原型：int KCBP_RsMore();

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 有后续结果集，其它无

用法说明：

用 RsOpen 结果集后，如果返回 100，说明是一个大查询，这时重复调用 RsFetchRow，当 RsFetchRow 返回非 0 时，意味着当前结果集已经结束，这时需要用 RsMore 查询是否有后续结果集，如果有，继续 RsFetchRow 操作。

3.4.26 关闭结果集

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_RsClose(LBMHANDLE hHandle) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败。

用法说明：该函数与客户端交互时，用于关闭结果集占用的资源；与另一个服务端交互时，KCBP_RsClose 通知对方 Server 端停止发送结果集数据并清除结果集占用的存储资源。

3.4.27 同步调用本系统的一个服务程序

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_CallProgram(LBMHANDLE hHandle, char * prg) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *prg	服务程序名	Input

返　　回：返回 **0** 表示成功，其它失败。

用法说明：客户端程序等待此服务程序返回后再继续执行，调用后系统自动提交。

3.4.28 同步调用本系统的一个服务程序(不提交)

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_CallProgramExt(LBMHANDLE hHandle, char * prg) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *prg	服务程序名	Input

返　　回：返回 **0** 表示成功，其它失败。

用法说明：客户端程序等待此服务程序返回后再继续执行，调用后系统不自动提交。

3.4.29 同步调用其它 KCBP 系统的一个服务程序

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_CallProgramSys(LBMHANDLE hHandle, char * prg, char * sys_id) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *prg	服务程序名	Input
char *sys_id	被调用的服务程序所在节点的 SYS_ID	Input

返　　回：返回 0 表示成功，其它失败。

用法说明：调用后等待此服务程序返回后再继续执行，调用后系统自动提交。

3.4.30 同步调用其它 KCBP 系统的一个服务程序(不提交)

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_CallProgramSysExt(LBMHANDLE hHandle, char * prg, char * sys_id) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *prg	服务程序名	Input
char *sys_id	被调用的服务程序所在节点的 SYS_ID	Input

返　　回：返回 0 表示成功，其它失败。

用法说明：调用后等待此服务程序返回后再继续执行，调用后系统不自动提交。

3.4.31 提交事务

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_Commit(LBMHANDLE hHandle) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：注意，KCBP 服务端应用程序应调用本函数而不是“EXEC SQL COMMIT;”来提交事务。

3.4.32 回滚事务

函 数 原 型： extern LBMAPI_API int LBMSTDCALL KCBP_RollBack(LBMHANDLE hHandle) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：注意，KCBP 服务端应用程序应调用本函数而不是“EXEC SQL ROLLBACK;”来回滚事务。

3.4.33 放弃事务

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_Abend(LBMHANDLE hHandle, char * AbendCode) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input

返　　回：0 成功，其它失败

用法说明：放弃一个事务，并指定放弃代码。

3.4.34 输出调试信息

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_PrintStatus(LBMHANDLE hHandle, char * statusbuf, ...) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char * statusbuf	输出格式信息	Input
…	输出变参	Input

返　　回：0 成功，其它失败

用法说明：该函数把一个字符串写到 KCBP 的消息控制台(message console),该信息还被保存系统运行文件中，注意，每次冷启动 KCBP 时，保存输出信息的文件被清空。

3.4.35 SLEEP 函数

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_Sleep(LBMHANDLE hHandle, int nSeconds) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nSeconds	睡眠时间，单位秒	Input

返　　回：0 成功，其它失败

用法说明：KCBP 提供的安全 sleep 函数,不受信号干扰。

3.4.36 取 KCBP 系统信息

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_GetSystemParam(LBMHANDLE hHandle, int nParam, char *pszBuffer, int nBufSize) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nParam	参数编号	Input

char *pszBuffer	用于存放返回参数	Output
int nBufSize	缓冲区长度	Input

返　　回：0 成功，其它失败

用法说明：

int nParam 取值含义：

```
#define KCBP_PARAM_NODE              0      /*KCBP 节点号，5Byte*/
#define KCBP_PARAM_CLIENT_MAC        1      /*客户端网卡地址, 12Byte */
#define KCBP_PARAM_CONNECTION_ID     2      /*KCBP 连接号, 10Byte */
#define KCBP_PARAM_SERIAL            3      /*KCBP 交易流水号, 26Byte */
```

3.4.37 分配内存

函数原型：extern LBMAPI_API　void * LBMSTDCALL KCBP_Malloc(LBMHANDLE hHandle, int nBytes, bool bShared) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nBytes	分配长度	Input
bool bShared	共享标志	Input

返　　回：函数返回分配到的内存地址，NULL 失败，其它成功

用法说明：

KCBP_Malloc 分配的内存由 KCBP 系统管理，malloc 分配的内存由操作系统管理。KCBP_Malloc 分配的内存可以有效地防止内存泄漏。

它有 2 种分配方式：private 和 shared。Private 归该笔交易进程所有，shared 归整个 KCBP 系统所有。

3.4.38 分配 PRIVATE 内存

函数原型：#define KCBP_MallocLocal(hHanddle, nBytes) KCBP_Malloc(hHandle, nBytes, 0)

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
NBytes	长度	Input

返　　回：0 成功，其它失败

用法说明：宏定义，分配 private 内存。

3.4.39 分配 SHARED 内存

函数原型：#define KCBP_MallocShared(hHandle, nBytes) KCBP_Malloc(hHandle, nBytes, 1)

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
Nbytes	长度	Input

返　　回：0 成功，其它失败

用法说明：宏定义，分配 shared 内存。

3.4.40 释放内存

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_Free(LBMHANDLE hHandle, char * pMem) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char * pMem	要释放的内存指针	Input

返　　回：0 成功，其它失败

用法说明：

3.4.41 写 CWA

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_SaveToCwa(LBMHANDLE hHandle, int nPos, void * pMem, int len) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nPos	CWA 位置	Input
void * pMem	要写入的 buffer	Input
int len	要写入的长度	Input

返　　回：0 成功，其它失败

用法说明：该函数从 CWA 中 nPos 开始写入 Len 个 Bytes。

3.4.42 读 CWA

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_LoadFromCwa(LBMHANDLE hHandle, int nPos, void * pMem, int len) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nPos	CWA 位置	Input
void * pMem	要写入的 buffer	Output
int len	要写入的长度	Input

返　　回：0 成功，其它失败

用法说明：该函数从 CWA 中 nPos 开始读出 Len 个 Bytes。

3.4.43 CWA 加锁

函数原型：extern LBMAPI_API　int LBMSTDCALL KCBP_LockCwa(LBMHANDLE hHandle, int nPos, int nLen) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nPos	CWA 位置	Input

int len	要写入的长度	Input

返　　回：0 成功，其它失败

用法说明：该函数锁定 CWA 中 nPos 开始的 Len 个 Bytes。

3.4.44 CWA 解锁

函数原型：extern LBMAPI_API int LBMSTDCALL KCBP_UnLockCwa(LBMHANDLE hHandle, int nPos, int nLen) ;

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nPos	CWA 位置	Input
int len	要写入的长度	Input

返　　回：0 成功，其它失败

用法说明：该函数解除 CWA 中 nPos 开始的 Len 个 Bytes 锁定状态。

4. LBM 编程注意事项

1. LBM 程序入口需要调用 KCBP_Init(),出口处需要调用 KCBP_Exit()。
2. 所有 1 维数据传输应该在 2 维数据传输之前执行。
3. 程序中的系统调用及库函数需要调用需要保证线程安全。
4. 程序中不可以使用的系统函数

fork
execl
system
用 KCBP_CallProgram 代替。

gethostbyname
gethostbyaddr
getprotent
getservbyname
用以_r 结尾的函数

exit
abort
用 KCBP_Exit()

5. 不推荐使用的系统函数

malloc、free 系列，用 KCBP_Malloc,KCBP_Free 代替。
kill
sleep
assert
signals

6. 以下进程状态要注意关闭

open file descriptors
TCP/IP socket descriptors
Environment variables
Current Working Directory
Process priority
Shared memory
Dynamically allocated memory

7. 跨系统编程注意结构对齐方式，Intel 与 RS/6000 高低位转化

在 RS/6000 上可以使用以下#pragma
#pragma options align =packed
#pragma options align=reset

8. 注意数据库 Cursor 操作

EXEC SQL DECLARE CURSOR;
EXEC SQL OPEN CURSOR;
EXEC SQL CLOSE CURSOR;
EXEC SQL DEALLOCATE CURSOR; //不要忘记

9. 临时表操作

EXEC SQL SELECT * FROM table1 INTO TEMP tempTable;
EXEC SQL DROP TABLE tempTable; //不要忘记

10. LBM 中静态变量要慎用

11. 通过 KCBP 访问数据库

通过 KCBP 访问数据库，无需连接及关闭，数据库访问使用 EXEC SQL …; //要用 Sqlca.sqlcode 检查返回状态
数据库访问提交使用 KCBP_Commit、回滚用 KCBP_RollBack

由于 KCBP 系统内部维护数据库的连接，因此，LBM 中可以不用进行 CONNECT 和 DISCONNECT 数据库的操作，如果 LBM 自己进行 CONNECT 和 DISCONNECT，提交时，要用 EXEC SQL COMMIT，不能使用 KCBP_Commit。

12. 内存资源

private 交易独享存储量，交易结束后自动释放
shared 所有交易共享存储量，交易结束后需要显式释放

13. 公共数据区使用时不要越界

14. 使用 KCBP 设计交易系统时，业务处理逻辑力求简明，LBM 应该尽快结束。

15. 输出结果的长度不限，但输入条件长度不得大于 32K。
16. 调用 CallProgram 接口函数后一定要提交，或者直接调用 CallProgramAndCommit 接口函数。
17. 调用 CallProgram/CallProgramAndCommit 后，如果不调用 ConnectServer 而直接再次调用 SetValue 和 CallProgram/CallProgramAndCommit，应先调用 BeginWrite 以清空公共缓冲区(这是因为上次调用 CallProgram/CallProgramAndCommit 的返回值还在公共缓冲区中)。
18. 客户机尽量调用 CallProgramAndCommit，此调用效率最高。CallProgramAndCommit 是否最终提交取决于 CICS Server 执行 Program 的情况：
    - Ø 服务器没有未截获的错误，正常返回(调用 ExitEasyCics)时，事务提交
    - Ø 服务器没有未截获的错误，并显式调用 CicsCommit 时，事务提交
    - Ø 服务器没有未截获的错误，并显式调用 CicsRollBack 时，事务回滚
    - Ø 服务器有未截获的错误，事务回滚
19. 客户机调用 CallProgram 时，最终一定要调用 Commit 或 RollBack，事务是否提交以此为

准。但是在此期间，program 一直占据 application server。如果发生网络故障，须等到超时，才能完成事务回滚，在此期间，可能因数据库记录锁定导致相关存取挂起。

20. 客户机调用 CallProgram 系列函数后，最好调用 GetErr/ GetErrCode 函数以判断是否调用成功，若 GetErr 返回空串或 GetErrCode 返回 0，则表示成功；若返回非空，表示出错。
21. 编写 CICS 与 KCBP 通用业务程序注意事项

使用 LBMAPI 编写的业务程序，既可以运行在 KCBP SERVER 上，又可运行在 CICS SERVER 上。在 UNIX 环境下，当业务程序在 KCBP 系统上运行时，使用 liblbmapi.so，在 cics 系统上运行时，它使用 liblbmapicics.so。

如果 KCBPSERVER 与 CICSSERVER 在操作同一数据库，该程序编译时，要注意 package 时间戳的一致性。建议的做法是先由.sqc 预编译出.c 文件，并保存.c 文件，然后根据.c 分别编译出用于 CICS 的程序和用于 KCBP 的服务程序。

编译 CICS 服务程序时，注意要在编译选项中定义宏 _IBMCICS，而编译 KCBP 服务程序时，不要定义该宏。

另外注意，CICS 系统限制服务程序名称不能超过 8 位，DB2 系统缺省 PACKAGE 名称也不能超过 8 位，因此，建议业务程序文件命名时，前缀不要超过 8 位，否则需要特殊处理。

KCBP 系统设计时，为了支持线程式的服务器，采用 pCA 传递 handle，因此业务函数的参数表中，都有一个 void *pCA 的入参，这个参数在 CICS 上并不需要，因此，当为 CICS 编写业务程序时，该参数传入 NULL 即可。

KCBP 的服务程序 EXPORT 服务模块名称，CICS 服务程序 EXPORT Main 函数，因此，CICS 服务程序要通过如下代码完成服务模块调用。

```
#ifdef _IBMCICS
LBMAPI_API int LBMSTDCALL main()
{
    LBM_TEST(NULL);/*功能函数调用*/
}
#endif
```

5. 编程实例

5.1 客户端

```
#include "stdafx.h"

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <time.h>


#include "KCBPCli.hpp"


int main(int argc,char *argv[])

{

  int i, nCount=1, n=0;

  char szTmp[1024];

  time_t td,td1;

  int nColNums;

  int nResultset;

  int nReturnCode;

  int nRow;

  char szopt[10][20];


  printf("KCBP Test tool, Version 1.0, Mr. Yuwei Du, 2002.10\n");
```

```
if(argc<2)
{
    printf("Usage: CLITest transname [count servername ip port sendq receiveq user password]\n");
    exit(2);
}

memset(szopt,0,sizeof(szopt));
strcpy(szopt[2],"1");
strcpy(szopt[3],"KCBP01");
strcpy(szopt[4],"192.168.54.2");
strcpy(szopt[5],"22000");
strcpy(szopt[6],"req1");
strcpy(szopt[7],"ans1");
strcpy(szopt[8],"KCXP00");
strcpy(szopt[9],"888888");

for(i=2;i<argc;i++)
{
        strncpy(szopt[i],argv[i],sizeof(szopt[i])-1);
        break;
}

tagKCBPConnectOption stKCBPConnection;
memset(&stKCBPConnection, 0 , sizeof(stKCBPConnection));
```

```
strcpy(stKCBPConnection.szServerName, szopt[3]);
stKCBPConnection.nProtocal = 0;
strcpy(stKCBPConnection.szAddress, szopt[4]);
stKCBPConnection.nPort = atoi(szopt[5]);
strcpy(stKCBPConnection.szSendQName, szopt[6]);
strcpy(stKCBPConnection.szReceiveQName, szopt[7]);

CKCBPCli *pKCBPCli=new CKCBPCli();
if(!pKCBPCli) return 1;
if(pKCBPCli->SetConnectOption( stKCBPConnection ) )
{
    delete pKCBPCli;
    return 2;
}

if(pKCBPCli->SQLConnect(szopt[3],szopt[8],szopt[9]))
{
    delete pKCBPCli;
    return 3;
}

time(&td);
printf("Begin at %s", ctime(&td));
```

```
nCount = atoi(szopt[2]);


while(n++<nCount)
{
    pKCBPCli->BeginWrite();
    pKCBPCli->SetValue("QUERYID","1111");
    pKCBPCli->SQLExecute(argv[1]);


    nResultset=0;
    do{
        nReturnCode = pKCBPCli->SQLNumResultCols( &nColNums );
        if( nReturnCode !=0 )
        {
            printf("unknown resultset colnums, rc=%d\n", nReturnCode );
            break;
        }
        pKCBPCli->SQLGetCursorName( szTmp, 32);
        printf("Resultset %d %s \n",++nResultset, szTmp);
        nRow=0;
        while(1)
        {
            nReturnCode = pKCBPCli->SQLFetch();
            if( nReturnCode == 0 )
```

```
                { //have result
                    printf("Row = %d ", ++nRow);
                    for(i=1;i<=nColNums;i++)
                    {
                        nReturnCode = pKCBPCli->RsGetCol(i,szTmp);
                        if( nReturnCode == 0)
                            printf("%d=%s ",i,szTmp);
                        else
                            printf("error %d", nReturnCode);
                    }
                    printf("\n");
                }
                else
                {
                    printf("SQLFetch return %d\n", nReturnCode);
                    break;
                }
            }
        }while( pKCBPCli->SQLMoreResults()==0 );
        pKCBPCli->SQLCloseCursor();
    }
    time(&td1);
    printf("Begin at %s", ctime(&td));
    printf("End     at %s", ctime(&td1));
```

```
        printf("%d trans has been done in %.2f second\n", nCount,
    difftime(td1,td));
        if(difftime(td1,td)!=0)
        {
            printf("Average response time: %.2f/s\n", nCount/difftime(td1,td));
        }
        pKCBPCli->SQLDisconnect();
        delete pKCBPCli;
        return 0;
}
```

5.2 服务端

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "lbmapi.h"

#ifdef _WIN32
  #define DLLIMPORT __declspec(dllimport)
  #define DLLEXPORT __declspec(dllexport)
  #define CDECL __cdecl
#else
  #define DLLIMPORT
  #define DLLEXPORT
  #define CDECL
```

```
#endif

/* #define either ORA, SYBASE, INFORMIX or DB2 here */
#define DB2
#define SQLNOTFOUND 100
#if defined ( DB2 )
  #include <sql.h>
#elif defined ( ORA )
  #define SQLNOTFOUND 1403
#endif


void* LBM_TEST(void *pCA){
  char statusbuf[1024], s[30];
  int i;
  LBMHANDLE hHandle;

  hHandle=KCBP_Init(pCA);
  if(hHandle==NULL) return NULL;
  KCBP_GetValue(hHandle, "QUERYID",s);
  printf("\nQUERYID=%s\n",s);
  KCBP_BeginWrite(hHandle);
  KCBP_RsCreate(hHandle, "table1", 3,"id,msg,name");
  i=0;
```

```
    do
    {
        KCBP_RsAddRow(hHandle);
        KCBP_RsSetCol(hHandle, 1,"123" );
        KCBP_RsSetCol(hHandle, 2,"456" );
        KCBP_RsSetCol(hHandle, 3,"789" );
        KCBP_RsSaveRow(hHandle);
    }while(i++<10);

    KCBP_RsNewTable(hHandle,"table2", 4,"abb,bbb,cbb,dbb");
    i=0;
    do
    {
        KCBP_RsAddRow(hHandle);
        KCBP_RsSetColByName(hHandle, "abb","ooooooooooooooo");
        KCBP_RsSetColByName(hHandle, "bbb","aaaaaaaaaaaaaaa");
        KCBP_RsSetColByName(hHandle, "cbb","uuuuuuuuuuuuuuu");
        KCBP_RsSetColByName(hHandle,"dbb","rrrrrrrrrrrrrrr");
        KCBP_RsSaveRow(hHandle);
    }while(i++<800);

    KCBP_Exit(hHandle);
    return NULL;
}
```

```
void* LBM_TEST1(void *pCA){
  char statusbuf[1024], s[30];
  LBMHANDLE hHandle;

  hHandle=KCBP_Init(pCA);
  if(hHandle==NULL) return NULL;
  KCBP_GetValue(hHandle, "QUERYID",s);
  printf("\nQUERYID=%s\n",s);
  KCBP_BeginWrite(hHandle);
  KCBP_SetValue(hHandle,"ARG1","ONE");
  KCBP_SetValue(hHandle,"ARG2","TWO");
  KCBP_SetValue(hHandle,"ARG3","THREE");
  KCBP_CallProgram(hHandle,"LBMTEST2");
  //view result
  KCBP_Exit(hHandle);
  return NULL;
}
void* LBM_TEST2(void *pCA)
{
  char statusbuf[1024], s[30];
  int i;
  LBMHANDLE hHandle;
```

```
hHandle=KCBP_Init(pCA);
if(hHandle==NULL) return NULL;
KCBP_GetValue(hHandle, "QUERYID",s);
KCBP_PrintStatus(hHandle, "QUERYID=%s",s);
KCBP_GetValue(hHandle, "ARG1",s);
KCBP_PrintStatus(hHandle,"ARG1=%s",s);
KCBP_GetValue(hHandle, "ARG2",s);
KCBP_PrintStatus(hHandle,"ARG2=%s",s);
KCBP_GetValue(hHandle, "ARG3",s);
KCBP_PrintStatus(hHandle,"ARG3=%s",s);

KCBP_BeginWrite(hHandle);
KCBP_RsCreate(hHandle, "table1", 3,"id,msg,name");
i=0;
do
{
    KCBP_RsAddRow(hHandle);
    KCBP_RsSetCol(hHandle, 1,"123" );
    KCBP_RsSetCol(hHandle, 2,"456" );
    KCBP_RsSetCol(hHandle, 3,"789" );
    KCBP_RsSaveRow(hHandle);
}while(i++<5000);

KCBP_RsNewTable(hHandle, "table2", 4,"abb,bbb,cbb,dbb");
```

```
    i=0;
    do
    {
        KCBP_RsAddRow(hHandle);
        KCBP_RsSetColByName(hHandle, "abb","ooooooooooooooo");
        KCBP_RsSetColByName(hHandle, "bbb","aaaaaaaaaaaaaaa");
        KCBP_RsSetColByName(hHandle, "cbb","uuuuuuuuuuuuuuu");
        KCBP_RsSetColByName(hHandle,"dbb","rrrrrrrrrrrrrrr");
        KCBP_RsSaveRow(hHandle);
    }while(i++<5000);

    KCBP_Exit(hHandle);
    return NULL;
}
void* LBM_TEST3(void *pCA){
    char statusbuf[1024], s[30];
    LBMHANDLE hHandle;

    hHandle=KCBP_Init(pCA);
    if(hHandle==NULL) return NULL;
    KCBP_GetValue(hHandle, "QUERYID",s);
    printf("\nQUERYID=%s\n",s);
    KCBP_BeginWrite(hHandle);
    KCBP_SetValue(hHandle,"ARG1","ONE");
```

```
  KCBP_SetValue(hHandle,"ARG2","TWO");
  KCBP_SetValue(hHandle,"ARG3","THREE");
  KCBP_CallProgramExt(hHandle,"LBMTEST2");
  //view result
  KCBP_Exit(hHandle);
  return NULL;
}
void* LBM_TEST4(void *pCA){
  char statusbuf[1024], s[30];
  LBMHANDLE hHandle;

  hHandle=KCBP_Init(pCA);
  if(hHandle==NULL) return NULL;
  KCBP_GetValue(hHandle, "QUERYID",s);
  printf("\nQUERYID=%s\n",s);
  KCBP_BeginWrite(hHandle);
  KCBP_SetValue(hHandle,"ARG1","ONE");
  KCBP_SetValue(hHandle,"ARG2","TWO");
  KCBP_SetValue(hHandle,"ARG3","THREE");
  KCBP_CallProgramSys(hHandle,"LBMTEST2","2");
  //view result
  KCBP_Exit(hHandle);
  return NULL;
}
```

```
void* LBM_TEST5(void *pCA){
  char statusbuf[1024], s[30];
  LBMHANDLE hHandle;

  hHandle=KCBP_Init(pCA);
  if(hHandle==NULL) return NULL;
  KCBP_GetValue(hHandle, "QUERYID",s);
  printf("\nQUERYID=%s\n",s);
  KCBP_BeginWrite(hHandle);
  KCBP_SetValue(hHandle,"ARG1","ONE");
  KCBP_SetValue(hHandle,"ARG2","TWO");
  KCBP_SetValue(hHandle,"ARG3","THREE");
  KCBP_CallProgramSys(hHandle,"LBMTEST2","1");
  //view result
  KCBP_Exit(hHandle);
  return NULL;
}

void* LBM_TEST6(void *pCA){
  char statusbuf[1024], s[30];
  LBMHANDLE hHandle;

  hHandle=KCBP_Init(pCA);
  if(hHandle==NULL) return NULL;
```

```
    KCBP_GetValue(hHandle, "QUERYID",s);
    printf("\nQUERYID=%s\n",s);
    KCBP_BeginWrite(hHandle);
    KCBP_SetValue(hHandle,"ARG1","ONE");
    KCBP_SetValue(hHandle,"ARG2","TWO");
    KCBP_SetValue(hHandle,"ARG3","THREE");
    KCBP_CallProgramSys(hHandle,"LBMTEST9","1");
    //view resu
    KCBP_Exit(hHandle);
    return NULL;
}
```

5.3 发布/订阅

5.3.1 发布

```
#include "stdafx.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include "KCBPCli.hpp"

int main(int argc, char *argv[])
{
```

```
int i, nCount=0, nTimeout=60, n=0;

char szTmp[1024];

time_t td,td1;

int nColNums;

int nResultset;

int nReturnCode;

int nRow;

char szopt[10][20];

tagKCBPPSControl stKCBPPSControl;

char szData[32767];

int nRet;


printf("KCBP subcribe test tool, Version 1.0, Mr. Yuwei Du, 2003.09\n");

if(argc<2)

{

    printf("Usage: subscribe topic [timemout servername ip port sendq receiveq user password]\n");

    printf("example: subscribe test\n");

    printf("    subscribe test 60\n");

    printf("    subscribe test 60 KCBP01\n");

    printf("    subscribe test 60 KCBP01 192.168.1.20 21000 req2 ans2 9999 888888\n");

    exit(2);

}


memset(szopt,0,sizeof(szopt));

strcpy(szopt[2],"60"); //60 second

strcpy(szopt[3],"KCBP01");

strcpy(szopt[4],"192.168.1.20");
```

```
strcpy(szopt[5],"21000");
strcpy(szopt[6],"req1");
strcpy(szopt[7],"ans1");
strcpy(szopt[8],"KCXP00");
strcpy(szopt[9],"888888");

for(i=2;i<argc;i++)
{
    strncpy(szopt[i],argv[i],sizeof(szopt[i])-1);
}

tagKCBPConnectOption stKCBPConnection;
memset(&stKCBPConnection, 0 , sizeof(stKCBPConnection));
strcpy(stKCBPConnection.szServerName, szopt[3]);
stKCBPConnection.nProtocal = 0;
strcpy(stKCBPConnection.szAddress, szopt[4]);
stKCBPConnection.nPort = atoi(szopt[5]);
strcpy(stKCBPConnection.szSendQName, szopt[6]);
strcpy(stKCBPConnection.szReceiveQName, szopt[7]);

CKCBPCli *pKCBPCli=new CKCBPCli();
if(!pKCBPCli) return 1;
if(argc>4)
{
    if(pKCBPCli->SetConnectOption( stKCBPConnection ) )
    {
        delete pKCBPCli;
        return 2;
```

```
        }
    }

    if(pKCBPCli->SQLConnect(szopt[3],szopt[8],szopt[9]))
    {
        delete pKCBPCli;
        return 3;
    }

    time(&td);
    printf("Begin at %s", ctime(&td));

    nTimeout = atoi(szopt[2]);

    memset(&stKCBPPSControl, 0, sizeof(stKCBPPSControl));
    stKCBPPSControl.nExpiry = nTimeout;

    nRet = KCBPCLI_RegisterPublisher((KCBPCLIHANDLE)pKCBPCli,
&stKCBPPSControl, argv[1]) ;
    if(nRet!=0)
    {
        printf("KCBPCLI_RegisterPublisher fail ret=%d\n", nRet) ;
        goto LABEL_EXIT;
    }
    printf("id=%s, msgid=%s, corrid=%s\n", stKCBPPSControl.szId,
stKCBPPSControl.szMsgId, stKCBPPSControl.szCorrId);

    while( difftime(time(NULL), td) <= nTimeout )
    {
```

```
        stKCBPPSControl.nPriority = 5; //publish message priority
        stKCBPPSControl.nExpiry = 30; //publish message lifetime 30 second
        sprintf(szData, "this is test message %d", nCount);
        nRet = KCBPCLI_Publish((KCBPCLIHANDLE)pKCBPCli,
&stKCBPPSControl, argv[1], szData, strlen(szData)) ;
        if(nRet==0)
        {
            nCount++;
            printf("topic:%s, message:%s\n", argv[1], szData);
        }
        sleep(2);
    }
    nRet = KCBPCLI_DeregisterPublisher( (KCBPCLIHANDLE)pKCBPCli,
&stKCBPPSControl);
    if(nRet!=0)
    {
        printf("KCBPCLI_DeregisterPublisher fail ret=%d\n", nRet) ;
    }
LABEL_EXIT:
    time(&td1);
    printf("Begin at %s", ctime(&td));
    printf("End   at %s", ctime(&td1));
    printf("%d message has been send in %.2f second\n", nCount, difftime(td1,td));
    pKCBPCli->SQLDisconnect();
    delete pKCBPCli;
    return 0;
}
```

5.3.2 订阅

```
#include "stdafx.h"


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>


#include "KCBPCli.hpp"


int main(int argc,char *argv[])
{
  int i, nCount=0, nTimeout=60, n=0;
  char szTmp[1024];
  time_t td,td1;
  int nColNums;
  int nResultset;
  int nReturnCode;
  int nRow;
  char szopt[10][20];
  tagKCBPPSControl stKCBPPSControl;
  char szData[32767];
  int nRet;
  printf("KCBP subcribe test tool, Version 1.0, Mr. Yuwei Du, 2003.09\n");
  if(argc<2)
  {
      printf("Usage: subscribe topic [timemout servername ip port sendq receiveq user
password]\n");
```

```
        printf("example: subscribe test\n");
        printf("    subscribe test 60\n");
        printf("    subscribe test 60 KCBP01\n");
        printf("    subscribe test 60 KCBP01 192.168.1.20 21000 req2 ans2 9999
888888\n");
        exit(2);
    }

    memset(szopt,0,sizeof(szopt));
    strcpy(szopt[2],"60"); //60 second
    strcpy(szopt[3],"KCBP01");
    strcpy(szopt[4],"192.168.1.20");
    strcpy(szopt[5],"21000");
    strcpy(szopt[6],"req1");
    strcpy(szopt[7],"ans1");
    strcpy(szopt[8],"KCXP00");
    strcpy(szopt[9],"888888");

    for(i=2;i<argc;i++)
    {
        strncpy(szopt[i],argv[i],sizeof(szopt[i])-1);
    }

    tagKCBPConnectOption stKCBPConnection;
    memset(&stKCBPConnection, 0 , sizeof(stKCBPConnection));
    strcpy(stKCBPConnection.szServerName, szopt[3]);
    stKCBPConnection.nProtocal = 0;
    strcpy(stKCBPConnection.szAddress, szopt[4]);
    stKCBPConnection.nPort = atoi(szopt[5]);
```

```
strcpy(stKCBPConnection.szSendQName, szopt[6]);
strcpy(stKCBPConnection.szReceiveQName, szopt[7]);

CKCBPCli *pKCBPCli=new CKCBPCli();
if(!pKCBPCli)
{
    printf("new CKCBPCli() return null\n");
    return 1;
}
if(argc>4)
{
    if(nRet = pKCBPCli->SetConnectOption( stKCBPConnection ) )
    {
        delete pKCBPCli;
        printf("pKCBPCli->SetConnectOption fail %d\n", nRet);
        return 2;
    }
}

if(nRet = pKCBPCli->SQLConnect(szopt[3],szopt[8],szopt[9]))
{
    delete pKCBPCli;
    printf("pKCBPCli->SQLConnect(%s,%s,%s)  %d\n",  szopt[3],szopt[8],szopt[9],
nRet);
    return 3;
}

time(&td);
printf("Begin at %s", ctime(&td));
```

```
  nTimeout = atoi(szopt[2]);


  memset(&stKCBPPSControl, 0, sizeof(stKCBPPSControl));
  stKCBPPSControl.nExpiry = nTimeout;


 nRet = KCBPCLI_Subscribe( (KCBPCLIHANDLE)pKCBPCli, &stKCBPPSControl,
argv[1], "") ;
  if(nRet!=0)
  {
        printf("KCBPCLI_Subscribe fail ret=%d\n", nRet) ;
        goto LABEL_EXIT;
  }
  printf("id=%s, msgid=%s, corrid=%s\n", stKCBPPSControl.szId,
stKCBPPSControl.szMsgId, stKCBPPSControl.szCorrId);


  while( difftime(time(NULL), td) <= nTimeout )
  {
        stKCBPPSControl.nExpiry = 1; //wait for 1 second
        memset(szData, 0, sizeof(szData));
        nRet = KCBPCLI_ReceivePublication((KCBPCLIHANDLE)pKCBPCli,
&stKCBPPSControl, szData, sizeof(szData)-1) ;
        if(nRet==0)
        {
              nCount++;
              printf("reveive message:%s\n", szData);
        }
        else
        {
```

```
                printf("return=%d\n", nRet);
            }
        }
        nRet = KCBPCLI_Unsubscribe( (KCBPCLIHANDLE)pKCBPCli, &stKCBPPSControl);
        if(nRet!=0)
        {
            printf("KCBPCLI_Unsubscribe fail ret=%d\n", nRet) ;
        }
LABEL_EXIT:
        time(&td1);
        printf("Begin at %s", ctime(&td));
        printf("End   at %s", ctime(&td1));
        printf("%d message has been receive in %.2f second\n", nCount, difftime(td1,td));
        pKCBPCli->SQLDisconnect();
        delete pKCBPCli;
        return 0;
}
```

6. 参考资料

- 《计算机软件工程规范国家标准汇编 2000》
- 《KCBP 需求说明书》
- 《KCBP 概要设计》
- 《KCBP 详细设计》
- 《EasyCics 开发手册》
- 《KCXP 应用程序编程参考书》
- 《XML 开发手册》
- 《SOAP XML 跨平台 Web Service 开发技术》
- 《COM+技术解决方案设计》