

# KCBP

(Kingdom Core Business Platform)

## 程序员手册



## 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	程序员手册		
摘要			
当前版本	V3.0		
日期	2013.6.14		
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
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
V1.2	2003/12/31	杜玉巍	增加客户端 API 函数： 异步调用接口： KCBPCLI_ACallProgramAndCommit KCBPCLI_ACallProgram KCBPCLI_GetReply KCBPCLI_Cancel 强制断开函数： KCBPCLI_DisConnectForce
V1.3	2004/06/30	杜玉巍	增加客户端 API 函数： KCBPCLI_SetOptions KCBPCLI_GetOptions KCBPCLI_GetSystemParam KCBPCLI_SetSystemParam 增加 LBM API 函数： KCBP_GetCurrentXAName
V1.4	2004/08/01	杜玉巍	增加了支持二进制数据、大数据操作 API 函数： 客户端： KCBPCLI_GetVal KCBPCLI_SetVal KCBPCLI_RsGetVal KCBPCLI_RsGetValByName KCBPCLI_RsSetVal KCBPCLI_RsSetValByName 服务端： KCBP_GetVal KCBP_SetVal KCBP_RsGetVal KCBP_RsGetValByName KCBP_RsSetVal KCBP_RsSetValByName

V2.0	2005/08/01	杜玉巍	增加了扩展编程、编程工具、部署、客户端编程注意事项等章节 编程实例增加了两个 Oracle LBM 编程实例 新增 API: KCBP_GetXAHandle 修改 API: KCBPCLI_GetReply 增加了方式 0 KCBPCLI_SetOptions 连接参数增加了 SSL 和 PROXY 配置
V2.1	2006/09/22	杜玉巍	HelloWorld 客户端增加设置服务方 IP 及端口的代码。
V2.2	2007/12/10	杜玉巍	支持异 GetReply 0 方式与同步方式共享队列。 支持非确认式发送消息
V2.5	2009/9/15	陈洪涛	LBM API 新增函数 KCBP_SetOptions KCBP_GetOptions KCBP_GetErrCode KCBP_GetErrMsg KCBP_GetCommArea KCBP_System KCBP_CallProgramByCA KCBP_SetStatus KCBP_GetCwa KCBP_SetCommArea
V3.0	2013/6/14	杜玉巍	增加以下函数说明: KCBPCLI_SetCommArea KCBPCLI_GetCommArea 增加 KCBPCLI_SetOption 选项 15-20 说明。 4.1 增加客户端连接池说明,增加常见错误号 1059,1060,1061,1064,1065,1068 说明。 客户端 API 说明统一更改为 C 语言 API 的说明。 增加了 7.1.3 kcbpcp 加载和调试 LBM 说明。

# 目录

<b>1. 简介</b> .....	<b>1</b>
1.1. KCBP API 与 KCBP 系统 .....	1
1.2. KCBP 应用程序编程思维模型 .....	2
1.3. KCBP 应用程序结构图 .....	2
1.4. KCBP 请求数据处理流程 .....	3
1.5. KCBP API 编程 HELLO WORLD! .....	4
<b>2. KCBP CLIENT API</b> .....	<b>7</b>
2.1. CLIENT API 通用调用流程 .....	7
2.2. CLIENT API 函数分类 .....	7
2.3. CLIENT API 头文件 .....	8
2.3.1. <i>KCBPCLI.H</i> .....	8
2.4. KCBP CLIENT API 函数说明 .....	25
2.4.1. 设置连接选项 .....	25
2.4.2. 查询连接选项 .....	27
2.4.3. 连接 KCBP SERVER .....	28
2.4.4. 断开与 KCBP SERVER 的连接 .....	28
2.4.5. 强制断开连接 .....	28
2.4.6. 同步调用一个服务程序 (提交方式) .....	29
2.4.7. 同步调用一个服务程序 (不提交方式) .....	29
2.4.8. 提交事务 .....	29
2.4.9. 回滚事务 .....	29
2.4.10. 异步调用 (提交方式) .....	30
2.4.11. 异步调用 (不提交方式) .....	30
2.4.12. 查询异步调用结果 .....	31
2.4.13. 取消异步调用 .....	32
2.4.14. 订阅 .....	33
2.4.15. 查询订阅消息 .....	33
2.4.16. 取消订阅 .....	34
2.4.17. 登记发布主题 .....	34
2.4.18. 发布 .....	35
2.4.19. 取消发布主题 .....	35
2.4.20. 清除公共数据区 .....	36
2.4.21. 根据键名 (KeyName) 设置键值 (Value) .....	36
2.4.22. 根据键名 (KeyName) 设置指定长度的键值 (Value) .....	37
2.4.23. 根据键名 (KeyName) 取键值 (Value) .....	37
2.4.24. 根据键名 (KeyName) 取键值 (Value) 及键值长度 .....	38
2.4.25. 创建结果集 .....	38
2.4.26. 增加结果集 .....	39
2.4.27. 使公共数据区的结果集增加一行 .....	39
2.4.28. 根据列号设置结果集中当前行的某一列值 .....	39
2.4.29. 根据列名设置结果集中当前行的某一列值 .....	40
2.4.30. 根据列号设置当前行的某一列值为指定长度的缓冲区 .....	40
2.4.31. 根据列名设置当前行的某一列值为指定长度的缓冲区 .....	40
2.4.32. 在公共数据区的结果集中存储当前行 .....	41
2.4.33. 打开结果集 .....	42
2.4.34. 获取当前结果集名称 .....	42
2.4.35. 获取当前结果集的全部列名称 .....	42
2.4.36. 获取当前结果指定列的名称 .....	42
2.4.37. 获取公共数据区的当前结果集的行数 .....	43
2.4.38. 获取公共数据区的当前结果集的列数 .....	43
2.4.39. 获取公共数据区的指定结果集的行数 .....	43

# 目录

2.4.40.	获取公共数据区的指定结果集的列数.....	44
2.4.41.	从公共数据区的结果集中依序获取一行，作为当前行.....	44
2.4.42.	从结果集的当前行的某一列取值（根据列序号）.....	44
2.4.43.	从结果集的当前行的某一列取值（根据列名）.....	44
2.4.44.	获得当前行的某一列的缓冲区和长度（根据列序号）.....	45
2.4.45.	获得当前行的某一列的缓冲区和长度（根据列名称）.....	45
2.4.46.	查询后续结果集.....	45
2.4.47.	关闭结果集.....	46
2.4.48.	获取公共数据区当前长度.....	46
2.4.49.	设置调用超时时间.....	46
2.4.50.	返回错误码和错误信息.....	47
2.4.51.	返回错误码.....	47
2.4.52.	返回错误信息.....	47
2.4.53.	查询API 当前版本号.....	47
2.4.54.	查询系统参数.....	48
2.4.55.	设置系统参数.....	53
2.4.56.	查询通讯参数.....	60
2.4.57.	设置通讯参数.....	60
2.4.58.	直接设置通讯缓冲区.....	61
2.4.59.	直接读取通讯缓冲区.....	61
2.4.60.	SQL 风格函数.....	61
<b>3.</b>	<b>KCBP 服务端编程接口 LBMAPI.....</b>	<b>63</b>
3.1.	服务端程序调用流程.....	63
3.1.1.	服务端程序通用流程.....	63
3.1.2.	获得输入参数.....	63
3.1.3.	返回结果.....	63
3.1.3.1.	使用 0 维表返回结果.....	63
3.1.3.2.	使用 2 维表返回结果.....	63
3.2.	LBMAPI 分类.....	64
3.3.	LBMAPI 头文件.....	65
3.4.	LBMAPI 函数说明.....	71
3.4.1.	初始化 KCBP 环境.....	71
3.4.2.	退出 KCBP 程序.....	71
3.4.3.	终止 KCBP 程序.....	72
3.4.4.	清除公共数据区.....	72
3.4.5.	取通讯区当前长度.....	72
3.4.6.	根据键名 (KeyName) 设置键值 (Value).....	72
3.4.7.	根据键名 (KeyName) 设置键值 (Value) 为指定长度的缓冲区.....	73
3.4.8.	根据键名 (KeyName) 取键值 (Value).....	73
3.4.9.	根据键名 (KeyName) 取键值 (Value) 缓冲区和长度.....	74
3.4.10.	创建结果集.....	74
3.4.11.	增加结果集.....	75
3.4.12.	使公共数据区的结果集增加一行.....	75
3.4.13.	根据列号设置结果集中当前行的某一列值.....	75
3.4.14.	根据列名设置结果集中当前行的某一列值.....	76
3.4.15.	根据列号设置当前行的某一列值为指定长度的缓冲区.....	76
3.4.16.	根据列名设置当前行的某一列值为指定长度的缓冲区.....	76
3.4.17.	在公共数据区的结果集中存储当前行.....	77
3.4.18.	打开结果集.....	77
3.4.19.	获取当前结果集名称.....	77
3.4.20.	获取当前结果集的全部列名称.....	78
3.4.21.	获取当前结果指定列的名称.....	78

# 目录

3.4.22.	获取公共数据区的当前结果集的行数.....	78
3.4.23.	获取公共数据区的当前结果集的列数.....	78
3.4.24.	获取公共数据区的指定结果集的行数.....	79
3.4.25.	获取公共数据区的指定结果集的列数.....	79
3.4.26.	从公共数据区的结果集中依序获取一行，作为当前行.....	79
3.4.27.	从结果集的当前行的某一列取值（根据列序号）.....	79
3.4.28.	从结果集的当前行的某一列取值（根据列名）.....	80
3.4.29.	获得当前行的某一列的缓冲区和长度（根据列序号）.....	80
3.4.30.	获得当前行的某一列的缓冲区和长度（根据列名称）.....	81
3.4.31.	查询后续结果集.....	81
3.4.32.	关闭结果集.....	81
3.4.33.	同步调用本系统的一个服务程序.....	81
3.4.34.	同步调用本系统的一个服务程序（不提交）.....	82
3.4.35.	同步调用其它 KCBP 系统的一个服务程序.....	82
3.4.36.	同步调用其它 KCBP 系统的一个服务程序（不提交）.....	82
3.4.37.	提交事务.....	83
3.4.38.	回滚事务.....	83
3.4.39.	放弃事务.....	83
3.4.40.	分配内存.....	83
3.4.41.	分配局部内存.....	84
3.4.42.	分配共享内存.....	84
3.4.43.	释放内存.....	84
3.4.44.	写 CWA.....	84
3.4.45.	读 CWA.....	85
3.4.46.	CWA 加锁.....	85
3.4.47.	CWA 解锁.....	85
3.4.48.	输出调试信息.....	85
3.4.49.	SLEEP 函数.....	86
3.4.50.	取 KCBP 系统信息.....	86
3.4.51.	XA 选择.....	87
3.4.52.	取 XA 句柄.....	87
3.4.53.	获取当前 XA 名称.....	88
3.4.54.	调试函数.....	88
3.4.55.	发布函数.....	88
3.4.56.	切换到缺省 XA，并获取句柄.....	89
3.4.57.	设置 LBM 调用相关的参数.....	89
3.4.58.	获取 LBM 调用相关的参数.....	90
3.4.59.	返回错误码.....	90
3.4.60.	返回错误信息.....	90
3.4.61.	获取通讯缓冲区内容.....	91
3.4.62.	直接设置通讯缓冲区内容.....	91
3.4.63.	获取/设置 KCBP 系统相关参数值.....	91
3.4.64.	通过指定缓冲区发起本地调用.....	93
3.4.65.	设置事务状态.....	93
3.4.66.	获取 CWA 信息.....	94
<b>4.</b>	<b>编程注意事项.....</b>	<b>95</b>
4.1.	客户端编程注意事项.....	95
4.2.	LBM 编程注意事项.....	98
<b>5.</b>	<b>编程实例.....</b>	<b>101</b>
5.1.	客户端.....	101
5.2.	服务端 LBM.....	106

# 目录

5.2.1.	不访问数据库的 LBM .....	106
5.2.2.	访问 Oracle 数据库的 LBM .....	114
5.2.3.	多线程状态运行的 KCBP 上访问 ORACLE 数据库的 LBM .....	116
5.3.	发布/订阅 .....	120
5.3.1.	发布 .....	120
5.3.2.	订阅 .....	125
5.4.	异步调用 .....	129
<b>6.</b>	<b>扩展编程 .....</b>	<b>137</b>
6.1.	用户出口 .....	137
6.1.1.	概述 .....	137
6.1.2.	用户出口程序范例 .....	139
6.1.3.	UserExit.h 文件 .....	141
6.1.4.	UserExit.cpp 文件 .....	146
6.2.	XA 编程 .....	158
6.2.1.	KCBPXA 简介 .....	158
6.2.2.	KCBPXA 分类 .....	159
6.2.3.	KCBPXA 规范中的核心数据结构 .....	159
6.2.4.	KCBP 对 XA 接口的调度流程 .....	160
6.2.5.	KCBPXA+ 介绍 .....	161
6.2.5.1.	LBM 程序中如何获得 XA 的指针并调用类中的方法 .....	161
6.2.5.2.	功能类如何与 KCBP 交互 .....	161
6.2.5.3.	虚类化的必要性 .....	161
6.2.5.4.	资源断开重连的约定 .....	162
6.2.6.	范例 1: DB21PC 的实现 .....	162
6.2.7.	范例 2: ODBC1PC 的实现 .....	181
6.2.7.1.	KCBPXA.h .....	181
6.2.7.2.	虚类定义文件 KCBPDatabase.h .....	186
6.2.7.3.	功能类头文件 odbclpc.h .....	190
6.2.7.4.	KCBPXA 实现代码 odbclpc.cpp .....	193
6.2.8.	范例 3: SemArray 的实现 .....	205
6.2.8.1.	用法描述 .....	205
6.2.8.2.	XA 配置 .....	205
6.2.8.3.	操作 .....	206
6.2.8.4.	例子程序 .....	206
6.2.8.5.	SemArray 实现代码 .....	207
<b>7.</b>	<b>编程工具 .....</b>	<b>229</b>
7.1.	程序调用工具 KCBPCP .....	229
7.1.1.	单次调用 LBM .....	229
7.1.2.	重复调用 LBM .....	230
7.1.3.	调试 LBM .....	230
7.2.	LBM 调试工具 DEBUGLBM .....	231
7.2.1.	使用 debuglbn 直接调用 LBM .....	232
7.2.2.	Windows 上使用 VC 和 debuglbn 调试 LBM .....	233
7.2.3.	LINUX 上使用 gdb+debuglbn 调试 LBM .....	234
7.2.4.	debuglbn 的局限性 .....	235
7.3.	LBM 提交工具 KCBPADD .....	235
7.3.1.	命令参数解释 .....	235
7.3.2.	makefile 中 kcbpadd 使用举例 .....	236
7.4.	KCBP 系统压力工具 KCBPTEST .....	239
7.4.1.	程序文件说明 .....	240
7.4.2.	配置文件说明 .....	241
7.4.3.	指令文件格式 .....	243

# 目录

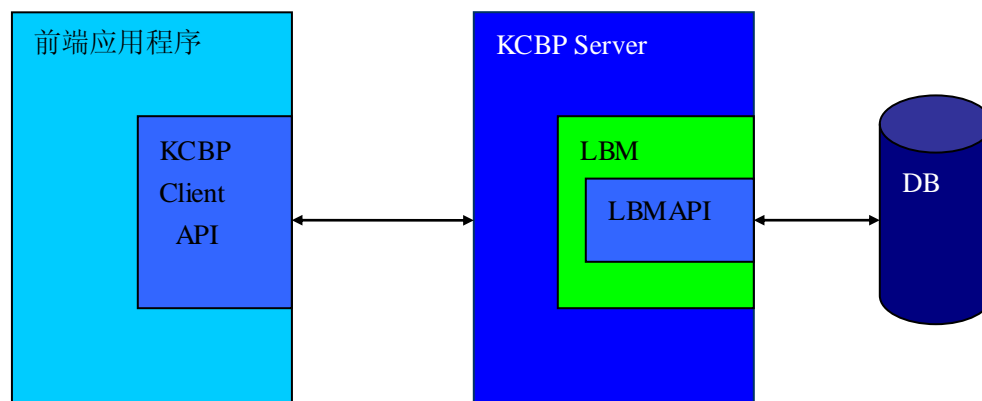
---

7.4.4.	测试系统架构.....	243
7.4.5.	运行界面.....	244
7.4.6.	界面显示结果说明.....	245
7.4.7.	适配器接口函数列表.....	246
7.4.8.	应用案例：使用 KCBPTest 对金证交易系统 V3.2 进行查询资金及股份的压力测试 247	
7.5.	LBM 压力测试工具 LBMTEST .....	249
<b>8.</b>	<b>部署方案.....</b>	<b>250</b>
8.1.	KCBP 和 KCXP 的关系 .....	250
8.2.	KCXP 多级部署方式 .....	252
8.2.1.	中心和远程两级 KCXP 部署.....	252
8.2.2.	中心和 n 个远程两级 KCXP 部署.....	253
8.2.3.	中心、分中心、远程三级 KCXP 部署 .....	254
8.3.	KCBP 动态负载均衡部署 .....	255
8.3.1.	单级单 KCXP+ 多 KCBP 动态负载均衡部署 .....	255
8.3.2.	单级多 KCXP+ 单 KCBP 动态负载均衡部署 .....	256
8.3.3.	单级多 KCXP+ 多 KCBP 动态负载均衡容错部署 .....	257
8.3.4.	多级多 KCXP+ 多 KCBP 动态负载均衡容错部署 .....	258
8.3.5.	多级 KCXP+ 多 KCBP 网状动态负载均衡容错部署 .....	258
8.4.	KCBP 转发部署 .....	259
8.4.1.	一对一转发部署.....	259
8.4.2.	一对多转发部署 1 .....	260
8.4.3.	一对多转发部署 2 .....	260
8.4.4.	一对多转发部署 3 .....	261
8.5.	KCXP 的消息重定向 .....	261
<b>9.</b>	<b>参考资料.....</b>	<b>263</b>



## 1. 简介

### 1.1. KCBP API 与 KCBP 系统

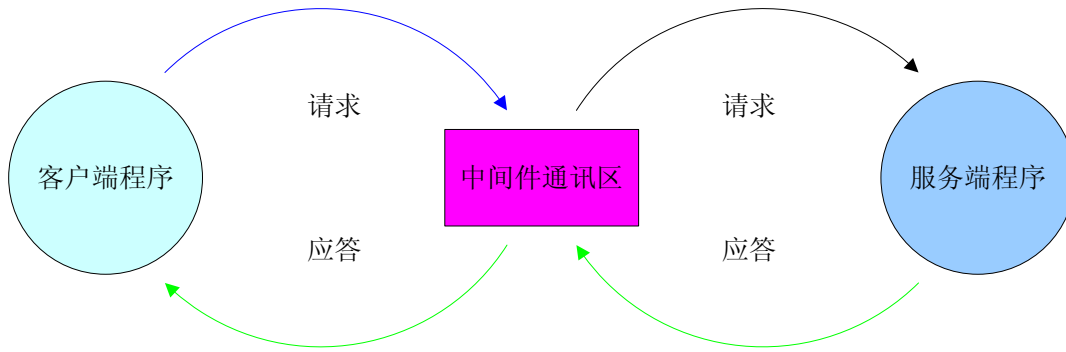


说明:

- 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；目前 KCBP 客户端 API 由两种：JAVA API 和 API。C API 有两种接口：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 Loadable business module 的缩写)。LBM 使用 LBM API(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 ESQ/C 方式访问数据库。
- LBM API 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 LBM API.DLL。LBM API 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 LBM API，这样，使用 LBM API 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。

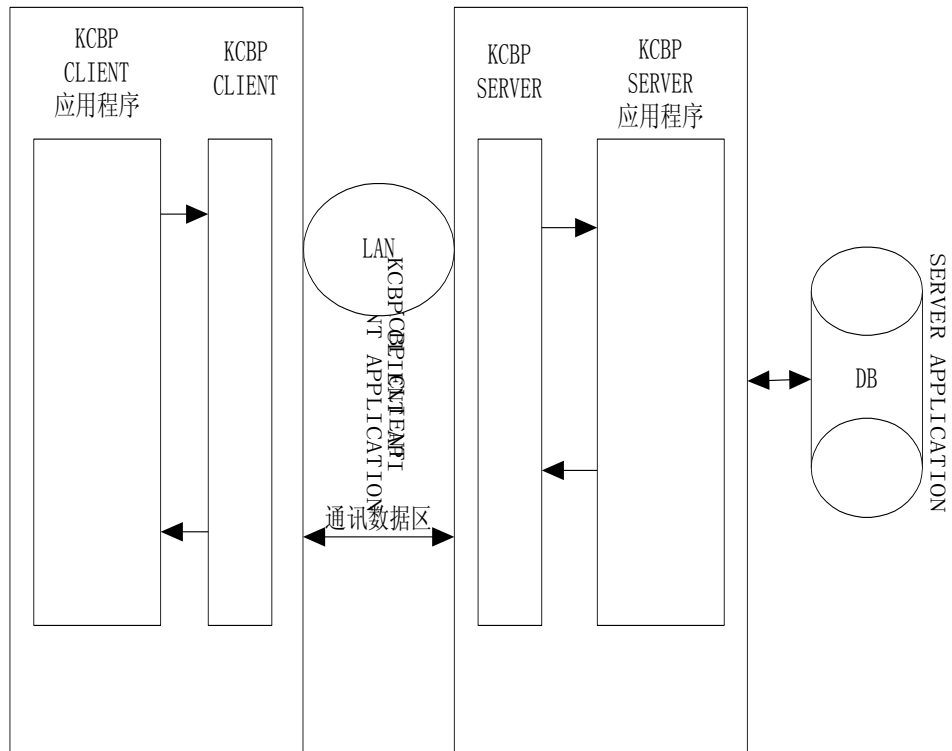
## 1.2. KCBP 应用程序编程思维模型

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。

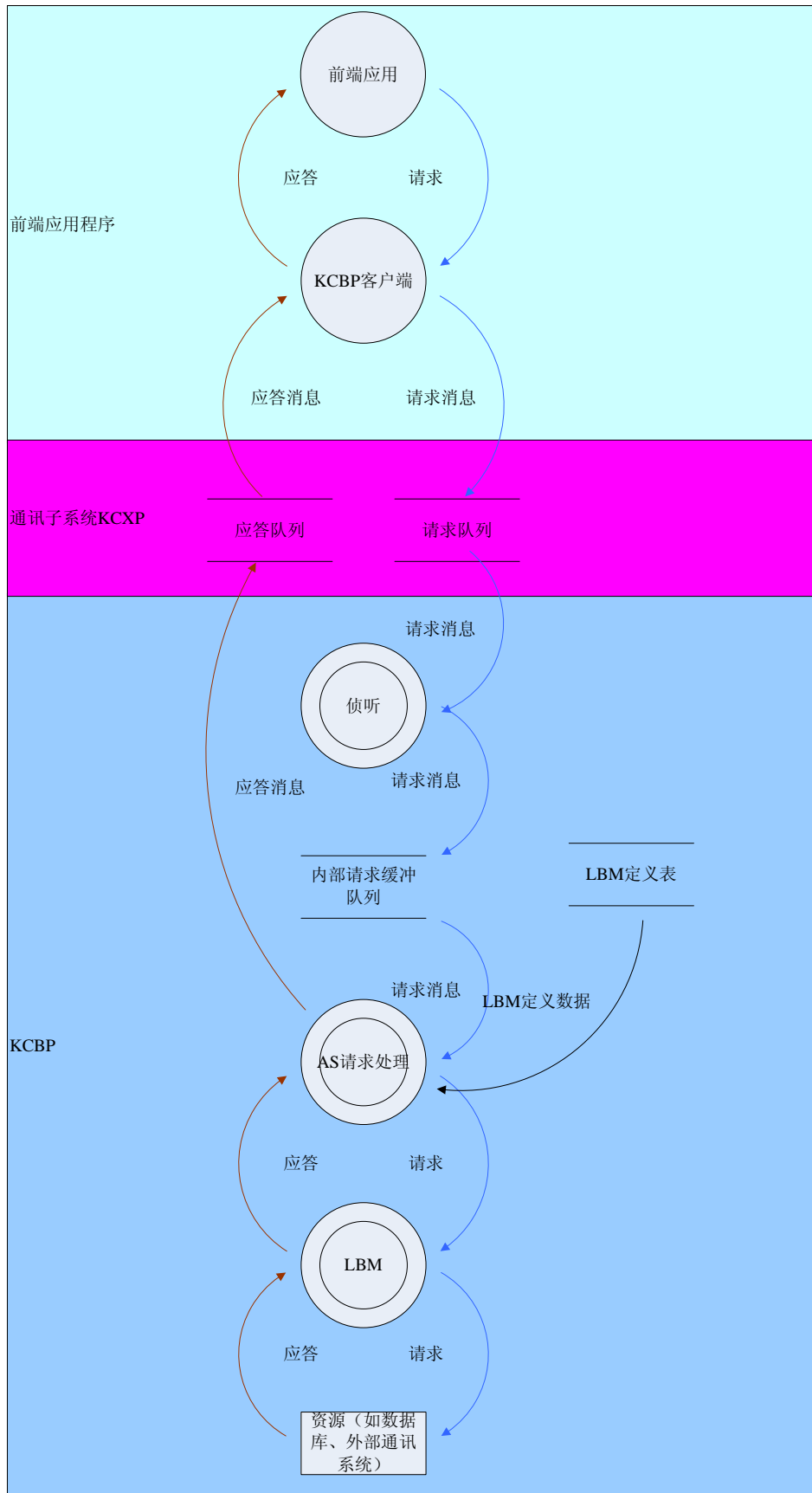


对应用程序开发者来讲，不用关心跨系统、跨平台、分布式等复杂操作，系统中的复杂操作由中间件系统处理。

## 1.3. KCBP 应用程序结构图



### 1.4. KCBP 请求数据处理流程



整个流程图分为三层：前端应用、通讯子系统、KCBP。

请求处理流程描述如下：

1. 前端应用程序发起请求，将其传递给 KCBP Client API;
2. KCBP Client API 调用 KCXP API 将请求转换成消息，传递到请求消息队列中;
3. 侦听线程从请求消息队列中取得请求消息;
4. 侦听线程将请求放到内部请求缓冲队列中;
5. AS 线程从内部请求队列中取得请求
6. AS 线程从请求中取得请求的 LBM 服务名称，从 LBM 定义表中查找 LBM 定义信息，并加载、执行相应的 LBM（可访问数据库）
7. LBM 接收请求，完成业务处理（可能访问资源）并返回结果给 AS 线程;
8. AS 线程将结果以消息形式发送到应答消息队列中;
9. KCBP Client API 从应答消息队列中取得应答消息;
10. KCBP Client 将消息转换成用户可识别的结果返回给前端应用程序。

## 1.5. 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.h"
int main()
{
    int nRet;
    char szInfo[20];
    KCBPCLIHANDLE hHandle;          /*KCBP 客户端句柄声明*/
    tagKCBPConnectOption stKCBPConnection;
    nRet = KCBPCLI_Init(&hHandle);  /*初始化客户端句柄*/
    if(nRet != 0)
    {
        return 1;
    }
}
```

```
    }

    memset(&stKCBPConnection, 0, sizeof(stKCBPConnection));
    strcpy(stKCBPConnection.szServerName, "KCBP1");
    stKCBPConnection.nProtocal = 0;
    strcpy(stKCBPConnection.szAddress, "192.168.40.65");
    stKCBPConnection.nPort = 21000;
    strcpy(stKCBPConnection.szSendQName, "req1b");
    strcpy(stKCBPConnection.szReceiveQName, "ans1");
    KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT, &stKCBPConnection,
sizeof(stKCBPConnection));

    /*连接 KCBP SERVER*/
    nRet = KCBPCLI_ConnectServer(hHandle, "KCBP1", "TEST", "TEST");
    if(nRet != 0)
    {
        printf("KCBPCLI_ConnectServer return %d\n", nRet);
        goto LabelExit;
    }
    KCBPCLI_BeginWrite(hHandle);

    /*调用名称为 HELLO 的服务端程序*/
    nRet = KCBPCLI_CallProgramAndCommit(hHandle, "HELLO");
    if(nRet != 0)
    {
        printf("KCBPCLI_CallProgramAndCommit return %d\n", nRet);
        goto LabelExit;
    }

    /*读 MESSAGE 的内容*/
    nRet = KCBPCLI_GetValue(hHandle, "MESSAGE", szInfo, sizeof(szInfo) - 1);
    szInfo[sizeof(szInfo) - 1] = 0x0;
    if(nRet != 0)
    {
        printf("KCBPCLI_GetValue return %d\n", nRet);
        goto LabelExit;
    }

    /*输出信息*/
    printf("%s\n", szInfo);

    /*断开连接*/
    nRet = KCBPCLI_DisConnect(hHandle);
    if(nRet != 0)
```

```
{  
    printf("KCBPCLI_DisConnect return %d\n", nRet);  
    goto LabelExit;  
}
```

LabelExit:

```
KCBPCLI_Exit(hHandle);  
return 0;  
}
```

服务器端:

```
#include <stdlib.h>  
#include "lbmapi.h"  
extern "C" LBMEXPORTS void* HelloWorld(void *pCA) /* pCA 传送通讯区参数 */  
{  
    int nRet;  
    LBMHANDLE hHandle; /* 声明 LBMHANDLE */  
    nRet = KCBP_Initialize(&hHandle, pCA); /* 初始化 LBM API, 获得 LBMHANDLE */  
    if(nRet != 0)  
    { /* 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 KCBPCLI\_ConnectServer 接口函数
2. 使用 KCBPCLI\_SetValue 接口函数设置输入的条件(可选)
3. 调用 KCBPCLI\_CallProgramAndCommit 接口函数
4. 使用 KCBPCLI\_GetValue 接口函数或 KCBPCLI\_RsOpen 接口函数得到输出的结果
5. 如果继续发送请求，先调用 KCBPCLI\_BeginWrite 清空公共缓冲区，再重复 STEP 2-STEP5
6. 调用 KCBPCLI\_DisConnect 断开连接

### 2.2. CLIENT API 函数分类

连接	设置通讯选项	KCBPCLI_SetConnectOption		
	查询通讯选项	KCBPCLI_GetConnectOption		
	连接	KCBPCLI_ConnectServer		
	断开	KCBPCLI_DisConnect		
	强制断开	KCBPCLI_DisConnectForce		
服务调用	异步	SERVER 端确认	KCBPCLI_ACallProgramAndCommit	
		CLIENT 端确认	KCBPCLI_ACallProgram	
		查应答	KCBPCLI_GetReply	
		取消调用	KCBPCLI_Cancel	
	同步	SERVER 端确认	KCBPCLI_CallProgramAndCommit	
		CLIENT 端确认	KCBPCLI_CallProgram	
			KCBPCLI_Commit	
			KCBPCLI_RollBack	
订阅	KCBPCLI_Subscribe KCBPCLI_Unsubscribe KCBPCLI_ReceivePublication			
发布	KCBPCLI_RegisterPublisher KCBPCLI_DeregisterPublisher KCBPCLI_Publish			
数据传输	初始化	KCBPCLI_BeginWrite		
	获取传输长度	KCBPCLI_GetCommLen		
	0 维表	设置值	KCBPCLI_SetValue, KCBPCLI_SetVal	
		获取值	KCBPCLI_GetValue, KCBPCLI_GetVal	
	2 维表	写	创建结果集	KCBPCLI_RsCreate
			增加结果集	KCBPCLI_RsNewTable
			增加行	KCBPCLI_RsAddRow
			设当前行各列值	KCBPCLI_RsSetCol KCBPCLI_RsSetColByName KCBPCLI_RsSetVal, KCBPCLI_RsSetValByName

	读	保存当前行	KCBPCLI_RsSaveRow
		打开表、关闭表	KCBPCLI_RsOpen, KCBPCLI_RsClose
		取表名称	KCBPCLI_RsGetCursorName
		取表列名表	KCBPCLI_RsGetColNames
		取表列名	KCBPCLI_RsGetColName
		后续表查询	KCBPCLI_RsMore
		获取表行数	KCBPCLI_RsGetTableRowNum , KCBPCLI_RsGetRowNum
		获取表列数	KCBPCLI_RsGetTableColNum , KCBPCLI_RsGetColNum
		读取一行	KCBPCLI_RsFetchRow
		读当前行的列值	KCBPCLI_RsGetCol , KCBPCLI_RsGetColByName KCBPCLI_RsGetVal, KCBPCLI_RsGetValByName
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	查询系统参数	KCBPCLI_GetOptions	
	设置系统参数	KCBPCLI_SetOptions	
	查询通讯参数	KCBPCLI_GetSystemParam	
	设置通讯参数	KCBPCLI_SetSystemParam	
	设置调用超时	KCBPCLI_SetCliTimeOut	
	查询当前版本号	KCBPCLI_GetVersion	
ODBC SQL 风格 API 函数	连接	KCBPCLI_SQLConnect	
	断开	KCBPCLI_SQLKCBPCLI_DisConnect	
	调用服务程序	KCBPCLI_SQLExecute	
	查结果列数	KCBPCLI_SQLNumResultCols	
	取结果集名称	KCBPCLI_SQLGetCursorName	
	取列名表	KCBPCLI_SQLGetColNames	
	读一行	KCBPCLI_SQLFetch	
	查询后续结果集	KCBPCLI_SQLMoreResults	
	关闭结果集	KCBPCLI_SQLCloseCursor	
	结束交易	KCBPCLI_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_MAX 32
```

```
#define KCBP_MSGID_LEN          32

#define KCBP_CORRID_LEN         32

#define KCBP_USERID_LEN        12

#define KCBP_SERIAL_LEN        26

#define KCBP_XA_ENTRY_SIZE     32

#define KCBP_XA_PATH_SIZE      128

#define KCBP_XA_OPEN_SIZE      512

#define KCBP_XA_CLOSE_SIZE     128

#define KCBP_XA_OPTION_SIZE    512

#define KCBP_XA_SERIAL_SIZE    20

#define KCBP_PSFLAG_PERSISTENT  0x00000001

#define KCBP_PSFLAG_NOPERSISTENT 0x00000002

#define KCBP_PSFLAG_REDELIVERED 0x00000004

#define KCBP_PSFLAG_NOREDELIVERED 0x00000008

#define KCBP_PSFLAG_SYSTEM     0x00000010

#define KCBP_PSFLAG_USER       0x00000020

#define KCBP_OPTION_CONNECT     0

#define KCBP_OPTION_TIMEOUT     1

#define KCBP_OPTION_TRACE      2

#define KCBP_OPTION_CRYPT      3

#define KCBP_OPTION_COMPRESS   4
```

```
#define KCBP_OPTION_PARITY          5

#define KCBP_OPTION_SEQUENCE        6

#define KCBP_OPTION_CURRENT_CONNECT 7

#define KCBP_OPTION_CONNECT_HANDLE  8

#define KCBP_OPTION_CONFIRM          9

#define KCBP_OPTION_NULL_PASS       10

#define KCBP_OPTION_TIME_COST       11

#define KCBP_OPTION_CONNECT_EX      12

#define KCBP_OPTION_CURRENT_CONNECT_EX 13

#define KCBP_OPTION_AUTHENTICATION  14

#define KCBP_OPTION_GROUPID         15

#define KCBP_OPTION_USE_GROUPID     16

#define KCBP_OPTION_RECEIVE_PACKET_BY_GROUP 17

#define KCBP_OPTION_XA              18

#define KCBP_OPTION_CURRENT_XA      19

#define KCBP_OPTION_CONNECTION_POOL 20

#define KCBP_PARAM_NODE              0

#define KCBP_PARAM_CLIENT_MAC        1

#define KCBP_PARAM_CONNECTION_ID     2

#define KCBP_PARAM_SERIAL            3

#define KCBP_PARAM_USERNAME          4

#define KCBP_PARAM_PACKETTYPE       5

#define KCBP_PARAM_PACKETTYPE
KCBP_PARAM_PACKETTYPE
```

```
#define KCBP_PARAM_SERVICENAME      6

#define KCBP_PARAM_RESERVED        7

#define KCBP_PARAM_DESTNODE        8

#define KCBP_FETCH_NEXT            1

#define KCBP_FETCH_FIRST           2

#define KCBP_FETCH_LAST            3

#define KCBP_FETCH_PRIOR           4

#define KCBP_FETCH_ABSOLUTE        5

#define KCBP_FETCH_RELATIVE        6

#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_PROXY_MAX          128

#define KCBP_SSL_MAX            256

typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX + 1];

    int nProtocol;

    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];

    char szProxy[KCBP_PROXY_MAX + 1];

    char szSSL[KCBP_SSL_MAX + 1];
}

tagKCBPConnectOptionEx;

/* struct for kcbpxa's parameter */

typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX + 1]; //XA name
```

```
int nProtocol;

char szSwitchLoadFile[KCBP_XA_PATH_SIZE + 1];

char szXAEntry[KCBP_XA_ENTRY_SIZE + 1];

char szXAOpen[KCBP_XA_OPEN_SIZE + 1];

char szXAClose[KCBP_XA_CLOSE_SIZE + 1];

char szXASerial[KCBP_XA_SERIAL_SIZE + 1];

char szXAOption[KCBP_XA_OPTION_SIZE];

} tagKCBPXAConnectOption;

/* 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 ,
include type, mode, redelivered flag*/

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

    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*/

} tagCallCtrl;

typedef tagCallCtrl 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_DisConnectForce(KCBPCLIHANDLE hHandle);
```

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

```
/*synchronize call*/
```

```
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_CallEx(KCBPCLIHANDLE hHandle, char *ProgramName, int
nFlag, void *pInput, int nInputLen, void **pOutput, int *pOutputLen);

/*asynchronize call*/

KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_ACallProgramAndCommit(KCBPCLIHANDLE hHandle, char
*ProgramName, tagCallCtrl *ptagCallCtrl);

KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_ACallProgram(KCBPCLIHANDLE hHandle, char
*ProgramName, tagCallCtrl *ptagCallCtrl);

KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_GetReply(KCBPCLIHANDLE hHandle, tagCallCtrl
*ptagCallCtrl);

KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_Cancel(KCBPCLIHANDLE hHandle, tagCallCtrl
*ptagCallCtrl);

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);
```

KCBPCLI\_API int KCBPCLISTDCALL  
KCBPCLI\_GetVal(KCBPCLIHANDLE hHandle, char \*szKeyName,  
unsigned char \*\*pValue, long \*pSize);

KCBPCLI\_API int KCBPCLISTDCALL  
KCBPCLI\_SetVal(KCBPCLIHANDLE hHandle, char \*szKeyName,  
unsigned char \*pValue, long nSize);

KCBPCLI\_API int KCBPCLISTDCALL  
KCBPCLI\_Convert(KCBPCLIHANDLE hHandle, char \*szType, int  
nSrcByteOrder, unsigned char \*pSrcBuf, long nSrcBufSize, int  
nDstByteOrder, unsigned char \*pDstBuf, long nDstBufSize, int nFlag);

/\*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\_RsSetColByName(KCBPCLIHANDLE hHandle, char \*Name,  
char \*Vlu);

---

KCBPCLI_API	int	KCBPCLISTDCALL
KCBPCLI_RsSetVal(KCBPCLIHANDLE hHandle, int nColumnIndex, unsigned char *pValue, long nSize);		
KCBPCLI_API	int	KCBPCLISTDCALL
KCBPCLI_RsSetValByName(KCBPCLIHANDLE hHandle, char *szColumnName, unsigned char *pValue, long nSize);		
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_RsFetchRowScroll(KCBPCLIHANDLE hHandle, int nOrientation, int nOffset);		
KCBPCLI_API	int	KCBPCLISTDCALL
KCBPCLI_RsGetCol(KCBPCLIHANDLE hHandle, int Col, char *Vlu);		

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

```
KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_RsGetVal(KCBPCLIHANDLE hHandle, int nColumnIndex,
unsigned char **pValue, long *pSize);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_RsGetValByName(KCBPCLIHANDLE hHandle, char
*szColumnName, unsigned char **pValue, long *pSize);
```

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

```
KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_RsGetCoINum(KCBPCLIHANDLE hHandle, int *pnCols);
```

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

```
KCBPCLI_API          int          KCBPCLISTDCALL
KCBPCLI_RsGetTableCoINum(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\_GetCommArea(KCBPCLIHANDLE hHandle, char \*\*p);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_SetCommArea(KCBPCLIHANDLE hHandle, char \*);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_SetCliTimeOut(KCBPCLIHANDLE hHandle, int TimeOut);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_SetOption(KCBPCLIHANDLE hHandle, int nIndex, void  
\*pValue);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_GetOption(KCBPCLIHANDLE hHandle, int nIndex, void  
\*pValue);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_SetOptions(KCBPCLIHANDLE hHandle, int nIndex, void  
\*pValue, int nLen);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_GetOptions(KCBPCLIHANDLE hHandle, int nIndex, void  
\*pValue, int \*nLen);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_SetSystemParam(KCBPCLIHANDLE hHandle, int nIndex, char  
\*szValue);

KCBPCLI\_API                   int                   KCBPCLISTDCALL  
KCBPCLI\_GetSystemParam(KCBPCLIHANDLE hHandle, int nIndex,  
char \*szValue, int nLen);

```
/*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_SQLFetchScroll(KCBPCLIHANDLE hHandle, int nOrientation,
int nOffset);
```

```
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);
```

```
/*broadcast/notify*/
```

```
KCBPCLI_API int KCBPCLISTDCALL  
KCBPCLI_Notify(KCBPCLIHANDLE hHandle, char *szConnectionId,  
char *szData, int nDataLen, int nFlags);
```

```
KCBPCLI_API int KCBPCLISTDCALL  
KCBPCLI_Broadcast(KCBPCLIHANDLE hHandle, char *szMachineId,
```

```
char *szUserName, char *szConnectionId, char *szData, int nDataLen, int  
nFlags);
```

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

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

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_SSLVerifyCertPasswd(KCBPCLIHANDLE hHandle, char  
*szKeyFileName, char *szPassword);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_SSLModifyCertPasswd(KCBPCLIHANDLE hHandle, char  
*szKeyFileName, char *szOldPassword, char *szNewPassword);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_SSLEncrypt(KCBPCLIHANDLE hHandle, char  
*szKeyFileName, char *szPassword, unsigned char *pInput, int nInLen,  
unsigned char **pOutput, int *pnOutLen);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_SSLDecrypt(KCBPCLIHANDLE hHandle, char  
*szKeyFileName, char *szPassword, unsigned char *pInput, int nInLen,  
unsigned char **pOutput, int *pnOutLen);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_FreeMemory(KCBPCLIHANDLE hHandle, void *memblock);
```

```
KCBPCLI_API          int          KCBPCLISTDCALL  
KCBPCLI_SSLGetCertInfo(KCBPCLIHANDLE hHandle, char  
*szCertFileName, int nInfoType, int nInfoSubType, char *szBuf, int nLen);
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```





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  nProtocol;    /*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
```

为了保持系统的向下兼容性, 也保留了以前的 tagKCBPConnectOption。  
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.h"
int main(int argc,char *argv[])
{
    int nRet;
    char szInfo[20];
    KCBPCLIHANDLE hHandle;          /*KCBP 客户端句柄声明*/
    tagKCBPConnectOption stKCBPConnection;
    nRet = KCBPCLI_Init(&hHandle); /*初始化客户端句柄*/
    if(nRet != 0)
    {
        return 1;
    }
}
```

```

memset(&stKCBPConnection, 0, sizeof(stKCBPConnection));
strcpy(stKCBPConnection.szServerName, "KCBP1");
stKCBPConnection.nProtocal = 0;
strcpy(stKCBPConnection.szAddress, "192.168.40.65");
stKCBPConnection.nPort = 21000;
strcpy(stKCBPConnection.szSendQName, "req1b");
strcpy(stKCBPConnection.szReceiveQName, "ans1");
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT, &stKCBPConnection,
sizeof(stKCBPConnection));

/*连接 KCBP SERVER*/
nRet = KCBPCLI_ConnectServer(hHandle, "KCBP1", "TEST", "TEST");
if(nRet != 0)
{
    printf("KCBPCLI_ConnectServer return %d\n", nRet);
    goto LabelExit;
}

/*断开连接*/
nRet = KCBPCLI_DisConnect(hHandle);
if(nRet != 0)
{
    printf("KCBPCLI_DisConnect return %d\n", nRet);
    goto LabelExit;
}

LabelExit:
KCBPCLI_Exit(hHandle);
return 0;
}

```

**参 见：**该函数不推荐使用，请用 KCBPCLI\_SetOptions，选项 KCBP\_OPTION\_CONNECT。

## 2.4.2. 查询连接选项

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_GetConnectOption(KCBPCLIHANDLE hHandle, tagKCBPConnectOption  
\*pstKCBPConnection);

**输入参数：**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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：

参见：该函数已作废，请用 KCBPCLI\_GetOptions，选项 KCBP\_OPTION\_CONNECT。

### 2.4.3. 连接 KCBP SERVER

函数原型：KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_ConnectServer(KCBPCLIHANDLE hHandle, char \*ServerName, char \*UserName, char \*Password);

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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：KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_DisConnect(KCBPCLIHANDLE hHandle);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返回：0 成功，其它失败，返回信息由返回代码决定。

用法说明：发送 logout 请求到 KCBP Server 端，并关闭本地的 socket 句柄

### 2.4.5. 强制断开连接

函数原型：KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_DisConnectForce(KCBPCLIHANDLE hHandle);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返回：0 成功，其它失败，返回信息由返回代码决定。

用法说明：直接关闭本地的 socket 句柄。与 KCBPCLI\_DisConnect () 区别是 KCBPCLI\_DisConnect 向 Server 端发送一个断开消息，然后关闭 Socket；而 KCBPCLI\_DisConnectForce 不发送消息，直接关闭 Socket。注意，可以通过这个函数在另外一个线程中解除调用线程的阻塞状态。

## 2.4.6. 同步调用一个服务程序（提交方式）

函数原型：`KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_CallProgramAndCommit(KCBPCLIHANDLE hHandle, char *ProgramName);`

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2.4.7. 同步调用一个服务程序（不提交方式）

函数原型：`KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_CallProgram(KCBPCLIHANDLE hHandle, char *ProgramName);`

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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：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。

## 2.4.8. 提交事务

函数原型：`KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Commit(KCBPCLIHANDLE hHandle);`

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

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. 回滚事务

函数原型：`KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RollBack(KCBPCLIHANDLE hHandle);`

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返回：返回 0 表示成功，其它失败。

用法说明：只适用于 CallProgram。

## 2.4.10. 异步调用（提交方式）

函数原型：

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_ACallProgramAndCommit(KCBPCLIHANDLE hHandle, char *ProgramName, tagCallCtrl *ptagCallCtrl);
```

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *ProgramName	服务程序名称	Input
tagCallCtrl *ptagCallCtrl	控制参数	Input/Output

tagCallControl \*ptagCallCtrl 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	Ignore
Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Output
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号(存放当前连接项名称)	Output
int nExpiry	超时时间，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	回调函数指针	Reserved

返回：返回 0 表示成功，其它失败。

用法说明：与同步调用函数 KCBPCLI\_CallProgramAndCommit 区别在于 KCBPCLI\_CallProgramAndCommit 是同步调用，它一直等待服务端处理完成或超时返回，而 KCBPCLI\_ACallProgramAndCommit 是异步调用，它不用等服务端处理完成就直接返回，调用后，用户需要使用 KCBPCLI\_GetReply 查询服务端处理结果。

## 2.4.11. 异步调用（不提交方式）

函数原型：

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_ACallProgram(KCBPCLIHANDLE hHandle, char *ProgramName, tagCallCtrl *ptagCallCtrl);
```

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *ProgramName	服务程序名称	Input
tagCallCtrl *ptagCallCtrl	控制参数	Input/Output

tagCallControl \*ptagCallCtrl 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	Ignore

Char szMsgId[KCBP_MSGID_LEN+1]	消息号	Output
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号(存放当前连接项名称)	Output
Int nExpiry	超时时间, 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	回调函数指针	Reserved

**返回:** 返回 0 表示成功, 其它失败。

**用法说明:**

KCBPCLI\_ACallProgram 与同步调用函数 KCBPCLI\_CallProgram 区别在于 KCBPCLI\_CallProgram 是同步调用, 它一直等待服务端处理完成或超时返回, 而 KCBPCLI\_ACallProgram 是异步调用, 它不用等服务端处理完成就直接返回, 调用后, 用户需要使用 KCBPCLI\_GetReply 查询服务端处理结果。

KCBPCLI\_ACallProgram 与同步调用函数 KCBPCLI\_ACallProgramAndCommit 区别在于 KCBPCLI\_ACallProgram 采用客户端提交方式, 在处理完成后, 由用户决定调用 KCBPCLI\_Comint 提交或调用 KCBPCLI\_RollBack 回滚; 而 KCBPCLI\_ACallProgramAndCommit 采用服务端提交方式。

## 2.4.12. 查询异步调用结果

**函数原型:**

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetReply(KCBPCLIHANDLE hHandle, tagCallCtrl *ptagCallCtrl);
```

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagCallCtrl *ptagCallCtrl	控制参数	Input/Output

tagCallControl \*ptagCallCtrl 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	Input
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号(存放当前连接项名称)	Input
Int nExpiry	超时时间, 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	回调函数指针	Reserved

**返回:** 返回 0 表示成功, 其它失败。

**用法说明:**

当用户使用 KCBPCLI\_ACallProgram 或 KCBPCLI\_ACallProgramAndCommit 发出一个异步调用请求后, 需要使用 KCBPCLI\_GetReply 查询服务端处理结果。ptagCallCtrl 的输入来源于异步调用函数的输出参数。此外, ptagCallCtrl->szMsgId 参数有一个特殊用法, 当它是"0"时(简称 0 方式), KCBPCLI\_GetReply 会取得任何请求的应答, 包括任何同步调用方式的应

答、异步调用的应答，0 方式是一种高效的异步方式。注意，0 方式也是一把双刃剑，使用得当能给编程带来很大的灵活性，使用不当可能带来混乱。

KCBPCLI\_ACallProgramAndCommit、KCBPCLI\_ACallProgram、KCBPCLI\_GetReply、KCBPCLI\_Cancel 的 tagCallCtrl \*ptagCallCtrl 几个参数含义：

int nFlags; //输入：报文分组接收标志，0 表示接收全部应答返回，其他表示接收一组应答返回；输出：报文接收状态标志

char szId[KCBP\_SERIAL\_LEN + 1]; //输入：KCBPCLI 连接名称，输出：发布、订阅调用返回订阅号，其他业务调用时返回 KCBPCLI 连接名称。

char szMsgId[KCBP\_MSGID\_LEN + 1]; //输入：发布订阅请求可以传递客户唯一标志，其它调用时需要设置为""；输出：消息号

char szCorrId[KCBP\_CORRID\_LEN + 1]; //输入：KCBPCLI\_GetReply 时设置为"0"，输出：发布、订阅调用返回 KCBPCLI 连接名称，其他业务调用返回相关消息号

#### 使用 0 方式注意以下事项：

1. 2007 年 12 月以后发布的 KCBP 客户端不再要求非 0 方式独享队列。
2. 当与非 0 方式的 GetReply 共用队列时，务必在非 0 方式连接前需要调用 KCBPCLI\_SetOption 设置 KCBP\_OPTION\_GROUPID 参数为空字符串""，否则可能导致应答都被 0 方式取走而非 0 方式取不到应答。
3. 如果客户开发的应用程序（调用 KCBP 客户端 API）运行多个实例（进程），要求每个实例单独使用自己的工作目录。

## 2.4.13. 取消异步调用

### 函数原型：

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_Cancel(KCBPCLIHANDLE hHandle, tagCallCtrl *ptagCallCtrl);
```

### 参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
tagCallCtrl *ptagCallCtrl	控制参数	Input/Output

tagCallControl \*ptagCallCtrl 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	Input
Char szCorrId[KCBP_CORRID_LEN+1]	相关消息号(存放当前连接项名称)	Input
Int nExpiry	超时时间，以秒为单位	Input
Int nPriority	消息的优先级	Ignore
time_t tTimeStamp	受理时间	Ignore
KCBPCLI_Notify_t lpfnCallback	回调函数指针	Reserved

返 回：返回 0 表示成功，其它失败。

用法说明：



当用户使用 KCBPCLI\_ACallProgram 或 KCBPCLI\_ACallProgramAndCommit 发出一个异步调用请求后，可以使用 KCBPCLI\_Cancel 函数通知服务端终止调用，pTagCallCtrl 的输入来源于内容异步调用函数的输出参数。

**注意：**KCBPCLI\_Cancel 不会对服务端已经处理完成的交易结果产生影响。

## 2.4.14. 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。

## 2.4.15. 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。

## 2.4.16. 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。

## 2.4.17. 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。

## 2.4.18. 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。

## 2.4.19. 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。

## 2.4.20. 清除公共数据区

函数原型: KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_BeginWrite(KCBPCLIHANDLE hHandle);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返 回: 0 成功, 其它失败

用法说明: 表示开始写通信的公共数据区, 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。

## 2.4.21. 根据键名 (KEYNAME) 设置键值 (VALUE)

函数原型: KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_SetValue(KCBPCLIHANDLE hHandle, char \*KeyName, char \*Vlu);

输入参数:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

Value 字符串长度限制在 30KBytes，如超过限制，请使用 KCBPCLI\_SetVal 设置，并指定长度。

## 2.4.22. 根据键名 (KEYNAME) 设置指定长度的键值 (VALUE)

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_SetVal(KCBPCLIHANDLE hHandle, char \*szKeyName, unsigned char \*pValue, long nSize);

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *szKeyName	关键字名称	Input
unsigned char *pValue	关键字值输入缓冲区	Input
long nSize	关键字值输入缓冲区长度	Input

返回：0 成功，其它失败

用法说明：

该函数既可设置指定长度的二进制缓冲区，也可以设置字符串缓冲区。

与此函数相关的 SetValue，只能设置字符串缓冲区。

## 2.4.23. 根据键名 (KEYNAME) 取键值 (VALUE)

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetValue(KCBPCLIHANDLE hHandle, char \*KeyName, char \*Vlu, int Len);

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2.4.24. 根据键名 (KEYNAME) 取键值 (VALUE) 及键值长度

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetVal(KCBPCLIHANDLE hHandle, char \*szKeyName, unsigned char \*\*pValue, long \*pSize);

输入参数:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *szKeyName	关键字名称	Input
unsigned char **pValue	输出缓冲区指针	Output
long *pSize	输出缓冲区长度	Output

返 回: 0 成功, 其它失败

用法说明:

该函数既可获取二进制缓冲区, 也可以获取字符串缓冲区。

该函数返回的缓冲区指针, 由 KCBP 客户端接口自动维护, 用户不需要它执行释放操作。用户应该在 GetVal 后立即处理该缓冲区中的数据 (比如复制到其他地方)。

与该函数类似的是 GetValue, 但 GetValue 只能用来获得字符串, 不能获得二进制 Buffer, 并且 GetValue 的缓冲区由用户指定, 无边界控制。

例 子:

```
char *pOut = NULL;
long lOut = 0;
nRet = KCBPCLI_GetVal(hHandle, "image", &pOut, &lOut);
if(nRet == 0 && pOut != NULL)
{
//display ther buffer content with Hex format
for(int i=0; i< lOut, i++)
{
printf("%x,", pOut[i]);
}
}
}
```

## 2.4.25. 创建结果集

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsCreate(KCBPCLIHANDLE hHandle, char \*Name, int ColNum, char \*pColInfo);

输入参数:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
Char *Name	结果集名称	Input
int CoNum	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。

RsCreate 用于创建第 1 个结果集。

结果集列名称表格式如"col1,col2,col3"。 KCBP V3.0 Build 120321 以前版本，列名总长度限制为 2K。KCBP V3.0 Build 120321 以后版本，列名长度无限制。

注意：结果集字符串行总长度（所有列长度综合，减去列分隔符）限制 4K，单个字符串长度限制也是 4K。如果行总长度超过 4K，对于长列，要使用二进制列。

## 2. 4. 26. 增加结果集

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsNewTable(KCBPCLIHANDLE hHandle, char \*Name, int ColNum, char \*pColInfo);

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2. 4. 27. 使公共数据区的结果集增加一行

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsAddRow(KCBPCLIHANDLE hHandle);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返回：0 成功，其它失败

用法说明：

## 2. 4. 28. 根据列号设置结果集中当前行的某一列值

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsSetCol(KCBPCLIHANDLE hHandle, int Col, char \*Vlu);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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. 29. 根据列名设置结果集中当前行的某一列值

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsSetColByName(KCBPCLIHANDLE hHandle, char \*Name, char \*Vlu);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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。

## 2. 4. 30. 根据列号设置当前行的某一列值为指定长度的缓冲区

函数原型:

KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsSetVal(KCBPCLIHANDLE hHandle, int nColumnIndex, unsigned char \*pValue, long nSize);

参数说明:

KCBPCLIHANDLE hHandle	客户端句柄	Input
Int nColumnName	列编号	Input
unsigned char *pValue	输入缓冲区	Input
long nSize	输入缓冲区长度	Input

返 回: 0 成功, 其它失败

用法说明:

该函数既可设置指定长度的二进制缓冲区, 也可以设置字符串缓冲区。

与此函数相关的 RsSetValue, 只能设置字符串缓冲区。

## 2. 4. 31. 根据列名设置当前行的某一列值为指定长度的缓冲区

函数原型:

KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsSetValByName(KCBPCLIHANDLE hHandle, char \*szColumnName, unsigned char \*pValue, long nSize);

参数说明:

KCBPCLIHANDLE hHandle	客户端句柄	Input
char * szColumnName	列名称	Input
unsigned char *pValue	输入缓冲区	Input
long nSize	输入缓冲区长度	Input

返 回: 0 成功, 其它失败

用法说明:

该函数既可设置指定长度的二进制缓冲区, 也可以设置字符串缓冲区。

与此函数相关的 RsSetValue, 只能设置字符串缓冲区。

例 子:

unsigned char \*pOut;



```

long lOut;
FILE *fp = fopen("a.gif", "rb");
if(fp != NULL)
{
    int nFileLength = _filelength(_fileno(fp));
    char *filebuf = (char *)malloc(nFileLength);
    if(filebuf != NULL)
    {
        int nBytes = fread(filebuf, 1, nFileLength, fp);
        int nRet = KCBPCLI_RsSetVal(hHandle, 3, (unsigned char *)filebuf,
nFileLength);
        if(nRet != 0)
        {
            printf("KCBPCLI_RsSetVal return %d\n", nRet);
            return nRet;
        }
        nRet = KCBPCLI_RsGetVal(pKCBPCLI, 3, &pOut, &lOut);
        if(nRet != 0)
        {
            printf("KCBPCLI_RsGetVal return %d\n", nRet);
            return nRet;
        }
        if(memcmp(filebuf, pOut, lOut) != 0)
        {
            printf("input != outout \n");
            return nRet;
        }
        free(filebuf);
    }
    fclose(fp);
}

```

## 2. 4. 32. 在公共数据区的结果集中存储当前行

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsSaveRow(KCBPCLIHANDLE hHandle);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

**返 回:** 0 成功, 其它失败

**用法说明:**

## 2. 4. 33. 打开结果集

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsOpen(KCBPCLIHANDLE hHandle);

**参数说明：**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2. 4. 34. 获取当前结果集名称

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetCursorName(KCBPCLIHANDLE hHandle, char \*pszCursorName, int nLen);

**参数说明：**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2. 4. 35. 获取当前结果集的全部列名称

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetColNames(KCBPCLIHANDLE hHandle, char \*pszInfo, int nLen);

**参数说明：**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2. 4. 36. 获取当前结果指定列的名称

**函数原型：** KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetColName(KCBPCLIHANDLE hHandle, int nCol, char \*pszName, int nLen);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
nCol	列序号	Input
pszName	列名	output
nLen	输出缓冲区长度	Input

返回: 0 成功, 列名放在 pszName 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。

## 2.4.37. 获取公共数据区的当前结果集的行数

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetRowNum(KCBPCLIHANDLE hHandle, int \*pnRows);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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:

## 2.4.38. 获取公共数据区的当前结果集的列数

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetColNum(KCBPCLIHANDLE hHandle, int \*pnCols);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int *pnCols	结果集列数	output

返回: 0 成功, 列数放在 pnCols 中; 其它失败。

用法说明:

## 2.4.39. 获取公共数据区的指定结果集的行数

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetTableRowNum(KCBPCLIHANDLE hHandle, int nt, int \*pnRows);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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:

## 2. 4. 40. 获取公共数据区的指定结果集的列数

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetTableColNum(KCBPCLIHANDLE hHandle, int nt, int \*pnCols);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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:

## 2. 4. 41. 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: KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsFetchRow(KCBPCLIHANDLE hHandle);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

返 回: 0 成功, 其它失败

用法说明:

## 2. 4. 42. 从结果集的当前行的某一列取值 (根据列序号)

函数原型: KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsGetCol(KCBPCLIHANDLE hHandle, int Col, char \*Vlu);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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. 43. 从结果集的当前行的某一列取值 (根据列名)

函数原型: KCBPCLI\_API int KCBPCLISTDCALL

KCBPCLI\_RsGetColByName(KCBPCLIHANDLE hHandle, char \*KeyName, char \*Vlu);

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *KeyName	列名	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. 44. 获得当前行的某一列的缓冲区和长度（根据列序号）

函数原型：

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetVal(KCBPCLIHANDLE hHandle, int nColumnIndex, unsigned char **pValue, long *pSize);
```

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int nColumnIndex	列编号	Input
unsigned char **pValue	输出缓冲区指针	Output
long *pSize	输出缓冲区长度	Output

返回：0 成功，其它失败

用法说明：

该函数既可获取二进制缓冲区，也可以获取字符串缓冲区。

该函数返回的缓冲区指针，由 KCBP 客户端接口自动维护，用户不需要它执行释放操作。用户应该在 RsGetVal 后立即处理该缓冲区中的数据（比如复制到其他地方）。

与该函数类似的是 RsGetCol，但 RsGetCol 只能用来获得字符串，不能获得二进制 Buffer，并且 RsGetCol 的缓冲区由用户指定，无边界控制。

## 2. 4. 45. 获得当前行的某一列的缓冲区和长度（根据列名称）

函数原型：

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_RsGetValByName(KCBPCLIHANDLE hHandle, char *szColumnName, unsigned char **pValue, long *pSize);
```

输入参数：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char *szColumnName	列名称	Input
unsigned char **pValue	输出缓冲区指针	Output
long *pSize	输出缓冲区长度	Output

返回：0 成功，其它失败

用法说明：

该函数既可获取二进制缓冲区，也可以获取字符串缓冲区。

该函数返回的缓冲区指针，由 KCBP 客户端接口自动维护，用户不需要它执行释放操作。用户应该在 RsGetVal 后立即处理该缓冲区中的数据（比如复制到其他地方）。

与该函数类似的是 RsGetColByName，但 RsGetColByName 只能用来获得字符串，不能获得二进制 Buffer，并且 RsGetColByName 的缓冲区由用户指定，无边界控制。

## 2. 4. 46. 查询后续结果集

函数原型：KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsMore(KCBPCLIHANDLE

hHandle);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	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。

## 2.4.47. 关闭结果集

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_RsClose(KCBPCLIHANDLE hHandle);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

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。

## 2.4.48. 获取公共数据区当前长度

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetCommLen(KCBPCLIHANDLE hHandle, int \*pnLen);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int *pnLen	缓冲区长度	Output

返回: 0 成功, 公共数据区当前长度放在 pnLen 中; 其它失败。

**用法说明:**

## 2.4.49. 设置调用超时时间

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_SetCliTimeOut(KCBPCLIHANDLE hHandle, int TimeOut);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int TimeOut	调用超时, 单位秒	Input

返回: 0 成功, 其它失败。

**用法说明:** 该函数已废弃, 请使用 KCBPCLI\_SetOption。

## 2.4.50. 返回错误码和错误信息

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetErr(KCBPCLIHANDLE hHandle, int \*pErrCode, char \*ErrMsg);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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。

## 2.4.51. 返回错误码

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetErrorCode(KCBPCLIHANDLE hHandle, int \*pnErrno);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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。

## 2.4.52. 返回错误信息

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetErrorMsg(KCBPCLIHANDLE hHandle, char \*szError);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
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。

## 2.4.53. 查询 API 当前版本号

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetVersion(KCBPCLIHANDLE hHandle, int \*pnVersion);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input

int *pnVersion	版本号	Output
----------------	-----	--------

返 回：0 成功，其它失败。

用法说明：

## 2.4.54. 查询系统参数

函数原型：KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetOptions(KCBPCLIHANDLE hHandle, int nIndex, void \*pValue, int \*nLen);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int nIndex	选项编号	Input
void *pValue	输出缓冲区	Output
int *nLen	输出缓冲区长度	Input/Output

nIndex 有效值在 KCBPCLI.H 中定义，目前有以下几项：

名称	值	说明
KCBP_OPTION_CONNECT	0	查询连接参数
KCBP_OPTION_TIMEOUT	1	超时
KCBP_OPTION_TRACE	2	通讯数据包跟踪
KCBP_OPTION_CRYPT	3	加密方式
KCBP_OPTION_COMPRESS	4	压缩方式
KCBP_OPTION_PARITY	5	数据报文校验
KCBP_OPTION_SEQUENCE	6	报文顺序检查
KCBP_OPTION_CURRENT_CONNECT	7	当前连接
KCBP_OPTION_CONNECT_HANDLE	8	缺省连接句柄数目
KCBP_OPTION_CONFIRM	9	消息确认
KCBP_OPTION_NULL_PASS	10	SetValue 是否接受空字符串输入
KCBP_OPTION_TIME_COST	11	统计调用时间开关，系统保留
KCBP_OPTION_CONNECT_EX	12	连接参数
KCBP_OPTION_CURRENT_CONNECT_EX	13	当前连接参数
KCBP_OPTION_AUTHENTICATION	14	权限验证方式
KCBP_OPTION_GROUPID	15	消息组号，用来识别同步、异步调用者。
KCBP_OPTION_USE_GROUPID	16	消息组编号启用标志，1 启用，0 不用。缺省是 1。
KCBP_OPTION_RECEIVE_PACKET_BY_GROUP	17	分组接收报文标志，1 分组，2 不分组。缺省 1。
KCBP_OPTION_XA	18	设置 KCBPXA 组件的参数信息。
KCBP_OPTION_CURRENT_XA	19	获取当前 KCBPXA 信息。
KCBP_OPTION_CONNECTION_POOL	20	连接池中连接数。缺省 0。

返 回：0 成功，其它失败。

用法说明：

### 1. KCBP\_OPTION\_CONNECT

查询连接参数，KCBP 客户端可以设置最多 16 个连接项。多个连接项的意义在于：当调用 KCBPCLI\_Connect 函数时，KCBP 客户端内部会随机选取其中一项进行连接，如果连接失败，KCBP 客户端会自动选取其他连接项重试，直到连接成功或全部失败。



用户可以通过 `KCBP_OPTION_CONNECT` 来查询全部连接项，每个连接项定义结构如下：

```
typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX + 1];
    int nProtocol;
    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;
```

`KCBPCLI_GetConnectOption` 与此区别在于 `KCBPCLI_GetConnectOption` 只取第一个连接参数。

## 2. KCBP\_OPTION\_TIMEOUT

查询超时时间设置，单位秒。例子：

```
int nValue;
int nBytes;
nBytes = sizeof(int);
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_TIMEOUT, &nValue, &nBytes);
```

## 3. KCBP\_OPTION\_TRACE

查询通讯数据包跟踪设置，`nValue == 0` 表示不跟踪，其他表示跟踪。当跟踪开关打开时，在当前目录下创建报文数据记录文件，该文件以 .dat 为后缀，文件名是调用线程标号。

```
int nValue;
int nBytes;
nBytes = sizeof(int);
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_TRACE, &nValue, &nBytes);
```

## 4. KCBP\_OPTION\_CRYPT

查询加密方式设置，`nValue == 0` 表示不加密，其他表示当前使用的加密方法。

```
int nValue;
int nBytes;
nBytes = sizeof(int);
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_CRYPT, &nValue, &nBytes);
```

## 5. KCBP\_OPTION\_COMPRESS

查询压缩方式设置，`nValue == 0` 表示不压缩，其他表示当前使用的压缩方法。

```
int nValue;
int nBytes;
nBytes = sizeof(int);
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_COMPRESS, &nValue, &nBytes);
```

注意，当使用压缩方法时，可以提高数据在低速网络上的传输速率，但会消耗一定的 CPU，尤其是 Server 端的 CPU。

## 6. KCBP\_OPTION\_PARITY

查询数据报文校验方式设置，nValue ==0 表示不检查，其他表示检查。

```
int nValue;
```

```
int nBytes;
```

```
nBytes = sizeof(int);
```

```
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_PARITY, &nValue, &nBytes);
```

注意，当使用检查方式时，可以保证数据包的正确性，但会消耗一定的 CPU。

## 7. KCBP\_OPTION\_SEQUENCE

查询报文顺序检查方式设置，nValue ==0 表示不检查，其他表示检查。

```
int nValue;
```

```
int nBytes;
```

```
nBytes = sizeof(int);
```

```
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_SEQUENCE, &nValue, &nBytes);
```

注意，当使用检查方式时，可以保证数据包的正确性，但会消耗一定的 CPU。

## 8. KCBP\_OPTION\_CURRENT\_CONNECT

查询当前连接参数设置。当设置了多个 KCBP 连接项时，KCBP 客户端随即选择一个连接项进行连接，可以用这种方式查询当前连接项。

```
tagKCBPConnectOption stConnectOption;
```

```
int nBytes;
```

```
nBytes = sizeof(tagKCBPConnectOption);
```

```
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_CURRENT_CONNECT,  
                  &stConnectOption, &nBytes);
```

## 9. KCBP\_OPTION\_CONNECT\_HANDLE

查询缺省连接句柄数目设置，nValue ==1 表示接收和发送使用同一连接，2 表示接收和发送各自建立一个连接。

```
int nValue;
```

```
int nBytes;
```

```
nBytes = sizeof(int);
```

```
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_CONNECT_HANDLE,  
                  &nValue, &nBytes);
```

## 10. KCBP\_OPTION\_CONFIRM

消息确认设置，nValue ==0 表示不采用确认方式发送消息，其他表示要求确认。如果使用不确认方式，由于减少了与 KCXP 通讯交互次数，可提高调用效率，这点在广域网上有显著效果。这项功能要求使用 2007 年 12 月以后发布的 KCXP 服务端和客户端。

```
int nValue;
```

```
int nBytes;
```

```
nBytes = sizeof(int);
```

```
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_CONFIRM, &nValue, &nBytes);
```

## 11. KCBP\_OPTION\_NULL\_PASS

查询 SetValue 是否接受空字符串输入, nValue ==0 表示不接受, 其他表示接受。

```
int nValue;
int nBytes;
nBytes = sizeof(int);
KCBPCLI_GetOptions(hHandle, KCBP_OPTION_NULL_PASS, &nValue, &nBytes);
```

## 12. KCBP\_OPTION\_CONNECT\_EX

与 KCBP\_OPTION\_CONNECT 区别是使用 tagKCBPConnectOptionEx 结构, 增加了 SSL 和 PROXY 选项。

```
typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX+1];/*用户自定义的 KCBP 服务器名称*/
    int nProtocol; /*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*/
    char szProxy[KCBP_PROXY_MAX + 1]; /*代理服务器参数*/
    char szSSL[KCBP_SSL_MAX + 1]; /*SSL 参数*/
}tagKCBPConnectOptionEx;
```

PROXY 采用 URL 格式: 协议://用户名:密码@服务器:端口, 如

<https://192.168.40.65:80>

socks4://192.168.40.65:1080

socks5://guest:password@192.168.40.65:1080。

SSL 格式: 状态,根证书,证书,密码,算法表,加密传输,主动发起握手, 如:

YES,ssccCA.pem, client.pem, client, ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH, YES, YES

## 13. KCBP\_OPTION\_CURRENT\_CONNECT\_EX

与 KCBP\_OPTION\_CURRENT\_CONNECT 区别是使用 tagKCBPConnectOptionEx 结构, 增加了 SSL 和 PROXY 选项。

## 14. KCBP\_OPTION\_AUTHENTICATION

查询设置权限验证方式, 1 表示 KCBPCLI\_ConnectServer 调用时验证用户权限; 0 表示 KCBPCLI\_ConnectServer 调用时不验证用户权限; 系统缺省值是 1。

### 1. KCBP\_OPTION\_USE\_GROUPID

用来控制发送请求报文时, 是否传递 GROUPID, 0 表示不传递 GROUPID; 1 异步调用时传递 GROUPID, 这是缺省值; 2 表示同步调用时传递; 3 表示同步、异步皆传递。

采用分组读应答报文, 可以防止多个客户端读同一个应答队列时取到不属于自己的应答。

GROUPID 用来区分应答报文的对应请求方式, 当异步调用 GET 0 方式取应答与同步调用并存时, 设置选项值为 1, 可确保同步请求的应答不被 GET 0 取走; 此外, GROUPID 对发布

订阅的取应答行为也具有同样的控制作用。

## 2. KCBP\_OPTION\_RECEIVE\_PACKET\_BY\_GROUP

这个选项用来控制 KCBP 客户端读取消息时，是否采用分组（段）读取方式。缺省采用。

如果启用，客户端读取一段报文，供客户端处理，有需要时再读取下一段。如果不用，客户端将读取所有应答报文到内存中。当存在大量应答数据时，采用分段处理方式能，让客户端分段处理，界面上可获得较快响应。

对于报文中存在大对象数据的情况，KCBP 服务器向客户端发送包含大对象数据的报文时，不会设置分组标志，这时，不论客户端是否启用分组读取，由于没有分组标志，客户端一次读取所有报文。

## 3. KCBP\_OPTION\_XA

KCBP 客户端可使用 KDMID1PCFactory，二者结合，可实现 KCBP 客户端的业务流控制，提供通用方法实现与其他系统集成的目的，降低与其他系统集成的复杂度，功能强大。

KCBPCLI\_SetOption 可以通过选项 KCBP\_OPTION\_XA 来设置 KCBPXA 连接参数，参数格式如下：

```
/* struct for kcbpxa's parameter */
typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX + 1]; //XA name
    int nProtocol;
    char szSwitchLoadFile[KCBP_XA_PATH_SIZE + 1];
    char szXAEntry[KCBP_XA_ENTRY_SIZE + 1];
    char szXAOpen[KCBP_XA_OPEN_SIZE + 1];
    char szXAClose[KCBP_XA_CLOSE_SIZE + 1];
    char szXASerial[KCBP_XA_SERIAL_SIZE + 1];
    char szXAOption[KCBP_XA_OPTION_SIZE];
} tagKCBPXAConnectOption;
```

为便于调试，KCBPCP 增加对使用 KDMID1PCFactory 的 KCBPCLi 调用支持，可以通过命令设置连接参数。例子如下：

```
set kcbpxa
name=kdmid1pc;open=ip=127.0.0.1,port=28946,adapter=kdmidapi.dll,rule=rule32.xml,connecttime
out=5,userid=9999,password=3Mc+w9uU5IM=,timeout=30;option=
```

## 4. KCBP\_OPTION\_CURRENT\_XA

设置/读取 KCBPXA 连接参数。通过一个 tagKCBPXAConnectOption 结构的指针传递参数。

## 5. KCBP\_OPTION\_CONNECTION\_POOL

设置/取得连接池中的连接数。

```
int n Value;
```

```
KCBPCLI_SetOption(handle, KCBP_OPTION_CONNECTION_POOL, &value);
```

## 2.4.55. 设置系统参数

**函数原型:** KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_SetOptions(KCBPCLIHANDLE hHandle, int nIndex, void \*pValue, int nLen);

**参数说明:**

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int nIndex	选项编号	Input
Void *pValue	输入缓冲区	Input
int nLen	输入缓冲区长度	Input

nIndex 有效值在 KCBPCLI.H 中定义，目前有以下几项：

名称	值	说明
KCBP_OPTION_CONNECT	0	连接参数
KCBP_OPTION_TIMEOUT	1	超时
KCBP_OPTION_TRACE	2	通讯数据包跟踪
KCBP_OPTION_CRYPT	3	加密方式
KCBP_OPTION_COMPRESS	4	压缩方式
KCBP_OPTION_PARITY	5	数据报文校验
KCBP_OPTION_SEQUENCE	6	报文顺序检查
KCBP_OPTION_CURRENT_CONNECT	7	当前连接
KCBP_OPTION_CONNECT_HANDLE	8	缺省连接句柄数目
KCBP_OPTION_CONFIRM	9	消息确认
KCBP_OPTION_NULL_PASS	10	SetValue 是否接受空字符串输入
KCBP_OPTION_TIME_COST	11	统计调用时间开关，系统保留
KCBP_OPTION_CONNECT_EX	12	连接参数
KCBP_OPTION_CURRENT_CONNECT_EX	13	当前连接参数
KCBP_OPTION_AUTHENTICATION	14	权限验证方式
KCBP_OPTION_GROUPID	15	消息组号，用来识别同步、异步调用者。
KCBP_OPTION_USE_GROUPID	16	消息组编号启用标志，1 启用，0 不用。缺省是 1。
KCBP_OPTION_RECEIVE_PACKET_BY_GROUP	17	分组接收报文标志，1 分组，2 不分组。缺省 1。
KCBP_OPTION_XA	18	设置 KCBPXA 组件的参数信息。
KCBP_OPTION_CURRENT_XA	19	获取当前 KCBPXA 信息。
KCBP_OPTION_CONNECTION_POOL	20	连接池中连接数。缺省 0。

**返回:** 0 成功，其它失败。

**用法说明:**

### 6. KCBP\_OPTION\_CONNECT

设置连接参数，KCBP 客户端可以设置最多 16 个连接项。多个连接项的意义在于：当调用 KCBPCLI\_Connect 函数时，KCBP 客户端内部会随机选取其中一项进行连接，如果连接失败，KCBP 客户端会自动选取其他连接项重试，直到连接成功或全部失败。

用户可以通过 KCBP\_OPTION\_CONNECT 来设置全部连接项，每个连接项定义结构如下：

typedef struct

```
{
    char szServerName[KCBP_SERVERNAME_MAX+1];/*用户自定义的 KCBP 服务器名称*/
    int nProtocol; /*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;

KCBPCLI\_SetConnectOption 与此区别在于 KCBPCLI\_SetConnectOption 只设置第一个连接参数。

例子 1，同组连接：

```

tagKCBPConnectOption KCBPConnectOption[2];
memset(KCBPConnectOption, 0, sizeof(KCBPConnectOption));
int nLen = sizeof(KCBPConnectOption);

/*set main connection parameter*/
strcpy(KCBPConnectOption[0].szServerName, "1");
KCBPConnectOption[0].nProtocal = 0;
strcpy(KCBPConnectOption[0].szAddress, "10.100.218.200");
KCBPConnectOption[0].nPort = 21000;
strcpy(KCBPConnectOption[0].szSendQName, "req1");
strcpy(KCBPConnectOption[0].szReceiveQName, "ans1");
strcpy(KCBPConnectOption[0].szReserved, "KCBPGRP1"); /* szReserved 存放组名称*/

/*set backup connection parameter*/
strcpy(KCBPConnectOption[1].szServerName, "2");
KCBPConnectOption[1].nProtocal = 0;
strcpy(KCBPConnectOption[1].szAddress, "10.100.218.201");
KCBPConnectOption[1].nPort = 21000;
strcpy(KCBPConnectOption[1].szSendQName, "req1");
strcpy(KCBPConnectOption[1].szReceiveQName, "ans1");
strcpy(KCBPConnectOption[1].szReserved, " KCBPGRP1"); /* szReserved 存放组名称*/

nLen = sizeof(KCBPConnectOption);
nRet = KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT, KCBPConnectOption,
nLen);
if (nRet !=0)
{
printf("KCBPCLI_SetOptions fail, return code =%d", nRet);
return nRet;
}
nRet = KCBPCLI_ConnectServer(hHandle, "KCBPGRP1", "9999", "888888"); /*使用组名*/
/*.....*/

```

例子 2，同名连接：

```

tagKCBPConnectOption KCBPConnectOption[2];
memset(KCBPConnectOption, 0, sizeof(KCBPConnectOption));

```

```
int nLen = sizeof(KCBPConnectOption);

/*set main connection parameter*/
strcpy(KCBPConnectOption[0].szServerName, "KCBP1");
KCBPConnectOption[0].nProtocol = 0;
strcpy(KCBPConnectOption[0].szAddress, "10.100.218.200");
KCBPConnectOption[0].nPort = 21000;
strcpy(KCBPConnectOption[0].szSendQName, "req1");
strcpy(KCBPConnectOption[0].szReceiveQName, "ans1");
strcpy(KCBPConnectOption[0].szReserved, "");

/*set backup connection parameter*/
strcpy(KCBPConnectOption[1].szServerName, "KCBP1");
KCBPConnectOption[1].nProtocol = 0;
strcpy(KCBPConnectOption[1].szAddress, "10.100.218.201");
KCBPConnectOption[1].nPort = 21000;
strcpy(KCBPConnectOption[1].szSendQName, "req1");
strcpy(KCBPConnectOption[1].szReceiveQName, "ans1");
strcpy(KCBPConnectOption[1].szReserved, "");

nLen = sizeof(KCBPConnectOption);
nRet = KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT, KCBPConnectOption,
nLen);
if (nRet !=0)
{
printf("KCBPCLI_SetOptions fail, return code =%d", nRet);
return nRet;
}
nRet = KCBPCLI_ConnectServer(hHandle, "KCBP1", "9999", "888888"); /*使用组名*/
/*.....*/
```

#### 7. KCBP\_OPTION\_TIMEOUT

设置超时时间设置，单位秒。例子：

```
int nValue;
int nBytes;
nValue = 30; /*30 second*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_TIMEOUT, &nValue, nBytes);
```

#### 8. KCBP\_OPTION\_TRACE

设置通讯数据包跟踪设置，nValue ==0 表示不跟踪，其他表示跟踪。当跟踪开关打开时，在当前目录下创建报文数据记录文件，该文件以.dat 为后缀，文件名是调用线程标号。

```
int nValue;
int nBytes;
```

```
nValue = 1; /*打开跟踪开关*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_TRACE, & nValue, nByte);
```

#### 9. KCBP\_OPTION\_CRYPT

设置加密方式设置，nValue ==0 表示不加密，其他表示当前使用的加密方法。  
注意，加密方式 3 与压缩方式 3 可同时使用，但不能与其他压缩方式同时使用。

```
int nValue;
int nBytes;
nValue = 3; /*设置加密方式为 3*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CRYPT, & nValue, nByte);
```

#### 10. KCBP\_OPTION\_COMPRESS

设置压缩方式设置，nValue ==0 表示不压缩，其他表示当前使用的压缩方法。  
注意，压缩方式 3 与加密方式 3 可同时使用，但不能与其他加密方式同时使用。

```
int nValue;
int nBytes;
nValue = 3; /*设置压缩方式为 3*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_COMPRESS, & nValue, nByte);
```

注意，当使用压缩方法时，可以提高数据在低速网络上的传输速率，但会消耗一定的 CPU，尤其是 Server 端的 CPU。

#### 11. KCBP\_OPTION\_PARITY

设置数据报文校验方式设置，nValue ==0 表示不检查，其他表示检查。

```
int nValue;
int nBytes;
nValue = 1; /*检查数据报文校验值*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_PARITY, & nValue, nByte);
```

注意，当使用检查方式时，可以保证数据包的正确性，但会消耗一定的 CPU。

#### 12. KCBP\_OPTION\_SEQUENCE

设置报文顺序检查方式设置，nValue ==0 表示不检查，其他表示检查。

```
int nValue;
int nBytes;
nValue = 1; /*检查报文顺序*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_SEQUENCE, & nValue, nByte);
```

注意，当使用检查方式时，可以保证数据包的正确性，但会消耗一定的 CPU。

#### 13. KCBP\_OPTION\_CURRENT\_CONNECT

设置当前连接参数设置。当设置了多个 KCBP 连接项时，KCBP 客户端随即选择一个连接



项进行连接，可以用这种方式更改当前连接项。

```
tagKCBPConnectOption stKCBPConnectOption;
int nBytes;

sprintf(stKCBPConnectOption.szServerName, "KCBP0001", i);
stKCBPConnectOption.nProtocol = 0;
strcpy(stKCBPConnectOption.szAddress, "10.100.218.200");
stKCBPConnectOption.nPort = 21000;
strcpy(stKCBPConnectOption.szSendQName, "req1");
strcpy(stKCBPConnectOption.szReceiveQName, "ans1");
strcpy(stKCBPConnectOption.szReserved, "");

nBytes = sizeof(tagKCBPConnectOption);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CURRENT_CONNECT,
                  & stKCBPConnectOption, nByte);
```

#### 14. KCBP\_OPTION\_CONNECT\_HANDLE

设置缺省连接句柄数目设置，nValue ==1 表示接收和发送使用同一连接，2 表示接收和发送各自建立一个连接。

```
int nValue;
int nBytes;
nValue = 1; /*接收和发送使用同一连接句柄*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT_HANDLE,
                  & nValue, nByte);
```

#### 15. KCBP\_OPTION\_CONFIRM

设置消息确认方式，nValue ==0 表示不采用确认方式发送请求，其他表示要求确认。由于减少了通讯环节，非确认方式速度较快，适合网络情况较好的情况。KCBP 客户端缺省采用确认方式发送请求。

```
int nValue;
int nBytes;
nValue = 1; /*采用确认方式*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONFIRM, & nValue, nByte);
```

#### 16. KCBP\_OPTION\_NULL\_PASS

设置 SetValue 是否接受空字符串输入，nValue ==0 表示不接受，其他表示接受。

```
int nValue;
int nBytes;
nValue = 1; /*支持""字符串输入*/
nBytes = sizeof(int);
KCBPCLI_SetOptions(hHandle, KCBP_OPTION_NULL_PASS, & nValue, nByte);
```

## 17. KCBP\_OPTION\_CONNECT\_EX

与 KCBP\_OPTION\_CONNECT 区别是使用 tagKCBPConnectOptionEx 结构，增加了 SSL 和 PROXY 选项。

```
typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX+1];/*用户自定义的 KCBP 服务器名称*/
    int nProtocol; /*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*/
    char szProxy[KCBP_PROXY_MAX + 1];/*代理服务器参数*/
    char szSSL[KCBP_SSL_MAX + 1];/*SSL 参数*/
}tagKCBPConnectOptionEx;
```

## 18. KCBP\_OPTION\_CURRENT\_CONNECT\_EX

与 KCBP\_OPTION\_CURRENT\_CONNECT 区别是使用 tagKCBPConnectOptionEx 结构，增加了 SSL 和 PROXY 选项。

## 19. KCBP\_OPTION\_AUTHENTICATION

设置权限验证方式，1 表示 KCBPCLI\_ConnectServer 调用时验证用户权限；0 表示 KCBPCLI\_ConnectServer 调用时不验证用户权限；系统缺省值是 1。对于一部调用的 GetReply0 方式，需要将该值设置为 0。

**See also:**

我们建议使用 KCBPCLI\_SetOptions，不使用 KCBPCLI\_SetOption。

对于超时设置，我们也建议使用 KCBPCLI\_SetOptions，而不要使用 KCBPCLI\_SetCliTimeout。KCBPCLI\_GetOption、KCBPCLI\_SetOption、KCBPCLI\_SetCliTimeout、KCBPCLI\_SetConnectOption 几个函数会在将来的版本中作废。

## 20. KCBP\_OPTION\_GROUPID

消息组号，字符串类型。Group Id 用于识别调用者身份。如果客户不指定 GroupId，客户端 API 采用缺省值，这个值与客户机器网卡地址、当前工作目录相关，这意味着，客户程序运行多个实例时，需要使用不同的工作目录，避免收不到应答。

采用非 0 方式调用 GetReply 取异步应答时，要将该值设为空字符串""，以避免与 0 方式冲突。

## 21. KCBP\_OPTION\_USE\_GROUPID

用来控制发送请求报文时，是否传递 GROUPID，0 表示不传递 GROUPID；1 异步调用时传递 GROUPID，这是缺省值；2 表示同步调用时传递；3 表示同步、异步皆传递。

采用分组读应答报文，可以防止多个客户端读同一个应答队列时取到不属于自己的应答。

GROUPID 可用来区分应答报文的对应请求方式，当异步调用 GET 0 方式取应答与同步调用并存时，设置选项值为 1，可确保同步请求的应答不被 GET 0 取走；此外，GROUPID 对发布订阅的取应答行为也具有同样的控制作用。

## 22. KCBP\_OPTION\_RECEIVE\_PACKET\_BY\_GROUP

这个选项用来控制 KCBP 客户端读取消息时，是否采用分组（段）读取方式。缺省采用。

如果启用，客户端读取一段报文，供客户端处理，有需要时再读取下一段。如果不用，客户端将读取所有应答报文到内存中。当存在大量应答数据时，采用分段处理方式能，让客户端分段处理，界面上可获得较快响应。

对于报文中存在大对象数据的情况，KCBP 服务器向客户端发送包含大对象数据的报文时，不会设置分组标志，这时，不论客户端是否启用分组读取，由于没有分组标志，客户端一次读取所有报文。

## 23. KCBP\_OPTION\_XA

KCBP 客户端可使用 KDMID1PCFactory，二者结合，可实现 KCBP 客户端的业务流控制，提供通用方法实现与其他系统集成目的，降低与其他系统集成的复杂度，功能强大。KCBPCLI\_SetOption 可以通过选项 KCBP\_OPTION\_XA 来设置 KCBPXA 连接参数，参数格式如下：

```
/* struct for kcbpxa's parameter */
typedef struct
{
    char szServerName[KCBP_SERVERNAME_MAX + 1]; //XA name
    int nProtocol;
    char szSwitchLoadFile[KCBP_XA_PATH_SIZE + 1];
    char szXAEntry[KCBP_XA_ENTRY_SIZE + 1];
    char szXAOpen[KCBP_XA_OPEN_SIZE + 1];
    char szXAClose[KCBP_XA_CLOSE_SIZE + 1];
    char szXASerial[KCBP_XA_SERIAL_SIZE + 1];
    char szXAOption[KCBP_XA_OPTION_SIZE];
} tagKCBPXAConnectOption;
```

为便于调试，KCBPCP 增加对使用 KDMID1PCFactory 的 KCBPCLi 调用支持，可以通过命令设置连接参数。例子如下：

```
set kcbpxa
name=kdmid1pc;open=ip=127.0.0.1,port=28946,adapter=kdmidapi.dll,rule=rule32.xml,connecttime
out=5,userid=9999,password=3Mc+w9uU5lM=,timeout=30;option=
```

## 24. KCBP\_OPTION\_CURRENT\_XA

设置/读取 KCBPXA 连接参数。通过一个 tagKCBPXAConnectOption 结构的指针传递参数。

## 25. KCBP\_OPTION\_CONNECTION\_POOL

设置/取得连接池中的连接数。

```
int nValue;
KCBPCLI_SetOption(handle, KCBP_OPTION_CONNECTION_POOL, &value);
```

## 2. 4. 56. 查询通讯参数

函数原型:

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_GetSystemParam(KCBPCLIHANDLE
hHandle, int nIndex, char *szValue, int nLen);
```

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int nIndex	参数编号	Input
char *szValue	用于存放返回参数	Output
int nLen	缓冲区长度	Input

返回: 0 成功, 其它失败

用法说明:

nIndex 取值含义:

#define KCBP_PARAM_NODE	0	/*KCBP 节点号, 5Bytes*/
#define KCBP_PARAM_CLIENT_MAC	1	/*客户端网卡地址, 12Bytes*/
#define KCBP_PARAM_CONNECTION_ID	2	/*KCBP 连接号, 10Bytes*/
#define KCBP_PARAM_SERIAL	3	/*KCBP 交易流水号, 26Bytes*/
#define KCBP_PARAM_USERNAME	4	/*KCBP 用户名称, 26Bytes*/
#define KCBP_PARAM_PACKETTYPE	5	/*KCBP 请求报文类型, 1Bytes*/
#define KCBP_PARAM_PACKETTYPE	KCBP_PARAM_PACKETTYPE	
#define KCBP_PARAM_SERVICENAME	6	/*KCBP 被调用的服务名称, 8Bytes*/
#define KCBP_PARAM_RESERVED	7	/*KCBP 保留, 10Bytes*/
#define KCBP_PARAM_DESTNODE	8	/*KCBP 目标节点号, 5bytes*/

## 2. 4. 57. 设置通讯参数

函数原型:

```
KCBPCLI_API int KCBPCLISTDCALL KCBPCLI_SetSystemParam(KCBPCLIHANDLE
hHandle, int nIndex, char *szValue);
```

参数说明:

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
int nIndex	参数编号	Input
char *szValue	用于存放返回参数	Input

返回: 0 成功, 其它失败

用法说明:

nIndex 取值含义:

#define KCBP_PARAM_NODE	0	/*KCBP 节点号, 5Bytes*/
#define KCBP_PARAM_CLIENT_MAC	1	/*客户端网卡地址, 12Bytes*/
#define KCBP_PARAM_CONNECTION_ID	2	/*KCBP 连接号, 10Bytes*/
#define KCBP_PARAM_SERIAL	3	/*KCBP 交易流水号, 26Bytes*/
#define KCBP_PARAM_USERNAME	4	/*KCBP 用户名称, 26Bytes*/

```
#define KCBP_PARAM_PACKETTYPE      5      /*KCBP 请求报文类型, 1Bytes*/
#define KCBP_PARAM_PACKETGETYPE    KCBP_PARAM_PACKETTYPE
#define KCBP_PARAM_SERVICENAME     6      /*KCBP 被调用的服务名称, 8Bytes*/
#define KCBP_PARAM_RESERVED        7      /*KCBP 保留, 10Bytes*/
#define KCBP_PARAM_DESTNODE        8      /*KCBP 目标节点号, 5bytes*/
```

注意：每次调用 KCBPCLI\_BeginWrite 后，通讯参数都会被清空。

## 2. 4. 58. 直接设置通讯缓冲区

函数原型：KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_SetCommArea (KCBPCLIHANDLE hHandle, char \*pComm);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char * pComm	通讯缓冲区字符串	Input

返回：0 成功，其它失败

用法说明：将 pComm 的内容编码后存入通讯缓冲区。

注意：

## 2. 4. 59. 直接读取通讯缓冲区

函数原型：KCBPCLI\_API int KCBPCLISTDCALL KCBPCLI\_GetCommArea (KCBPCLIHANDLE hHandle, char \*\*p);

参数说明：

参数名称	参数说明	用法
KCBPCLIHANDLE hHandle	客户端句柄	Input
char **p	通讯缓冲区指针	Input

返回：0 成功，其它失败

用法说明：成功后，p 指向通讯缓冲区，缓冲区中的内容是编过码的。

注意：

## 2. 4. 60. SQL 风格函数

SQLConnect 与 KCBPCLI\_ConnectServer 相同；  
 SQLKCBPCLI\_DisConnect 与 KCBPCLI\_DisConnect 相同；  
 SQLExecute 与 KCBPCLI\_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 LBM API

LBM API 是 KCBP 服务端提供的 C 语言调用接口。

业务程序是 LBM，它一般用 ESQL/C、PROC 方式编写，LBM 通过 LBM API 访问 KCBP。

LBM API 通过 void \*pCA 与 KCBP 传递通讯信息。

一般来讲，LBM API 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. LBMAP I 分类

初始化、退出	初始化		KCBP_Initialize		
	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	设置变量值		KCBP_Set Value KCBP_Set Val	
		获取变量值		KCBP_Get Value KCBP_Get ValueN KCBP_Get Val	
	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 KCBP_RsGet Val KCBP_RsGet ValByName		
	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	XA 选择	KCBP_XASelect
	获取 XA 句柄	KCBP_GetXAHandle
	获取当前 XA 名称	KCBP_GetCurrentXAName
	调试函数	KCBP_DebugBreak
	输出调试信息	KCBP_PrintStatus
	读 KCBP 系统参数	KCBP_GetSystemParam
	安全 Sleep	KCBP_Sleep

### 3.3. LBMAP I 头文件

下面服务端编程接口的头文件 lbmapi.h:

```

/*
KCBP LBM API for C, Version 2.3
Copyright (c) 2002 SZKingdom Corp. All rights reserved.
Version 2.4, Mr. Yuwei Du, 20040801
    Add binary compatible API KCBP_SetVal, KCBP_GetVal, KCBP_RsSetVal,
KCBP_RsSetValByName, KCBP_RsGetVal, KCBP_RsGetValByName
Version 2.3, Mr. Yuwei Du, 20040630
    Add API KCBP_GetCurrentXAName
Version 2.2, Mr. Yuwei Du, 20031208
    Add publish api
Version 2.1, Mr. Yuwei Du, 20031128
    Add LBMEEXPORTS for recognize LBM function entry
Version 2.0, Mr. Yuwei Du, 20030618
    Add Support to KCBP/WIN
Version 1.0, Mr. Yuwei Du, 20020611
    Support KCBP/UNIX & CICS
*/
#ifdef _LBMAP_I_H
#define _LBMAP_I_H
#include <time.h>
#if defined(WIN32)
#if defined(LBMAP_I_EXPORTS)

```

```
#define LBMAPI_API __declspec(dllexport)
#else
#define LBMAPI_API __declspec(dllimport)
#endif
#define LBMSTDCALL __stdcall /* ensure stcall calling convention on NT */
#define LBMEXPORTS __declspec(dllexport)
#else
#define LBMAPI_API
#define LBMSTDCALL /* leave blank for other systems */
#define LBMEXPORTS
#endif
#ifndef __cplusplus
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
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
#define KCBP_PARAM_USERNAME 4
#define KCBP_PARAM_PACKETTYPE 5
#define KCBP_PARAM_PACKETTYPE KCBP_PARAM_PACKETTYPE
#define KCBP_PARAM_SERVICENAME 6
#define KCBP_PARAM_RESERVED 7
#define KCBP_PARAM_DESTNODE 8
#define KCBP_PARAM_VERSION 9
#define KCBP_PARAM_PACKETTYPEEXTEND 10
#define KCBP_PARAM_PASSWORD 11
#define KCBP_PARAM_SENDNODE 12

#define KCBP_NOBLOCK 0x00000100
#define KCBP_SYNCONRETURN 0x00000200
```

```
#define KCBP_NOTRAN                KCBP_SYNCONRETURN
#define KCBP_NOTIMEOUT             0x00000400
#define KCBP_XCTL                  0x00000800
#define KCBP_FORWARD               KCBP_XCTL

#define KCBP_OPTION_RETURNFLAG    30
#define KCBP_OPTION_CALL_TIMEOUT  31
#define KCBP_OPTION_HOLD_HANDLE   32

#if defined(__xLC__)
#pragma options align = packed
#else
#pragma pack(1)
#endif

/*****

/* callback notification function definition.                                */

*****/

/*****

typedef void (KCBP_Callback_t) (void *);
typedef KCBP_Callback_t *KCBP_Notify_t;

/* control parameters to publish/subscribe queue primitives */
typedef struct
{
    int nFlags;                /* indicates which of the values are set , include type, mode,
redelivered flag*/
    char szId[26 + 1];         /* pub/sub serial identifier */
    char szMsgId[24 + 1];     /* id of message before which to queue */
    int nExpiry;              /* subscribe message duration time, unit with second */
    int nPriority;             /* publish priority */
    time_t tTimeStamp;        /* pub/sub timestamp*/
    KCBP_Notify_t lpfnCallback; /* callback function pointer*/
} tagKCBPCallCtrl;

#if defined(__xLC__)
#pragma options align = reset
#else
#pragma pack()
#endif
#endif
extern "C"
{
```

```
#endif
LBMAPI_API LBMHANDLE LBMSTDCALL KCBP_Init(void *pCA);
LBMAPI_API int LBMSTDCALL KCBP_Initialize(LBMHANDLE * hHandle, void *pCA);
LBMAPI_API int LBMSTDCALL KCBP_Exit(LBMHANDLE hHandle);
LBMAPI_API int LBMSTDCALL KCBP_ExitEx(LBMHANDLE hHandle, void *pRetBuf, int
nRetBufLen);

LBMAPI_API int LBMSTDCALL KCBP_Abort(LBMHANDLE hHandle, int nErrno, int nLevel,
char *szErrMsg);

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

LBMAPI_API int LBMSTDCALL KCBP_GetValue(LBMHANDLE hHandle, char *KeyName,
char *Value);
LBMAPI_API int LBMSTDCALL KCBP_GetValueN(LBMHANDLE hHandle, char *KeyName,
char *Value, int Num);
LBMAPI_API int LBMSTDCALL KCBP_GetVal(LBMHANDLE hHandle, char *szKeyName,
unsigned char **pValue, long *pSize);

LBMAPI_API int LBMSTDCALL KCBP_BeginWrite(LBMHANDLE hHandle);

LBMAPI_API int LBMSTDCALL KCBP_SetValue(LBMHANDLE hHandle, char *KeyName, char
*Value);
LBMAPI_API int LBMSTDCALL KCBP_SetVal(LBMHANDLE hHandle, char *szKeyName,
unsigned char *pValue, long nSize);

LBMAPI_API int LBMSTDCALL KCBP_RsCreate(LBMHANDLE hHandle, char *Name, int
ColNum, char *TableInfo);
LBMAPI_API int LBMSTDCALL KCBP_RsNewTable(LBMHANDLE hHandle, char *Name, int
ColNum, char *TableInfo);
LBMAPI_API int LBMSTDCALL KCBP_RsAddRow(LBMHANDLE hHandle);
LBMAPI_API int LBMSTDCALL KCBP_RsSetCol(LBMHANDLE hHandle, int Col, char
*Value);
LBMAPI_API int LBMSTDCALL KCBP_RsSetColByName(LBMHANDLE hHandle, char *Name,
char *Value);
LBMAPI_API int LBMSTDCALL KCBP_RsSetVal(LBMHANDLE hHandle, int nColumnIndex,
unsigned char *pValue, long nSize);
LBMAPI_API int LBMSTDCALL KCBP_RsSetValByName(LBMHANDLE hHandle, char
*szColumnName, unsigned char *pValue, long nSize);
LBMAPI_API int LBMSTDCALL KCBP_RsSaveRow(LBMHANDLE hHandle);

LBMAPI_API int LBMSTDCALL KCBP_RsOpen(LBMHANDLE hHandle);
LBMAPI_API int LBMSTDCALL KCBP_RsGetCursorName(LBMHANDLE hHandle, char
*pszCursorName, int nLen);
```

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColNames(LBMHANDLE hHandle, char \*pszInfo, int nLen);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColName(LBMHANDLE hHandle, int nColIndex, char \*pszInfo, int nLen);

LBMAPI\_API int LBMSTDCALL KCBP\_RsFetchRow(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetCol(LBMHANDLE hHandle, int Col, char \*Value);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColN(LBMHANDLE hHandle, int Col, char \*Value, int Num);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColByName(LBMHANDLE hHandle, char \*Name, char \*Value);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColByNameN(LBMHANDLE hHandle, char \*Name, char \*Value, int Num);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetVal(LBMHANDLE hHandle, int nColumnIndex, unsigned char \*\*pValue, long \*pSize);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetValByName(LBMHANDLE hHandle, char \*szColumnName, unsigned char \*\*pValue, long \*pSize);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetRowNum(LBMHANDLE hHandle, int \*pnRows);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetColNum(LBMHANDLE hHandle, int \*pnCols);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetTableRowNum(LBMHANDLE hHandle, int nTableIndex, int \*pnRows);

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetTableColNum(LBMHANDLE hHandle, int nTableIndex, int \*pnCols);

LBMAPI\_API int LBMSTDCALL KCBP\_RsMore(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_RsClose(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_Begin(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_Commit(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_RollBack(LBMHANDLE hHandle);

LBMAPI\_API int LBMSTDCALL KCBP\_Abend(LBMHANDLE hHandle, char \*AbendCode);

LBMAPI\_API int LBMSTDCALL KCBP\_PrintStatus(LBMHANDLE hHandle, char \*szFormatter, ...);

LBMAPI\_API int LBMSTDCALL KCBP\_Warning(LBMHANDLE hHandle, int nType, int nValue, char \*szFormatter, ...);

LBMAPI\_API int LBMSTDCALL KCBP\_GetSystemParam(LBMHANDLE hHandle, int nParam, char \*pszBuffer, int nBufSize);

LBMAPI\_API int LBMSTDCALL KCBP\_Sleep(LBMHANDLE hHandle, int nSeconds);

LBMAPI\_API void \*LBMSTDCALL KCBP\_Malloc(LBMHANDLE hHandle, int nBytes, bool bShared);

LBMAPI\_API int LBMSTDCALL KCBP\_Free(LBMHANDLE hHandle, char \*pMem);

```
#define KCBP_MallocLocal(hHandle, nBytes)  KCBP_Malloc(hHandle, nBytes, 0)
#define KCBP_MallocShared(hHandle, nBytes) KCBP_Malloc(hHandle, nBytes, 1)

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

LBMAPI_API int LBMSTDCALL KCBP_Call(LBMHANDLE hHandle, char *prg, int nFlag);

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

LBMAPI_API int LBMSTDCALL KCBP_GetDBProcess(LBMHANDLE hHandle, void
**pDBHandle);
LBMAPI_API int LBMSTDCALL KCBP_XASelect(LBMHANDLE hHandle, char *szXAName,
void **pDBHandle);
LBMAPI_API int LBMSTDCALL KCBP_GetXAHandle(LBMHANDLE hHandle, char
*szXAName, void **pDBHandle);
LBMAPI_API int LBMSTDCALL KCBP_GetCurrentXAName(LBMHANDLE hHandle, char
*szXAName, int nLen);

LBMAPI_API int LBMSTDCALL KCBP_DebugBreak(LBMHANDLE hHandle);

LBMAPI_API int LBMSTDCALL KCBP_Publish(LBMHANDLE hHandle, tagKCBPCallCtrl *
pstPSCtl, char *pszTopicExpr, char *pszData, int nDataLen);

LBMAPI_API int LBMSTDCALL KCBP_SetOptions(LBMHANDLE hHandle, int nIndex, void
*pValue, int nLen);
LBMAPI_API int LBMSTDCALL KCBP_GetOptions(LBMHANDLE hHandle, int nIndex, void
*pValue, int *nLen);

LBMAPI_API int LBMSTDCALL KCBP_GetErrCode(LBMHANDLE hHandle, int *pnErrno);
LBMAPI_API int LBMSTDCALL KCBP_GetErrMsg(LBMHANDLE hHandle, char *szError, int
nSize);
```

```

LBMAPI_API int LBMSTDCALL KCBP_GetCommArea(LBMHANDLE hHandle, void **p);
LBMAPI_API int LBMSTDCALL KCBP_System(LBMHANDLE hHandle, int nType, void
**pBuffer);
LBMAPI_API int LBMSTDCALL KCBP_CallProgramByCA( LBMHANDLE hHandle, char *prg,
char **pCommArea, int *nCommAreaLen);
LBMAPI_API int LBMSTDCALL KCBP_SetStatus(LBMHANDLE hHandle, char Status);
LBMAPI_API int LBMSTDCALL KCBP_GetCwa(LBMHANDLE hHandle, char **pCWAPointer,
int *nSize);
LBMAPI_API int LBMSTDCALL KCBP_SetCommArea(LBMHANDLE hHandle, char
**pCommArea);

#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。

返回值说明：一般来讲，函数如执行成功，返回值则为 0。

#### 3.4.1. 初始化 KCBP 环境

**函数原型：** LBMAPI\_API int LBMSTDCALL KCBP\_Initialize(LBMHANDLE \* hHandle, void \*\*pCA);

**输入参数：**

参数名称	参数说明	用法
LBMHANDLE *hHandle	LBM 句柄指针	Output
void * pCA	KCBP 通讯区指针	Input

**返回：** 返回 0 成功，其他失败，

**用法说明：** 失败应立即调用 KCBP\_Exit 或 KCBP\_Abort 退出。

**参见：** extern LBMAPI\_API LBMHANDLE LBMSTDCALL KCBP\_Init( void \* pCA) ;  
KCBP\_Init 已经被 KCBP\_Initialize 替代，不建议用户使用 KCBP\_Init。

#### 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, 或用在业务处理过程中需要进行跨函数返回等特殊操作。该函数一经调用, LBM 立即返回, 后面代码不再执行。用于使用这个函数前, 要注意资源的回收操作, 比如前面申请的内存, 要在这个函数调用前释放。

### 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) 设置键值 (VALUE) 为指定长度的缓冲区

函数原型:

```
LBMAPI_API int LBMSTDCALL KCBP_SetVal(LBMHANDLE hHandle, char *szKeyName, unsigned char *pValue, long nSize);
```

输入参数:

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
char *szKeyName	关键字	Input
char *pValue	输入缓冲区	Input
long nSize	输入缓冲区长度	Input

返 回: 0 成功, 其它失败

用法说明:

支持设置二进制数据。

参 见:

KCBPCLI\_SetVal、KCBP\_GetVal。

### 3.4.8. 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(Value)。

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。

### 3.4.9. 根据键名 (KEYNAME) 取键值 (VALUE) 缓冲区和长度

函数原型：

```
LBMAPI_API int LBMSTDCALL KCBP_GetVal(LBMHANDLE hHandle, char *szKeyName, unsigned char **pValue, long *pSize);
```

输入参数：

参数名称	参数说明	用法
LBMHANDLE hHandle	KCBP_Init 返回的 handle	Input
Char *szKeyName	关键字	Input
Char **pValue	关键字值缓冲区指针	Output
long *pSize	Value 缓冲区长度	Output

返回：0 成功，其它失败

用法说明：

该函数可返回二进制缓冲区及长度。

KCBP\_GetValue 只能处理字符串数据，不能处理二进制数据。

KCBP\_GetVal 可以处理 KCBPCLI\_SetVal、KCBPCLI\_SetValue、KCBP\_SetVal、KCBP\_SetValue 输入的数据。

参见：

KCBPCLI\_GetVal, KCBP\_SetVal。

### 3.4.10. 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 结果集行长度有上限，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"。

### 3.4.11. 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。

### 3.4.12. 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：

### 3.4.13. 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. 14. 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. 15. 根据列号设置当前行的某一列值为指定长度的缓冲区

**函数原型:**

LBMAPI\_API int LBMSTDCALL KCBP\_RsSetVal(LBMHANDLE hHandle, int nColumnName, unsigned char \*pValue, long nSize);

**参数说明:**

LBMHANDLE hHandle	LBMAPI 句柄	Input
Int nColumnName	列编号	Input
unsigned char *pValue	输入缓冲区	Input
long nSize	输入缓冲区长度	Input

**返 回:** 0 成功, 其它失败

**用法说明:**

该函数既可设置指定长度的二进制缓冲区, 也可以设置字符串缓冲区。

与此函数相关的 RsSetValue, 只能设置字符串缓冲区。

RsSetVal 的输入如果是字符串, 可以用 RsGetCol、RsGetColByName 和 RsGetVal、RsGetValByName 取出。

RsSetVal 的输入如果是二进制数据, 只能用 RsGetVal 及 RsGetValByName 取出。

**参 见:**

KCBPCLI\_RsSetVal,

KCBPCLI\_RsSetValByName,

KCBP\_RsGetVal,

KCBP\_RsGetValByName。

### 3. 4. 16. 根据列名设置当前行的某一列值为指定长度的缓冲区

**函数原型:**

LBMAPI\_API int LBMSTDCALL KCBP\_RsSetValByName(LBMHANDLE hHandle, char \*szColumnName, unsigned char \*pValue, long nSize);

**参数说明:**

LBMHANDLE hHandle	LBMAPI 句柄	Input
char *szColumnName	列名称	Input
unsigned char *pValue	输入缓冲区	Input
long nSize	输入缓冲区长度	Input

返回：0 成功，其它失败

用法说明：

用法同上。与 KCBP\_RsSetVal 差别在于 KCBP\_RsSetValByName 使用的是列名称，KCBP\_RsSetVal 使用列编号。

### 3.4.17. 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：

### 3.4.18. 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。

### 3.4.19. 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. 20. 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. 21. 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. 22. 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. 23. 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. 24. 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. 25. 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. 26. 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. 27. 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:**

### 3.4.28. 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。

### 3.4.29. 获得当前行的某一列的缓冲区和长度 (根据列序号)

**函数原型:**

LBMAPI\_API int LBMSTDCALL KCBP\_RsGetVal(LBMHANDLE hHandle, int nColumnIndex, unsigned char \*\*pValue, long \*pSize);

**输入参数:**

参数名称	参数说明	用法
LBMHANDLE hHandle	LBMAPI 句柄	Input
int nColumnIndex	列编号	Input
unsigned char **pValue	输出缓冲区指针	Output
long *pSize	输出缓冲区长度	Output

**返回:** 0 成功, 其它失败

**用法说明:**

该函数既可获取二进制缓冲区, 也可以获取字符串缓冲区。

该函数返回的缓冲区指针, 由 KCBP 客户端接口自动维护, 用户不需要它执行释放操作。用户应该在 RsGetVal 后立即处理该缓冲区中的数据 (比如复制到其他地方)。

与该函数类似的是 RsGetCol, 但 RsGetCol 只能用来获得字符串, 不能获得二进制 Buffer, 并且 RsGetCol 的缓冲区由用户指定, 无边界控制。



参 见:

KCBPCLI\_RsGetVal、

KCBPCLI\_RsSetVal、

KCBP\_RsSetVal。

### 3. 4. 30. 获得当前行的某一列的缓冲区和长度（根据列名称）

函数原型:

```
LBMAPI_API int LBMSTDCALL KCBP_RsGetValByName(LBMHANDLE hHandle, char *szColumnName, unsigned char **pValue, long *pSize);
```

输入参数:

参数名称	参数说明	用法
LBMHANDLE hHandle	LBMAPI 句柄	Input
char *szColumnName	列编号	Input
unsigned char **pValue	输出缓冲区指针	Output
long *pSize	输出缓冲区长度	Output

返 回: 0 成功, 其它失败

用法说明:

该函数既可获取二进制缓冲区, 也可以获取字符串缓冲区。

### 3. 4. 31. 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. 32. 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. 33. 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. 34. 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. 35. 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. 36. 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。

### 3.4.37. 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。

### 3.4.38. 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。

### 3.4.39. 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。

### 3.4.40. 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。

### 3.4.41. 分配局部内存

**函数原型：** #define KCBP\_MallocLocal( hHandle, 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。

### 3.4.42. 分配共享内存

**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。

### 3.4.43. 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：**

### 3.4.44. 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. 45. 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. 46. 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. 47. 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。

### 3. 4. 48. 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. 49. 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. 50. 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, 5Bytes*/
#define KCBP_PARAM_CLIENT_MAC	1	/*客户端网卡地址, 12Bytes */
#define KCBP_PARAM_CONNECTION_ID	2	/*KCBP 连接号, 10Bytes */
#define KCBP_PARAM_SERIAL	3	/*KCBP 交易流水号, 26Bytes */
#define KCBP_PARAM_USERNAME	4	/*KCBP 用户名称, 26Bytes */
#define KCBP_PARAM_PACKETTYPE	5	/*KCBP 请求报文类型, 1Bytes*/
#define KCBP_PARAM_PACKETTYPE	KCBP_PARAM_PACKETTYPE	
#define KCBP_PARAM_SERVICENAME	6	/*KCBP 被调用的服务名称, 8Bytes*/
#define KCBP_PARAM_RESERVED	7	/*KCBP 保留, 10Bytes*/
#define KCBP_PARAM_DESTNODE	8	/*KCBP 目标节点号, 5bytes*/

### 3.4.51. XA 选择

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_XASelect(LBMHANDLE hHandle, char \*szXAName, void \*\*pDBHandle);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char *szXAName	XA 名称	Input
void **pDBHandle	XA 句柄指针	Output

**返 回:** 0 成功, 其它失败

**用法说明:** KCBP 事务服务器支持多 XA 操作, 该函数提供 XA 切换功能。例如, 在证券交易系统中, 有一个存放当前数据的数据库, 还有一个存放历史数据的服务器, 在 KCBP 中, 配置两个 XA, 指向当前库的 XA 名称 current, 指向历史库的 XA 的名称是 history, 这样, 就可以使用下面方式访问这两个资源:

```
int nRet;
```

```
void *pXAHandle;
```

```
nRet = KCBP_XASelect(hHandle, "current", &pXAHandle);
```

```
if(nRet != 0)
```

```
{
```

```
    //未找到名称为 currnet 的 XA 定义, 错误处理
```

```
    return false;
```

```
}
```

```
//操作 current 数据库
```

```
//...
```

```
nRet = KCBP_XASelect(hHandle, "history", &pXAHandle);
```

```
if(nRet != 0)
```

```
{
```

```
    //未找到名称为 history 的 XA 定义, 错误处理
```

```
    return false;
```

```
}
```

```
//操作 history 数据库
```

```
//...
```

```
return true;
```

### 3.4.52. 取 XA 句柄

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_GetXAHandle(LBMHANDLE hHandle, char \*szXAName, void \*\*pDBHandle);

**参数说明:**

参数名称	参数说明	用法
------	------	----

LBMHANDLE hHandle	Lbm 句柄	Input
char *szXAName	XA 名称	Input
void **pDBHandle	XA 句柄指针	Output

返回：0 成功，其它失败

用法说明：KCBP 事务服务器支持多 XA 操作，该函数提供取 XA 句柄功能。KCBP\_GetXAHandle 与 KCBP\_XASelect 的区别是 KCBP\_XASelect 在取得 XA 句柄的同时并将其切换为当前句柄，而 KCBP\_GetXAHandle 只取，不切换。

当调用 KCBP\_Commit、KCBP\_RollBack 等操作时，影响当前的 XA。

KCBP\_GetXAHandle 一般用来管理非数据库资源，比如用于通讯的 KCXPXA、KDMID1PC、KCBPXA 等。

### 3.4.53. 获取当前 XA 名称

函数原型：LBMAPI\_API int LBMSTDCALL KCBP\_GetCurrentXAName(LBMHANDLE hHandle, char \*szXAName, int nLen);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char *szXAName	XA 名称缓冲区	Output
nLen	XA 名称缓冲区长度	Input

返回：0 成功，其它失败

用法说明：KCBP 事务服务器支持多 XA 操作，该函数用于获取当前使用的 XA 名称功能。

### 3.4.54. 调试函数

函数原型：LBMAPI\_API int LBMSTDCALL KCBP\_DebugBreak(LBMHANDLE hHandle);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	LBM 句柄	Reserved

返回：0 成功，其它失败

用法说明：该函数调用后将触发一次调试中断，使调试器进入断点状态。

注意：在 UNIX 平台，当程序单独运行，未被调试器调试时，调用该函数将造成当前程序挂起。在 WINDOWS 平台上，当程序单独运行，未被调试器调试时，调用该函数将造成当前 LBM 异常结束。

### 3.4.55. 发布函数

函数原型：

LBMAPI\_API int LBMSTDCALL KCBP\_Publish(LBMHANDLE hHandle, tagKCBPCallCtrl \*pstPSCtl, char \*pszTopicExpr, char \*pszData, int nDataLen);

参数说明：

参数名称	参数说明	用法
LBMHANDLE hHandle	LBM 句柄	Input
tagKCBPCallCtrl *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

tagKCBPCallCtrl \*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	Ignore
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	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:** 这是一个服务端的消息发布函数,提供按主题发布消息的功能。

**参见:** KCBPCLI\_Publish

### 3.4.56. 切换到缺省 XA, 并获取句柄

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_GetDBProcess(LBMHANDLE hHandle, void \*\*pDBHandle);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
void **pDBHandle	XA 名称缓冲区	Output

**返回:** 0 成功,其它失败

**用法说明:** 切换到缺省 XA, 并返回其 XA 句柄。相当于调用 KCBP\_XASelect(hHandle,"",(void \*\*)&pDBHandle);

### 3.4.57. 设置 LBM 调用相关的参数

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_SetOptions(LBMHANDLE hHandle, int nIndex, void \*pValue, int nLen);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nIndex	参数索引	Input
void *pValue	参数值指针	Input
int nLen	参数值长度	Input

**返回:** 0 成功,其它失败

**用法说明:** 参数索引的值:

```
#define KCBP_OPTION_RETURNFLAG 30
#define KCBP_OPTION_CALL_TIMEOUT 31
#define KCBP_OPTION_HOLD_HANDLE 32
```

### 3.4.58. 获取 LBM 调用相关的参数

**函数原型：** LBMAPI\_API int LBMSTDCALL KCBP\_GetOptions(LBMHANDLE hHandle, int nIndex, void \*pValue, int \*nLen);

**参数说明：**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nIndex	参数索引	Input
void *pValue	返回值缓冲区	Output
int *nLen	返回值缓冲区长度	Output

**返回：** 0 成功，其它失败

**用法说明：** 参数索引的值：

```
#define KCBP_OPTION_RETURNFLAG    30
#define KCBP_OPTION_CALL_TIMEOUT  31
#define KCBP_OPTION_HOLD_HANDLE   32
```

### 3.4.59. 返回错误码

**函数原型：** LBMAPI\_API int LBMSTDCALL KCBP\_GetErrCode(LBMHANDLE hHandle, int \*pnErrno);

**参数说明：**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int *pnErrno	错误代码指针	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，返回错误代码，该种失败是由 KCBP 返回的，与业务程序无关。pnErrno 为 0 表示无错误。

### 3.4.60. 返回错误信息

**函数原型：** LBMAPI\_API int LBMSTDCALL KCBP\_GetErrMsg(LBMHANDLE hHandle, char \*szError, int nSize);

**参数说明：**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char *szError	错误信息缓冲区	Output
int nSize	错误信息缓冲区长度	Input

**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。

### 3.4.61. 获取通讯缓冲区内容

**函数原型:** LBM\_API int LBMSTDCALL KCBP\_GetCommArea(LBMHANDLE hHandle, void \*\*p);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
void **p	返回的内容指针	Output

**返回:** 0 成功, 其它失败。

**用法说明:** void \*\*p 指针不用 LBM 释放, 缓冲区中的内容是经过编码的。

### 3.4.62. 直接设置通讯缓冲区内容

**函数原型:** LBM\_API int LBMSTDCALL KCBP\_SetCommArea(LBMHANDLE hHandle, char \*pCommArea)

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
*pCommArea	要设置的缓冲区	Input

**返回:** 0 成功, 其它失败。

**用法说明:** 将 pCommArea 内容编码后存入通讯缓冲区,。

### 3.4.63. 获取/设置 KCBP 系统相关参数值

**函数原型:** LBM\_API int LBMSTDCALL KCBP\_System(LBMHANDLE hHandle, int nType, void \*pBuffer);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
int nType	参数类型索引	Input
void *pBuffer	参数值指针	Input/Output

**返回:** 0 成功, 其它失败。

**用法说明:** 根据 nType 的值, 完成获取或者设置 KCBP 系统相关的参数。nType 的值有:

```

#define KCBP_SYSTEM_MANAGE          1    //char*
#define KCBP_FINDFIRST_SUBSCRIBE    2    // tagSubscribe 指针
#define KCBP_FINDNEXT_SUBSCRIBE     3    // tagSubscribe 指针
#define KCBP_GET_MSGID               4    //char*,长度 24
#define KCBP_GET_CORRID              5    //char*,长度 24
#define KCBP_GET_GROUPID             6    //char*,长度 24
#define KCBP_SWITCH_XA_HANDLE        7    //tagXAItem 指针
#define KCBP_GET_CRYPTMODE           8    //int*
#define KCBP_GET_GETCOMPRESSMODE     9    //int*
#define KCBP_GET_CWAPOINTER          10   //char*

```

```
#define KCBP_GET_CWASIZE 11 //int*
#define KCBP_GET_NODEID 12 //int*
#define KCBP_GET_CURRENT_XA_HANDLE 13 // tagXAItem 指针
#define KCBP_GET_LASTERROR 14 //int*
#define KCBP_GET_PUBLISH_LIFETIME 15 //int*
#define KCBP_SYSTEM_WARNING 16 //char*
#define KCBP_GET_GETCOMPRESSTHRESHOLD 17 //int*
#define KCBP_GET_XA_HANDLE 18 // tagXAItem 指针
#define KCBP_GET_DESTNODE 19 //int*
#define KCBP_GET_DESTQ 20 //char*, 长度 33
#define KCBP_GET_REQUEST_ORIGINALNODE 21 //char*, 长度 4
#define KCBP_GET_REQUEST_MAC 22 //char*, 长度 12
#define KCBP_GET_REQUEST_CONNECTIONID 23 //char*,长度 10
#define KCBP_GET_REQUEST_SERIAL 24 //char*,长度 26
#define KCBP_GET_REQUEST_USERNAME 25 //char*,长度 8
#define KCBP_GET_ENDSTATUS 26 //int*
#define KCBP_GET_SERVICE_STATISTIC 27 //char*,大于 LBM 数*42
#define KCBP_GET_PEERIP 28 //char*,长度 16
#define KCBP_GET_AUTH_USER 29 //char*,长度 20
#define KCBP_GET_AUTH_CODE 30 //char*,长度 4
#define KCBP_SET_MSGID 31 //char*
#define KCBP_SET_CORRID 32 //char*
#define KCBP_SET_GROUPID 33 //char*
#define KCBP_SET_DESTNODE 34 //char*
#define KCBP_SET_DESTQ 35 //char*
#define KCBP_GET_INPUTQ_ID 36 //int*
#define KCBP_GET_QM_CONNECT_HANDLE 37 //int*
#define KCBP_RESET_ANSWER_PACKET_INDEX 38 //不用
#define KCBP_GET_CALL_STATUS 39 //int*
#define KCBP_SET_CALL_STATUS 40 //int*
#define KCBP_SET_INPUTQ_ID 41 // int*
#define KCBP_GET_LIFETIME 42 // int*
#define KCBP_SET_LIFETIME 43 // int*
#define KCBP_GET_SERVICENAME 44 //char*,长度 8
#define KCBP_GET_TRACEDATA 45 //char*,长度最少 512
#define KCBP_GET_NESTEDLEVEL 46 // int*
#define KCBP_GET_ISC_TIMEOUT 47 // int*
#define KCBP_GET_LBM_TIMEOUT 48 // int*
#define KCBP_GET_SPD_BY_NAME 49 //入参 char*,出参 tagSPD*
#define KCBP_GET_MULTICAST_PARAM 50 // MULTICASTPARAM*
#define KCBP_SET_INTERNAL_CALL_ANSWER 51
#define KCBP_GET_INTERNAL_CALL_ANSWER 52
#define KCBP_SET_PACKET_INDEX 53 // int*
#define KCBP_GET_PACKET_INDEX 54 // int*
```

```

#define KCBP_GET_REQUEST_RESERVED    55 //char*,长度 10
#define KCBP_SET_REQUEST_RESERVED    56 //char*
#define KCBP_UPDATE_RUNTIMEDB        61

```

KCBP\_GET\_MULTICAST\_PARAM 返回多播结构如下:

```

typedef struct tagMulticastParam
{
    int nNodeId;
    int nProtocol;
    char szAddress[16];
    int nPort;
    int nTTL;
}MULTICASTPARAM, *LPMULTICASTPARAM;

```

### 3. 4. 64. 通过指定缓冲区发起本地调用

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_CallProgramByCA( LBMHANDLE hHandle, char \*prg, char \*\*pCommArea, int \*nCommAreaLen);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char *prg		Input
char **pCommArea	通讯缓冲区	Input/Output
int *nCommAreaLen	通讯缓冲区长度	Input/Output

**返回:** 0 成功, 其它失败。

**用法说明:** 本函数发起同步调用, 返回的应答内容也放在 pCommArea 里, 应答长度是 nCommAreaLen。如果应答长度比请求长度大时, 将会 realloc pCommArea, 因此这时 pCommArea 要是通过 malloc 分配的, 若是其它方式, 将会出错。

### 3. 4. 65. 设置事务状态

**函数原型:** LBMAPI\_API int LBMSTDCALL KCBP\_SetStatus(LBMHANDLE hHandle, char Status);

**参数说明:**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char Status	状态	Input

**返回:** 0 成功, 其它失败。

**用法说明:**

### 3.4.66. 获取 CWA 信息

**函数原型：** LBMAPI\_API int LBMSTDCALL KCBP\_GetCwa(LBMHANDLE hHandle, char \*\*pCWAPointer, int \*nSize);

**参数说明：**

参数名称	参数说明	用法
LBMHANDLE hHandle	Lbm 句柄	Input
char **pCWAPointer	CWA 缓冲区指针	Output
int *nSize	CWA 长度	Output

**返回：** 0 成功，其它失败。

**用法说明：**

## 4. 编程注意事项

### 4.1. 客户端编程注意事项

#### 1. C API 与 C++ API

我们推荐用户使用 C API, 原因是它的功能更多, 并且作为一种 Windows API, 它可以被更多的程序设计语言使用, 如 Visual Basic 及 Dephi 等。

在 KCBP3.0 版本之后, C++API 已废弃。

#### 2. 断线重连

KCBP3.0 以前版本, 当使用服务调用类函数 (参见第二章 CLIENT API 函数分类) 如 KCBPCLI\_CallProgramAndCommit、KCBPCLI\_GetReply 等, 返回码如果是 2054、2055、2003 时, 表示连接已断, 这时需要调用 KCBPCLI\_DisConnectForce 关闭连接, 然后再调用 KCBPCLI\_ConnectServer 重现建立连接。

KCBP V3.0 Build 120321 以后版本, 客户端 API 具有连接自动维护功能。

#### 3. KCBPCLI\_DisConnect 与 KCBPCLI\_DisConnectForce 的区别

KCBPCLI\_DisConnectForce 不与 KCBP Server 通讯, 直接关闭本地的 Socket 句柄; KCBPCLI\_DisConnect 函数要和 KCBP Server 进行一次通讯, 在连接已经断开的情况下, 这次通讯会失败, 返回码不是 0。当连接已经异常断开时, 请调用 KCBPCLI\_DisConnectForce。

4. KCBP V3.0 Build 120321 以后版本, 客户端增加连接池功能。连接池中缓存的连接数, 缺省是 0, 表示不启用, 与以前版本兼容。启用连接池, 可通过 2 种方式控制, 一是设置环境变量 KCBPCLI\_OBJECTSPOOLSIZE=连接数, 二是调用函数 KCBPCLI\_SetOption(handle, KCBP\_OPTION\_CONNECTION\_POOL, &value);

#### 5. 作废函数

我们建议不要使用下面这几个函数, 他们已经被 KCBPCLI\_SetOptions、

KCBPCLI\_GetOptions 替代:

KCBPCLI\_SetConnectOption

KCBPCLI\_GetConnectOption

KCBPCLI\_SetOption

KCBPCLI\_GetOption

KCBPCLI\_SetCliTimeOut

为了兼容以前的版本，客户端 API 保留了这几个函数。

## 6. BLOB 数据

为了适合 BLOB 类型的数据传输，KCBP 提供了下面函数:

KCBPCLI\_GetVal

KCBPCLI\_SetVal

KCBPCLI\_RsSetVal

KCBPCLI\_RsSetValByName

KCBPCLI\_RsGetVal

KCBPCLI\_RsGetValByName

这几个函数可以替代原有的函数:

KCBPCLI\_SetValue

KCBPCLI\_GetValue

KCBPCLI\_RsSetCol

KCBPCLI\_RsSetColByName

KCBPCLI\_RsGetColByName

KCBPCLI\_RsSetCol

使用新的\*GetVal 函数时，需要注意一点，返回的 Buffer 由 KCBP Client API 自动分配，有效期至下次调用\*GetVal 或\*BeginWrite，这就需要用户再调用\*GetVal 后马上将其内容复制到自己的数据缓冲区中进行保



存，以免数据被后续调用修改。

## 7. 常见返回码说明

0 一般表示调用成功，非 0 表示失败。KCBP 错误返回码一般小于 2000，KCBP 的返回码定义在 KCBPError.XML 中；KCXP 错误返回码一般大于 2000，可以在 KCXPAPI.H 或 KCXP 编程手册中查找错误说明。

1002 被调用的 LBM 在 KCBP Server 端未定义。

1004 被调用的 LBM 不能加载，可能是路径错误，或加载 LBM 时找不到相关动态库(在 Windows 上用 VC 的 Depends 工具检查 DLL 的相关性，在 LINUX 上用 ldd 检查 so 的相关性)。

1006 被调用的 LBM 入口函数定义错误，在动态库中找不到该入口。

1059 远程调用（或转发请求）完成后，在 KCBP 内部处理应答结果失败。

1060 远程调用（或转发请求）读应答失败。

1061 远程调用（或转发请求）发送请求失败。

1064 远程调用（或转发请求）发送请求超时错误。

1065 远程调用（或转发请求）读应答超时错误。

1068 远程调用功能未授权，需要向金证索取支持 KDMID1PCFactory 功能的授权号。

2003 表示连接句柄错误。

2011 表示调用超时，主要发生在等待结果阶段。

2054、2055 表示 Socket 连接发生中断。

2103 KCXP 插件调用错误，这类错误出现条件是 KCXP 的插件动态库（UNIX 版文件名是 libzlib.so、libzlib1.so、libzip.so、libidea.so、libidea1.so、libdes.so，Window 版文件名与 UNIX 版差别在于后缀是.dll）找不到，这些文件需要存放到当前程序运行目录下，或存放到系统搜索路径下。

2016 表示 KCXP 的插件加载失败，原因是找不到数据文件 KCXPUser.dat、KCXPPlugin.Dat。这两个文件需要与 KCXP 插件动态库

放到同一个目录下。并要注意在 UNIX 和 Windows 要使用正确的文件。

2300 SSL 连接错误，原因是客户端和 KCXP Server 端 SSL 设置不匹配。

## 4. 2. 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 KCBPCLI\_DISCONNECT 数据库的操作, 如果 LBM 自己进行 CONNECT 和 KCBPCLI\_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 KCBPCLI\_CallProgram 接口函数后一定要提交, 或者直接调用 KCBPCLI\_CallProgramAndCommit 接口函数。

17. 调用 KCBPCLI\_CallProgram/KCBPCLI\_CallProgramAndCommit 后, 如果不调用 KCBPCLI\_ConnectServer 而直接再次调用 KCBPCLI\_SetValue 和 KCBPCLI\_CallProgram/KCBPCLI\_CallProgramAndCommit, 应先调用 KCBPCLI\_BeginWrite 以清空公共缓冲区(这是因为上次调用 KCBPCLI\_CallProgram/KCBPCLI\_CallProgramAndCommit 的返回值还在公共缓冲区中)。

18. 客户机尽量调用 KCBPCLI\_CallProgramAndCommit, 此调用效率最高。

KCBPCLI\_CallProgramAndCommit 是否最终提交取决于 CICS Server 执行 Program 的情况:

- 服务器没有未截获的错误, 正常返回(调用 ExitEasyCics)时, 事务提交
- 服务器没有未截获的错误, 并显式调用 CicsCommit 时, 事务提交
- 服务器没有未截获的错误, 并显式调用 CicsRollBack 时, 事务回滚
- 服务器有未截获的错误, 事务回滚

19. 客户机调用 KCBPCLI\_CallProgram 时, 最终一定要调用 KCBPCLI\_Commit 或 KCBPCLI\_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 KCBPCLI\_CallProgram 系列函数后, 最好调用 KCBPCLI\_GetErr/KCBPCLI\_GetErrCode 函数以判断是否调用成功, 若 KCBPCLI\_GetErr 返回空串或 KCBPCLI\_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 LBM API 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 KCBP SERVER 与 CICS SERVER 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 的服务程序和用于 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
LBM 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.nProtocol = 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->KCBPCLI_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->SQLKCBPCLI_DisConnect();

    delete pKCBPCli;

    return 0;

}
```

## 5. 2. 服务端 LBM

### 5. 2. 1. 不访问数据库的 LBM

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "lbmap.i.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", "oooooooooooooooooooo");

    KCBP_RsSetColByName(hHandle, "bbb", "aaaaaaaaaaaaaaaaaaaa");

    KCBP_RsSetColByName(hHandle, "cbb", "uuuuuuuuuuuuuuuuuu");

    KCBP_RsSetColByName(hHandle, "dbb", "rrrrrrrrrrrrrrrrrr");

    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","oooooooooooooooo");

    KCBP_RsSetColByName(hHandle, "bbb","aaaaaaaaaaaaaaaa");
```

```
        KCBP_RsSetColByName(hHandle, "cbb", "uuuuuuuuuuuuuuuuuu");

        KCBP_RsSetColByName(hHandle, "dbb", "rrrrrrrrrrrrrrrrrr");

        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;
}
```

### 5.2.2. 访问 ORACLE 数据库的 LBM

下面这个程序是一个 LBM 程序，是用 ORACLE 的 PROC 方法实现的，它适合在多进程方式运行的 KCBP，其中展现了如何编写 LBM 以调用 ORACLE 数据库中存储过程的技术。

```
#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include "lbmap.h"

/*

proc iname=test.pc  userid=ta/test  sqlcheck=semantics  ireclen=256  dynamic=ansi  mode=ansi
code=cpp parse=none

*/

EXEC SQL INCLUDE SQLCA;

int SetLbmError(LBMHANDLE hHandle, int nErrorCode, int nErrorLevel, char *szErrorMsg)

{

    char szTmp[12];

    KCBP_BeginWrite(hHandle);

    KCBP_RsCreate(hHandle, "MESSAGE", 3, "LEVEL, CODE, TEXT");

    KCBP_RsAddRow(hHandle);

    sprintf(szTmp, "%d", nErrorLevel);

    KCBP_RsSetColByName(hHandle, "LEVEL", szTmp);
```

```
    sprintf(szTmp,"%d",nErrorCode);

    KCBP_RsSetColByName(hHandle, "CODE", szTmp);

    KCBP_RsSetColByName(hHandle, "TEXT", szErrorMsg);

    KCBP_RsSaveRow(hHandle);

    return 0;

}

extern "C" LBMEXPORTS void *p_test(void *pCA)

{

EXEC SQL BEGIN DECLARE SECTION;

    char szServiceId[20];

    char szRetcode[20];

    char szReturnMessage[100];

EXEC SQL END DECLARE SECTION;

    LBMHANDLE hHandle;

    hHandle = KCBP_Init(pCA);

    if(hHandle == NULL) return NULL;

    memset(szRetcode, 0, sizeof(szRetcode));

    memset(szReturnMessage, 0, sizeof(szReturnMessage));

    KCBP_GetValue(hHandle, "QUERYID", szServiceId);

    KCBP_BeginWrite(hHandle);

EXEC SQL EXECUTE
```

```
BEGIN

    P_Test(:szServiceId, :szRetcode, :szReturnMessage);

END;

END-EXEC;

if(sqlca.sqlcode!=0)

{

    SetLbmError(hHandle, sqlca.sqlcode, 0, " call procedure fail");

    goto LabelExit;

}

/*

KCBP_SetValue(hHandle, "RETCODE", szRetcode);

KCBP_SetValue(hHandle, "MESSAGE", szReturnMessage);

*/

SetLbmError(hHandle, atoi(szRetcode), 0, szReturnMessage);

LabelExit:

KCBP_Commit(hHandle);

KCBP_Exit(hHandle);

return NULL;

}
```

### 5.2.3. 多线程状态运行的 KCBP 上访问 ORACLE 数据库的 LBM

下面这个程序是一个 LBM 程序，是用 ORACLE 的 PROC 方法实现的，它展现了在多线程方式运行的 KCBP 上如何编写 LBM 以调用 ORACLE 数据库中存储过程的技术。阅读时请注意 **envtext** 的获取及使用方法。

```
#include "stdafx.h"
```

```
#include <stdio.h>

#include <stdlib.h>

#include "lbmap.i.h"

/*

proc iname=test_r.pc userid=ta/test sqlcheck=semantics ireclen=256 dynamic=ansi mode=ansi
code=cpp parse=none THREADS=YES

*/

EXEC SQL INCLUDE SQLCA;

int SetLbmError(LBMHANDLE hHandle, int nErrorCode, int nErrorLevel, char *szErrorMsg)

{

char szTmp[12];

KCBP_BeginWrite(hHandle);

KCBP_RsCreate(hHandle, "MESSAGE", 3, "LEVEL, CODE, TEXT");

KCBP_RsAddRow(hHandle);

sprintf(szTmp, "%d", nErrorLevel);

KCBP_RsSetColByName(hHandle, "LEVEL", szTmp);

sprintf(szTmp, "%d", nErrorCode);

KCBP_RsSetColByName(hHandle, "CODE", szTmp);

KCBP_RsSetColByName(hHandle, "TEXT", szErrorMsg);

KCBP_RsSaveRow(hHandle);

return 0;

}
```

```
void * GetOracleThreadCtx(LBMHANDLE hHandle)

{

    int nRet;

    struct

    {

        void *envtext;

    }*pctx;

    nRet = KCBP_GetDBProcess(hHandle, (void **)&pctx);

    if(nRet != 0)

    {

        return NULL;

    }

    return pctx->envtext;

}

extern "C" LBMEXPORTS void *p_test_r(void *pCA)

{

    EXEC SQL BEGIN DECLARE SECTION;

    char szServiceId[20];

    char szRetcode[20];

    char szReturnMessage[100];

    sql_context envtext;

    EXEC SQL END DECLARE SECTION;

    LBMHANDLE hHandle;
```

```
hHandle = KCBP_Init(pCA);

if(hHandle == NULL) return NULL;

szRetcode[0]=0;

szReturnMessage[0]=0;

KCBP_GetValue(hHandle, "QUERYID", szServiceId);

KCBP_BeginWrite(hHandle);

envtext = GetOracleThreadCtx(hHandle);

if (envtext == NULL)

{

    SetLbmError(hHandle, 999999, 0, "GetOracleThreadCtx fail");

    goto LabelExit;

}

EXEC SQL CONTEXT USE :envtext;

if (sqlca.sqlcode != 0)

{

    SetLbmError(hHandle, sqlca.sqlcode, 0, "set thread context fail");

    goto LabelExit;

}

EXEC SQL EXECUTE

BEGIN
```

```
        P_Test(:szServiceId, :szRetcode, :szReturnMessage);

    END;

    END-EXEC;

    if(sqlca.sqlcode!=0)

    {

        SetLbmError(hHandle, sqlca.sqlcode, 0, "call procedure fail");

        goto LabelExit;

    }

    /*

    KCBP_SetValue(hHandle, "RETCODE", szRetcode);

    KCBP_SetValue(hHandle, "MESSAGE", szReturnMessage);

    */

    SetLbmError(hHandle, atoi(szRetcode), 0, szReturnMessage);

LabelExit:

    KCBP_Commit(hHandle);

    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 subscribe test tool, Version 1.0, Mr. Yuwei Du, 2003.09\n");

    if(argc<2)

    {

        printf("Usage: subscribe topic [timeout servername ip port sendq receiveq user\npassword]\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->KCBPCLI_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->SQLKCBPCLI_DisConnect();  
  
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 subscribe test tool, Version 1.0, Mr. Yuwei Du, 2003.09\n");

if(argc<2)

{

    printf("Usage: subscribe topic [timeout 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->KCBPCLI_SetConnectOption( stKCBPConnection ) )

    {

        delete pKCBPcli;

        printf("pKCBPcli->KCBPCLI_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("receive 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->SQLKCBPCLI_DisConnect();

    delete pKCBPcli;

    return 0;
}
```

## 5.4. 异步调用

```
#include "stdafx.h"
```

```
#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

#include "KCBPcli.h"

int main(int argc, char *argv[])
{
    int          i, nCount = 1, n = 0;
    char         szTmp[1024];
    time_t       td, tdl;
    int          nColNums;
    int          nResultset;
    int          nReturnCode;
    int          nRow;
    char         szopt[10][20];
    KCBPCLIHANDLE hHandle;
    tagCallCtrl  stControl;
    int          nRet;

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

    if(argc < 2)
    {
        printf("Usage: acall transname [count servername ip port sendq receiveq user\npassword]\n");

        printf("example: acall LBMTEST\n");

        printf(" acall LBMTEST 10\n");

        printf(" acall LBMTEST 1 KCBP01\n");
    }
}
```

```
    printf(" acall LBMTTEST 1 KCBP01 192.168.1.20 21000 req2 ans2 9999 888888\n");

    exit(2);

}

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

strcpy(szopt[2], "1");

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

strcpy(szopt[4], "10.240.106.194");

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.nProtocol = 0;

strcpy(stKCBPConnection.szAddress, szopt[4]);

stKCBPConnection.nPort = atoi(szopt[5]);

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

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

```
nRet = KCBPCLI_Init(&hHandle);

if(nRet != 0)

{

    printf("KCBPCLI_Init return %d\n", nRet);

    return 1;

}

if(argc > 4)

{

    nRet = KCBPCLI_SetOptions(hHandle, KCBP_OPTION_CONNECT ,&stKCBPConnection,
sizeof(stKCBPConnection));

    if(nRet != 0)

    {

        printf("KCBPCLI_SetOptions return %d\n", nRet);

        KCBPCLI_Exit(hHandle);

        return 2;

    }

}

nRet = KCBPCLI_SQLConnect(hHandle, szopt[3], szopt[8], szopt[9]);

if(nRet)

{

    printf("KCBPCLI_SQLConnect return %d\n", nRet);

    KCBPCLI_Exit(hHandle);

    return 3;

}

time(&td);

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

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

while(n++ < nCount)
{
    KCBPCLI_BeginWrite(hHandle);

    KCBPCLI_SetValue(hHandle, "QUERYID", "TESTTRAN");

    KCBPCLI_SetValue(hHandle, "RESULTROW", "200000");

    memset(&stControl, 0, sizeof(stControl));

    stControl.nExpiry = 20;

    nReturnCode = KCBPCLI_ACallProgramAndCommit(hHandle, argv[1], &stControl);

    if(nReturnCode != 0)
    {
        KCBPCLI_GetErrorMsg(hHandle, szTmp);

        printf("KCBPCLI_ACallProgramAndCommit error, %d, %s\n", nReturnCode,
szTmp);

        continue;
    }

    stControl.nExpiry = 10;

    nReturnCode = KCBPCLI_GetReply(hHandle, &stControl);

    if(nReturnCode != 0)
    {
        KCBPCLI_GetErrorMsg(hHandle, szTmp);

        printf("KCBPCLI_ACallProgramAndCommit error, %d, %s\n", nReturnCode,
szTmp);

        continue;
    }
}
```

```
nResultset = 0;

do
{
    nReturnCode = KCBPCLI_SQLNumResultCols(hHandle, &nColNums);

    if(nReturnCode != 0)
    {
        printf("unknown resultset colnums, rc=%d\n", nReturnCode);

        break;
    }

    KCBPCLI_SQLGetCursorName(hHandle, szTmp, 32);

    printf("Resultset %d %s \n", ++nResultset, szTmp);

    for(i = 1; i <= nColNums; i++)
    {
        nReturnCode = KCBPCLI_SQLGetColName(hHandle, i, szTmp, sizeof(szTmp)
- 1);

        if(nReturnCode == 0)
            printf("%d=%s ", i, szTmp);

        else
            printf("error %d", nReturnCode);
    }

    printf("\n");

    nRow = 0;

    while(1)
    {
        nReturnCode = KCBPCLI_SQLFetch(hHandle);

        if(nReturnCode == 0)
        {
            // have result
```

```
        printf("Row = %d ", ++nRow);

        for(i = 1; i <= nColNums; i++)

        {

            nReturnCode = KCBPCLI_RsGetCol(hHandle, 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(KCBPCLI_SQLMoreResults(hHandle) == 0);

Labelclosecursor:

    KCBPCLI_SQLCloseCursor(hHandle);

}

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));  
}  
  
KCBPCLI_SQLKCBPCLI_DisConnect(hHandle);  
  
KCBPCLI_Exit(hHandle);  
  
return 0;  
}
```



## 6. 扩展编程

### 6.1. 用户出口

#### 6.1.1. 概述

KCBP 提供了用户出口机制，为用户参与控制 KCBP 的运行提供了一种方法。

KCBP 的用户出口是指 KCBP 在过程中，经过一些预定的出口，在这些出口处，用户可以编制程控制 KCBP 的运行。

用户出口分为客户端出口和服务端出口两类，目前，我们仅提供服务端出口。下面描述中的用户出口，是指服务端用户出口。

用户出口程序是按 KCBP 用户出口编写原则编写的程序，KCBP 提供将自身的一些信息作为输入，用户出口程序对这些输入进行处理，然后输出一些信息给 KCBP。用户出口程序以动态库形式存在，在 Windows 系统中动态库名称是 `userexit.dll`，在 LINUX 系统中动态库名称是 `userexit.so`。在用户出口动态库中包含了所有的用户出口程序。用户出口程序动态库文件要放在 `kcbp` 可执行文件所在目录。

KCBP 中对用户出口依据编号+入口函数名称来管理。每一个用户出口都有一个编号，这个编号主要用于设置，如在 KCBP 系统配置中有一项 `User exit string`，在程序定义中也有一个 `userexitnumber` 的属性，用户出口编号就是用来填写这些设置项的。这些设置项可以包含多个用户出口，比如 KCBP 系统配置项的 `Userexit string` 可以设置为 4,26 两个。KCBP 系统根据设置的用户出口编号调用对应的用户出口函数，编号和入口函数名称对应关系是确定的，已经写到了 KCBP 系统的代码中，用户不可更改。比如用户出口编号 1 对应的入口函数名称是 `KCBP_UE_Startup`。在 KCBP/WIN 中已支持的用户出口编号和入口函数名称列表如下：

编号	出口函数名称	说明

1	KCBP_UE_Startup	系统启动用户出口，在 KCBP 系统启动时被调用一次
2	KCBP_UE_Shutdown	系统关闭用户出口，在 KCBP 系统关闭时被调用一次
3	KCBP_UE_Syncpoint	同步点用户出口，在 LBM 调用 KCBP_Commit、KCBP_RollBack 时被调用
4	KCBP_UE_DumpRequest	记录请求用户出口，当 KCBP 接收到请求时调用
5	KCBP_UE_EndTask	任务结束用户出口，当 LBM 执行结束时被调用
22	KCBP_UE_GetDeputyName	代理名称用户出口，当按 LBM 名称转发时被调用
23	KCBP_UE_DumpAnswer	记录应答用户出口，当 KCBP 发送应答给客户端时被调用
24	KCBP_UE_AS_Startup	服务进程启动用户出口，当每个 KCBPAS 进程或线程启动时被调用一次
25	KCBP_UE_AS_Shutdown	服务进程关闭用户出口，当每个 KCBPAS 进程或线程关闭时被调用一次
26	KCBP_UE_DumpLog	日志用户出口，当 KCBP 系统记录日志时被调用

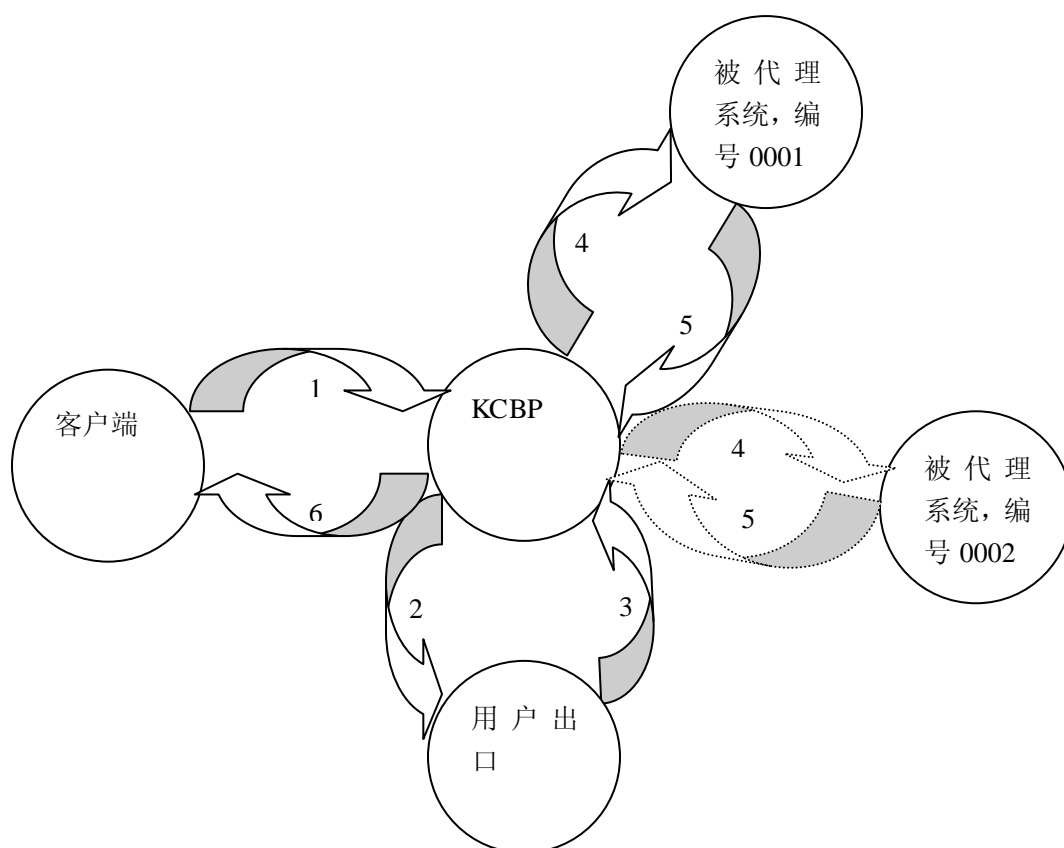
KCBP/LINUX 版的用户如果需要了解用户出口的支持情况，请与 KCBP 项目组联系。

用户出口和 LBM 是两类不同程序，用户出口是 KCBP 本身的扩展程序，而 LBM 是用户编写的业务程序。由于用户出口是和 KCBP 自身的运行相关，因此，它的编写要求相对较高，如果用户出口程序编写质量不高，可能造成 KCBP 运行混乱。

### 6.1.2. 用户出口程序范例

KCBP 中，有一种 LBM，类型是 deputy 或 transfer，它代表另外一个 KCBP 节点或其它系统的一个服务。当一个 KCBP 代理了多个其它系统的服务时，而其它系统完成相同的功能，当 KCBP 收到一个调用代理类型的 LBM 请求时，需要根据请求报文内容中的目的节点编号定位一个目的系统，然后将请求交给目的系统处理。

例如，我们要建设一个系统，KCBP 代理了另外 2 个节点 0001 和 0002 的业务 X，当客户调用 X 服务时，KCBP 采用用户出口查找目的节点编号，将请求转给目的节点处理。系统结构示意图如下：



系统处理流程描述如下：

1. 客户端调用业务 X，将目的节点编号作为参数传递；

2. KCBP 收到 X 请求, 发现是代理请求, 调用取代理名称的用户出口;
3. 用户出口从请求报文中取出目的节点编号, 将其返回给 KCBP;
4. KCBP 根据该节点编号, 找到对应的 XA 定义, 将请求转发给被代理系统;
5. 被代理系统处理该请求, 返回结果给 KCBP;
6. KCBP 将结果返回给客户端;

KCBP 依靠一个用户出口来完成目的系统的判断功能, 这个用户出口的编号是 22, 在名称为 X 的 LBM 的属性 Userexitnumber 中定义, 函数名称是 KCBP\_UE\_GetDeputyName, 实现代码如下:

```
extern          "C"          UE_EXPORTS          KCBP_UE_Return_t          UE_CDECL
KCBP_UE_GetDeputyName(KCBP_UE_Head_t *pUE_Header, void *pUE_Specific)

{

    struct tagGetDeputyNameParam

    {

        char *pszName;

        int nSize;

    } *p;

    p = (struct tagGetDeputyNameParam *) pUE_Specific;

    /* TODO: Add your specialized code here*/

    strncpy(p->pszName, pUE_Header->UE_Workarea+113, 4);

    p->pszName[4] = 0;

    /* get name form pUE_Header->UE_Workarea and copy into pUE_Specific's pszName */

    return(UE_Normal);

}
```

程序说明如下：

- KCBP\_UE\_GetDeputyName 是用户出口函数名称；
- KCBP\_UE\_Head\_t \*pUE\_Header 是用户出口输入参数，其中数据成员 UE\_Workarea 是指向请求报文的指针，请求报文依据 KCBP 的报文数据格式存放；
- void \*pUE\_Specific 是用户出口的输入输出参数，在上例中，我们通过它返回一个节点编号给 KCBP，KCBP 根据该编号使用对应的 XA 转发请求；
- 每个用户出口程序都有一个 KCBP\_UE\_Return\_t 类型的返回值，它是一个 long 类型，UE\_Normal 值为 0，表示成功，其它值表示失败。

### 6.1.3. USEREXIT.H 文件

用户出口程序需要的头文件 UserExit.h 位于 KCBP 安装目录下面的 samples\server\userexit 中。目前版本的 UserExit.h 具体内容如下：

```
/*  
  
* NAME:          UserExit.h, KCBP User Exit header  
  
*  
  
* VERSION:      1.0  
  
*  
  
* AUTHOR:       Mr. Yuwei Du  
  
*  
  
* (C) COPYRIGHT Shenzhen Kingdom Tech. Co. Ltd. 2002  
  
* All Rights Reserved  
  
*  
  
* P.R.C Government Users Restricted Rights - Use, duplication or  
  
* disclosure restricted by GSAADP Schedule Contract with Kingdom Corp.  
  
*  
*/
```

```
*  
  
* DESCRIPTION:  
  
* This file contains all the User Exit public definitions.  
  
*/  
  
#ifndef _USEREXIT_H  
  
#define _USEREXIT_H  
  
  
#define KCBP_UE_HEADER_VERSION 1 /* all version numbers start at 1 */  
  
#define KCBP_UE_NAME_MAX 32 /* max length of UE name */  
  
  
typedef long KCBP_UE_Return_t; /* return code from UE */  
  
#define UE_Normal (KCBP_UE_Return_t) 0  
  
  
typedef struct  
  
{  
  
    int UE_Version;  
  
    char UE_Name[KCBP_UE_NAME_MAX + 1];  
  
    char *UE_Workarea;  
  
} KCBP_UE_Head_t;  
  
  
#define KCBP_UE_ID_STARTUP 1  
  
#define KCBP_UE_STARTUP "KCBP_UE_Startup"  
  
  
#define KCBP_UE_ID_SHUTDOWN 2  
  
#define KCBP_UE_SHUTDOWN "KCBP_UE_Shutdown"
```

```
#define KCBP_UE_ID_SYNCPOINT          3

#define KCBP_UE_SYNCPOINT             "KCBP_UE_Synqpoint"

#define KCBP_UE_ID_DUMPREQUEST        4

#define KCBP_UE_DUMPREQUEST           "KCBP_UE_DumpRequest"

#define KCBP_UE_ID_ENDTASK            5

#define KCBP_UE_ENDTASK               "KCBP_UE_EndTask"

#define KCBP_UE_ID_DISPATCHSTART      6

#define KCBP_UE_DISPATCHSTART         "KCBP_UE_DispatchStart"

#define KCBP_UE_ID_DISPATCHEND        7

#define KCBP_UE_DISPATCHEND           "KCBP_UE_DispatchEnd"

#define KCBP_UE_ID_TRANSFERSTART      8

#define KCBP_UE_TRANSFERSTART         "KCBP_UE_TransferStart"

#define KCBP_UE_ID_TRANSFEREND        9

#define KCBP_UE_TRANSFEREND           "KCBP_UE_TransferEnd"

#define KCBP_UE_ID_BATCHTRANSFERSTART 10

#define KCBP_UE_BATCHTRANSFERSTART    "KCBP_UE_BatchTransferStart"
```

```
#define KCBP_UE_ID_BATCHTRANSFEREND      11

#define KCBP_UE_BATCHTRANSFEREND        "KCBP_UE_BatchTransferEnd"

#define KCBP_UE_ID_BIZSTART              12

#define KCBP_UE_BIZSTART                 "KCBP_UE_BizStart"

#define KCBP_UE_ID_BIZEND                13

#define KCBP_UE_BIZEND                   "KCBP_UE_BizEnd"

#define KCBP_UE_ID_BIZISCSTART           14

#define KCBP_UE_BIZISCSTART              "KCBP_UE_BizISCStart"

#define KCBP_UE_ID_BIZISCEND             15

#define KCBP_UE_BIZISCEND                "KCBP_UE_BizISCEnd"

#define KCBP_UE_ID_DEPUTYSTART           16

#define KCBP_UE_DEPUTYSTART              "KCBP_UE_DeputyStart"

#define KCBP_UE_ID_DEPUTYEND             17

#define KCBP_UE_DEPUTYEND                "KCBP_UE_DeputyEnd"

#define KCBP_UE_ID_MANAGESTART           18

#define KCBP_UE_MANAGESTART              "KCBP_UE_ManageStart"

#define KCBP_UE_ID_MANAGEEND            19
```



```
#define KCBP_UE_MANAGEEND                "KCBP_UE_ManageEnd"

#define KCBP_UE_ID_ALARMSTART            20

#define KCBP_UE_ALARMSTART              "KCBP_UE_AlarmStart"

#define KCBP_UE_ID_ALARMEND              21

#define KCBP_UE_ALARMEND                "KCBP_UE_AlarmEnd"

#define KCBP_UE_ID_GETDEPUTYNAME        22

#define KCBP_UE_GETDEPUTYNAME          "KCBP_UE_GetDeputyName"

#define KCBP_UE_ID_DUMPANSWER           23

#define KCBP_UE_DUMPANSWER              "KCBP_UE_DumpAnswer"

#define KCBP_UE_ID_AS_STARTUP            24

#define KCBP_UE_AS_STARTUP              "KCBP_UE_AS_Startup"

#define KCBP_UE_ID_AS_SHUTDOWN           25

#define KCBP_UE_AS_SHUTDOWN             "KCBP_UE_AS_Shutdown"

#define KCBP_UE_ID_DUMPLOG              26

#define KCBP_UE_DUMPLOG                 "KCBP_UE_DumpLog"

/*

typedef union

{
```

```
    } KCBP_UE_Specific_t;

    KCBP_UE_Return_t      KCBP_UE_Entry(KCBP_UE_Header_t      *UE_Header,
    KCBP_UE_Specific_t *UE_Specific);

    */

    typedef KCBP_UE_Return_t (*UE_FUNC) (KCBP_UE_Head_t *, void *);

    #endif
```

#### 6.1.4. USEREXIT.CPP 文件

KCBP 安装目录下的 samples\server\userexit 的 Userexit.dsp 是 Windows 上用户出口的 VC++工程文件，可以直接使用它创建用户出口程序。其中包含了用户出口程序框架文件 UserExit.cpp，具体内容如下：

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <time.h>

#if defined(WIN32)

#include <windows.h>

#endif

#include "UserExit.h"

#if defined(WIN32)

#ifdef USEREXIT_EXPORTS

#define UE_EXPORTS __declspec(dllexport)

#else
```

```
#define UE_EXPORTS __declspec(dllimport)

#endif

#define UE_CDECL __cdecl

#else

#define UE_CDECL /* leave blank for other systems */

#define UE_EXPORTS

#endif

#if defined(WIN32)

BOOL WINAPI DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)

{

    switch(ul_reason_for_call)

    {

        case DLL_PROCESS_ATTACH:

        case DLL_THREAD_ATTACH:

        case DLL_THREAD_DETACH:

        case DLL_PROCESS_DETACH:

            break;

    }

    return TRUE;

}

#endif

int VerifyUEHead(const KCBP_UE_Head_t *pUE_Header, const char *pszUEName)

{

    if(pUE_Header == NULL) return 1;
```

```
    if(pUE_Header->UE_Version != KCBP_UE_HEADER_VERSION) return 2;

    if(strcmp(pUE_Header->UE_Name, pszUEName) != 0) return 3;

    return 0;
}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_Startup(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)
{
    /* TODO: Add your specialized code here */

    return(UE_Normal);
}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_Shutdown(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)
{
    /* TODO: Add your specialized code here */

    return(UE_Normal);
}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_AS_Startup(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)
{
    /* TODO: Add your specialized code here */

    return(UE_Normal);
}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_AS_Shutdown(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_Syncpoint(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
/*
```

userexit.h 中定义:

记录请求日志的用户出口编号是 4, 处理函数是 KCBP\_UE\_DumpRequest

```
#define KCBP_UE_ID_DUMPREQUEST 4
```

```
#define KCBP_UE_DUMPREQUEST "KCBP_UE_DumpRequest"
```

记录第一个应答包的用户出口编号是 23, 处理函数是 KCBP\_UE\_DumpAnswer

```
#define KCBP_UE_ID_DUMPANSWER 23
```

```
#define KCBP_UE_DUMPANSWER "KCBP_UE_DumpAnswer"
```

配置方法是在 KCBPSetup 界面 system 页面 miscellaneous 组配置 userexit string 为 4, 23

KCBP\_UE\_DumpRequest 和 KCBP\_UE\_DumpAnswer 的函数输出到调试器窗口, 实现示范如下:

```
*/

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DumpRequest(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    /*

    char szMsg[4096];

    struct tagUERequest

    {

        int nNodeId;

        int nQueueId;

        char MsgId[25];

        int nDataLen;

        char Buffer[2048];

    } *p;

    p = (tagUERequest *)pUE_Specific;

    _snprintf(szMsg, sizeof(szMsg), "Req: NodeId=%d, QueueId=%d, MsgId=%s, Len=%d,
Buf=%s",

        p->nNodeId, p->nQueueId, p->MsgId, p->nDataLen, p->Buffer);

    OutputDebugString(szMsg);

    */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DumpAnswer(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    /*

char szMsg[4096];

struct tagUERequest

{

    int nNodeId;

    int nQueueId;

    char MsgId[25];

    int nDataLen;

    char Buffer[2048];

} *p;

p = (tagUERequest *)pUE_Specific;

_sprintf(szMsg, sizeof(szMsg), "Ans: NodeId=%d, QueueId=%d, MsgId=%s, Len=%d,
Buf=%s",

        p->nNodeId, p->nQueueId, p->MsgId, p->nDataLen, p->Buffer);

OutputDebugString(szMsg);

*/

return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DumpLog(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)
```

```
{

    /* TODO: Add your specialized code here */

    /*

    char szMsg[4096];

    struct tagLog

    {

        int nType;

        int nLevel;

        int nReserved;//tickcount

        int nProcessId;

        int nThreadId;

        time_t tTime;

        char szMsg[80 + 1];

    } *p;

    p = (struct tagLog *)pUE_Specific;

    _snprintf(szMsg, sizeof(szMsg),

        "%d %d %d %d %d %ld %s",

        p->nType,

        p->nLevel,

        p->nReserved,

        p->nProcessId,

        p->nThreadId,

        p->tTime,

        p->szMsg);

    OutputDebugString(szMsg);
```



```
    */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_EndTask(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DispatchStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DispatchEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_TransferStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_TransferEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_BatchTransferStart

(

    KCBP_UE_Head_t *pUE_Header,

    void *pUE_Specific

)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL
```

```
KCBP_UE_BatchTransferEnd(KCBP_UE_Head_t *pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_BizStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_BizEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_BizISCSstart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
    }

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_BizISCEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DeputyStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_DeputyEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_ManageStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{
```

```
    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_ManageEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_AlarmStart(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}

extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_AlarmEnd(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    /* TODO: Add your specialized code here */

    return(UE_Normal);

}
```

```
extern "C" UE_EXPORTS KCBP_UE_Return_t UE_CDECL KCBP_UE_GetDeputyName(KCBP_UE_Head_t
*pUE_Header, void *pUE_Specific)

{

    struct tagGetDeputyNameParam

    {

        char *pszName;

        int nSize;

    }*p;

    p = (struct tagGetDeputyNameParam *) pUE_Specific;

    /* TODO: Add your specialized code here*/

    /* get name form pUE_Header->UE_Workarea and copy into pUE_Specific's pszName */

    return(UE_Normal);

}
```

## 6.2. XA 编程

### 6.2.1. KCBPXA 简介

KCBP 通过 KCBPXA 规范访问各种资源，包括数据库资源、通讯资源、其他 KCBP 等。KCBPXA 接口本身以动态库形式存在，通过 KCBP 的 XA Resource 配置来通知 KCBP 其自身的基本信息，包括名称、路径、入口、打开参数、关闭参数、选项参数等。

KCBP 系统本身已经提供了一些 XA 接口，包括：

- 访问 DB2 的 db21pc 接口
- 访问 Oracle 的 Oracle1pc 接口
- 访问 SQL Server 的 ODBC1PC 接口

- 访问其他 KCBP 的 KCBPXA 接口
- 访问 MQ 的 mqxa 接口
- 访问 KCXP 的 KCXPXA 接口

当这些接口无法满足需求时，用户可以开发既符合 KCBPXA 规范，又满足需求的新接口。

## 6.2.2. KCBPXA 分类

依据功能划分，有 3 类：

- 数据源访问 XA 接口
- 数据源访问 IPC 接口
- 通讯资源访问接口

## 6.2.3. KCBPXA 规范中的核心数据结构

```
struct KCBP_xa_switch_t {  
  
    char name[RMNAMESZ];        /* name of resource manager */  
  
    long flags;                 /* resource manager specific options */  
  
    long version;               /* must be 0 */  
  
#ifdef _TMPROTOTYPES  
  
    int (*xa_open_entry)(void *, char *, int, long);    /* xa_open function pointer */  
  
    int (*xa_close_entry)(void *, char *, int, long);   /* xa_close function pointer*/  
  
    int (*xa_start_entry)(void *, XID *, int, long); /* xa_start function pointer */  
  
    int (*xa_end_entry)(void *, XID *, int, long); /* xa_end function pointer */  
  
    int (*xa_rollback_entry)(void *, XID *, int, long); /* xa_rollback function pointer */  
  
    int (*xa_prepare_entry)(void *, XID *, int, long); /* xa_prepare function pointer */  
  
    int (*xa_commit_entry)(void *, XID *, int, long); /* xa_commit function pointer */  
  
    int (*xa_recover_entry)(void *, XID *, long, int, long);  
  
        /* xa_recover function pointer*/  
  
    int (*xa_forget_entry)(void *, XID *, int, long); /* xa_forget function pointer */  
  
    int (*xa_complete_entry)(void *, int *, int *, int, long);
```

```
        /* xa_complete function pointer */

#else /* ifndef _TMPROTOTYPES */

    int (*xa_open_entry);    /* xa_open function pointer */

    int (*xa_close_entry);  /* xa_close function pointer */

    int (*xa_start_entry); /* xa_start function pointer */

    int (*xa_end_entry); /* xa_end function pointer */

    int (*xa_rollback_entry); /* xa_rollback function pointer */

    int (*xa_prepare_entry); /* xa_prepare function pointer */

    int (*xa_commit_entry); /* xa_commit function pointer */

    int (*xa_recover_entry); /* xa_recover function pointer */

    int (*xa_forget_entry); /* xa_forget function pointer */

    int (*xa_complete_entry); /* xa_complete function pointer */

#endif /* ifndef _TMPROTOTYPES */

    int (*xa_init_entry)(void **);

    int (*xa_exit_entry)(void *);

    int (*xa_setopt_entry)( void *handle, char *option, char *str);

    int (*xa_read_entry)( void *handle, char **str, int *len);

    int (*xa_write_entry)( void *handle, char *str, int len);

    int (*ax_reg_entry)(void *handle, int rmid, XID *xid, long flags);

    int (*ax_unreg_entry)(void *handle, int rmid, long flags);

};
```

## 6.2.4. KCBP 对 XA 接口的调度流程

### 1. KCBP 系统初始化阶段



- 1) 将配置文件中设置的 switchloadfile 装入内存
  - 2) 调用配置文件中设置的 entry 函数
  - 3) 调用 xa\_init\_entry, 获得并保存 pXACA
  - 4) 调用 xa\_open\_entry 函数
  - 5) 调用 xa\_setopt\_entry 函数
2. KCBP 系统关闭阶段
- 1) 调用 xa\_close\_entry 函数
  - 2) 调用 xa\_exit\_entry, 释放 pXACA
  - 3) 将 switchloadfile 从内存中释放
3. KCBP 运行阶段, LBM 对 XA 接口的使用
- 1) LBM 调用 KCBP\_Commit 时, KCBP 会调用 xa\_commit\_entry 函数
  - 2) LBM 调用 KCBP\_RollBack 时, KCBP 会调用 xa\_rollback\_entry 函数

## 6.2.5. KCBPXA+介绍

KCBPXA+是 KCBP 对 XA 规范的扩展。

### 6.2.5.1. LBM 程序中如何获得 XA 的指针并调用类中的方法

在 LBM 程序, 调用 KCBP\_XASelect 取得 pXACA, 然后就可以调用类中的方法。

### 6.2.5.2. 功能类如何与 KCBP 交互

- a) 实现一个功能类 (如用于访问数据库的类)
- b) 在 xa\_init\_entry 中, 实例化这个类, 并将地址通过 pXACA 的返回
- c) LBM 程序中, 调用 KCBP\_XASelect 获得类指针, 调用类方法
- d) 在 xa\_exit\_entry 中, 将这个类实例释放掉

### 6.2.5.3. 虚类化的必要性

为了实现类的实现代码与 LBM 无关, 可以通过虚类实现, 参见 ODBC1PC 和 SemArray 的实现。

#### 6.2.5.4. 资源断开重连的约定

`xa_commit_entry` 和 `xa_rollback_entry` 实现函数返回 `XAER_RMFAIL` 将触发 KCBP 释放资源并重新初始化该资源。

#### 6.2.6. 范例 1: DB2IPC 的实现

DB2IPC 是 KCBP 中实现的对 DB2 进行一阶段提交访问的接口。该程序适合多进程方式运行的 KCBP，对多线程方式运行的 KCBP 不适合，适合多线程的接口是 `db2ipc_r`，其内容未在这个手册中列出。

```
/*  
  
Copyright (c) 2003 Shenzhen Kingdom Tech. Co. Ltd.  
  
All rights reserved  
  
Author :Mr. Yuwei Du  
  
db2ipc.cpp, this module implement a one phrase db2 interface  
  
Version 1.0, 20031128  
  
*/  
  
#include "stdafx.h"  
  
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <string.h>  
  
#include <sql.h>  
  
#include <sqlca.h>  
  
#include "Crypt.h"  
  
#include "KCBPXA.h"
```

```
#ifdef WIN32

#define DLLIMPORT __declspec(dllimport)

#define DLLEXPORT __declspec(dllexport)

#define KCBPCDECL __cdecl

#if defined(__cplusplus)

extern "C" {

#endif

        DLLEXPORT struct KCBP_xa_switch_t * KCBPCDECL KCBP_DB21PC(void);

#if defined(__cplusplus)

}

#endif

#else

#define DLLIMPORT

#define DLLEXPORT

#define KCBPCDECL

#if defined(__cplusplus)

extern "C" {

#endif

        struct KCBP_xa_switch_t * KCBP_DB21PC(void);

#if defined(__cplusplus)

}

#endif

#endif

#endif
```

```
int db2_no_xa_OnePhaseInit(void **pXACA);

int db2_no_xa_OnePhaseExit(void * pXACA);

/*-----*/

/*function prototypes                                     */

/*-----*/

int KCBPCDECL db2_no_xa_open(void *, char *, int, long);

int KCBPCDECL db2_no_xa_close(void *, char *, int, long);

int KCBPCDECL db2_no_xa_start(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_end(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_rollback(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_prepare(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_commit(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_recover(void *, XID *, long, int, long);

int KCBPCDECL db2_no_xa_forget(void *, XID *, int, long);

int KCBPCDECL db2_no_xa_complete(void *, int *, int *, int, long);

int KCBPCDECL db2_no_xa_setopt(void *handle, char *option, char *str);

int KCBPCDECL db2_no_xa_read(void *handle, char **str, int *len);

int KCBPCDECL db2_no_xa_write(void *handle, char *, int len);

int API_ReadField(char * pszInput, int nIndex, char * pszOutput, int nLen, char chTerm);

class CDB2EsqI

{

public:
```

```
struct sqlca sqlca;

void *m_pContext;

CDB2Esq1(){m_pContext=NULL;}

~CDB2Esq1(){ }

int no_xa_open(char *, int, long);

int no_xa_close(char *, int, long);

int no_xa_start(XID *, int, long);

int no_xa_end(XID *, int, long);

int no_xa_rollback(XID *, int, long);

int no_xa_prepare(XID *, int, long);

int no_xa_commit(XID *, int, long);

int no_xa_recover(XID *, long, int, long);

int no_xa_forget(XID *, int, long);

int no_xa_complete(int *, int *, int, long);

int do_sql_error(int *, char *, int);

};

/*-----*/

/*global storage areas                                     */

/*-----*/

DLLEXPORT struct KCBP_xa_switch_t db2noxaswitch = {

    "DB2 Single Phase",

    0,

    0,

    db2_no_xa_open,
```

```
    db2_no_xa_close,  
  
    db2_no_xa_start,  
  
    db2_no_xa_end,  
  
    db2_no_xa_rollback,  
  
    db2_no_xa_prepare,  
  
    db2_no_xa_commit,  
  
    db2_no_xa_recover,  
  
    db2_no_xa_forget,  
  
    db2_no_xa_complete,  
  
    db2_no_xa_OnePhaseInit,  
  
    db2_no_xa_OnePhaseExit,  
  
    db2_no_xa_setopt,  
  
    db2_no_xa_read,  
  
    db2_no_xa_write  
};  
  
#ifdef WIN32  
  
BOOL WINAPI DllMain( HANDLE hModule,  
  
                    DWORD ul_reason_for_call,  
  
                    LPVOID lpReserved  
                    )  
  
{  
  
    switch (ul_reason_for_call)  
  
    {  
  
        case DLL_PROCESS_ATTACH:
```

```
        case DLL_THREAD_ATTACH:

        case DLL_THREAD_DETACH:

        case DLL_PROCESS_DETACH:

                break;

    }

    return TRUE;

}

#endif

/*****

/*      KCBP_DB21PC()          -- entry point --          */

*****/

#ifdef __cplusplus

extern "C"

#endif

DLLEXPORT struct KCBP_xa_switch_t * KCBPCDECL KCBP_DB21PC(void)

{

    return((KCBP_xa_switch_t *)&db2noxaswitch);

}

/*****

/*      do_sql_error()

*/

*****/

int CDB2Esq1::do_sql_error(int *errorcode, char *errmsg, int len)
```

```
{

    *errorcode=sqlca.sqlcode;

    sqlaintp(errormsg, len, 0, &sqlca); /* Get message text */

    return(XA_OK);

}

/*****

/*      no_xa_open()
*/

*****/

int CDB2Esq!:no_xa_open(char *str1, int i0, long i1)

{

    EXEC SQL BEGIN DECLARE SECTION;

        char database[9] = "";

        char user[9] = "";

        char password[19] = "";

    EXEC SQL END DECLARE SECTION;

    char szCipherPassword[32];

    char openstring[MAXINFOSIZE];

    int ret = XA_OK;

    char *ev;

    /*

    * check that DB2INSTANCE is set
```



```
    */

    if ( (ev = (char *)getenv("DB2INSTANCE")) == NULL )

    {

        return(XAER_RMERR);

    }

    /*

    * Get database, user name and password from open string

    */

    strncpy(openstring, str1, sizeof(openstring));

    openstring[MAXINFOSIZE - 1] = '\0';

    //sscanf(openstring, "%8[^,],%8[^,],%18s", database, user, password);

    API_ReadField(str1, 1, database, sizeof(database)-1, ',');

    API_ReadField(str1, 2, user, sizeof(user)-1, ',');

    API_ReadField(str1, 3, szCipherPassword, sizeof(szCipherPassword)-1, ',');

    DecryptPassword(user, szCipherPassword, password, sizeof(password)-1);

    /*

    * Connect to database

    */

    if (*user != '\0')

    {

        EXEC SQL CONNECT TO :database USER :user USING :password;

    }

    else
```

```
{  
  
    EXEC SQL CONNECT TO :database;  
  
}  
  
/*  
  
    * Check return code from CONNECT  
  
*/  
  
if (sqlca.sqlcode != 0)  
  
    {  
  
        return(XAER_RMERR);  
  
    }  
  
return(ret);  
  
}  
  
/*****  
  
/*      no_xa_close()  
*/  
  
/*****  
  
int CDB2Esq::no_xa_close(char *str1, int i0, long i1)  
  
    {  
  
        EXEC SQL CONNECT RESET;  
  
        return(XA_OK);  
  
    }
```

```

/*****
*/
/*      no_xa_start()                               */
*/
/*****

int CDB2EsqL::no_xa_start(XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      no_xa_end()
*/
*/
/*****

int CDB2EsqL::no_xa_end(XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      no_xa_rollback()                             */
*/
/*****

int CDB2EsqL::no_xa_rollback(XID *str1, int i0, long i1)

{

    EXEC SQL ROLLBACK WORK;

    if( sqlca.sqlcode== -900 || sqlca.sqlcode == -1224 || sqlca.sqlcode== -1024 || sqlca.sqlcode== -30081 )
//lose connection

```

```
    {

        return XAER_RMFAIL;

    }

    return((sqlca.sqlcode < 0 ? XAER_RMERR : XA_OK));

}

/*****

/*      no_xa_prepare()
*/

/*****

int CDB2Esq!:no_xa_prepare(XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      no_xa_commit()
*/

/*****

int CDB2Esq!:no_xa_commit(XID *str1, int i0, long i1)

{

    EXEC SQL COMMIT WORK;

    if( sqlca.sqlcode===-900 || sqlca.sqlcode == -1224 || sqlca.sqlcode===-1024 || sqlca.sqlcode===-30081 )
//lose connection

    {
```

```
        return XAER_RMFAIL;

    }

    return((sqlca.sqlcode < 0 ? XAER_RMERR : XA_OK));

}

/*****

/*      no_xa_recover()
*/

*****/

int CDB2Esq::no_xa_recover(XID *str1, long i0, int i1, long i2)

{

    return(XA_OK);

}

/*****

/*      no_xa_forget()
*/

*****/

int CDB2Esq::no_xa_forget(XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      no_xa_complete()                                     */
```

```

/*****

int CDB2EsqL::no_xa_complete(int *i0, int *i1, int i2, long i3)

{

    return(XA_OK);

}

/*****

/*      KCBP customize initialisation function      */

/*****

int db2_no_xa_OnePhaseInit(void **pXACA)

{

    *pXACA = (void *) new CDB2EsqL;

    if(*pXACA==NULL) return 1;

    return(0);

}

int db2_no_xa_OnePhaseExit(void * pXACA)

{

    if(pXACA==NULL) return 1;

    delete (CDB2EsqL*) pXACA;

    return(0);

}

/*****

/*      db2_no_xa_open()

*/
```

```

/*****/

int KCBPCDECL db2_no_xa_open(void *handle, char *str1, int i0, long i1)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq1=(CDB2Esq1 *)handle;

    return pCDB2Esq1->no_xa_open(str1, i0, i1);

}

/*****/

/*    db2_no_xa_close() */

/*****/

int KCBPCDECL db2_no_xa_close(void *handle, char *str1, int i0, long i1)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq1=(CDB2Esq1 *)handle;

    return pCDB2Esq1->no_xa_close(str1, i0, i1);

}

/*****/

/*    db2_no_xa_start() */

/*****/

int KCBPCDECL db2_no_xa_start(void *handle, XID *str1, int i0, long i1)

{
```

```
CDB2Esq1 *pCDB2Esq1;

if(handle==NULL) return XAER_INVALID;

pCDB2Esq1=(CDB2Esq1 *)handle;

return pCDB2Esq1->no_xa_start(str1, i0, i1);

}

/*****

/*   db2_no_xa_end()
*/

*****/

int KCBPCDECL db2_no_xa_end(void *handle, XID *str1, int i0, long i1)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq1=(CDB2Esq1 *)handle;

    return pCDB2Esq1->no_xa_end(str1, i0, i1);

}

/*****

/*   db2_no_xa_rollback()
*/

*****/

int KCBPCDECL db2_no_xa_rollback(void *handle, XID *str1, int i0, long i1)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;
```



```
    pCDB2Esq=(CDB2Esq *)handle;

    return pCDB2Esq->no_xa_rollback(str1, i0, i1);

}

/*****

/*   db2_no_xa_prepare()
*/

/*****

int KCBPCDECL db2_no_xa_prepare(void *handle, XID *str1, int i0, long i1)

{

    CDB2Esq *pCDB2Esq;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq=(CDB2Esq *)handle;

    return pCDB2Esq->no_xa_prepare(str1, i0, i1);

}

/*****

/*   db2_no_xa_commit()
*/

/*****

int KCBPCDECL db2_no_xa_commit(void *handle, XID *str1, int i0, long i1)

{

    CDB2Esq *pCDB2Esq;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq=(CDB2Esq *)handle;
```

```
        return pCDB2Esq->no_xa_commit(str1, i0, i1);

    }

/*****

/*   db2_no_xa_recover()
*/

*****/

int KCBPCDECL db2_no_xa_recover(void *handle, XID *str1, long i0, int i1, long i2)

{

    CDB2Esq *pCDB2Esq;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq=(CDB2Esq *)handle;

    return pCDB2Esq->no_xa_recover(str1, i0, i1, i2);

}

/*****

/*   db2_no_xa_forget()
*/

*****/

int KCBPCDECL db2_no_xa_forget(void *handle, XID *str1, int i0, long i1)

{

    CDB2Esq *pCDB2Esq;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq=(CDB2Esq *)handle;

    return pCDB2Esq->no_xa_forget(str1, i0, i1);

}
```

```
}

/*****

/*   db2_no_xa_complete()                               */

*****/

int KCBPCDECL db2_no_xa_complete(void *handle, int *i0, int *i1, int i2, long i3)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq1=(CDB2Esq1 *)handle;

    return pCDB2Esq1->no_xa_complete(i0, i1, i2, i3);

}

/*****

/*   db2_do_sql_error()                               */

*****/

int KCBPCDECL db2_do_sql_error(void *handle, int *errcode, char *errmsg, int len)

{

    CDB2Esq1 *pCDB2Esq1;

    if(handle==NULL) return XAER_INVALID;

    pCDB2Esq1=(CDB2Esq1 *)handle;

    return pCDB2Esq1->do_sql_error(errcode, errmsg, len);

}

*****/
```

```
/* db2_no_xa_setopt()
*/

/*****

int KCBPCDECL db2_no_xa_setopt(void *handle, char *option, char *str)

{

    return(XAER_INVALFUNC);

}

/*****

/* db2_no_xa_read() */

/*****

int KCBPCDECL db2_no_xa_read(void *handle, char **str, int *len)

{

    return(XAER_INVALFUNC);

}

/*****

/* db2_no_xa_write() */

/*****

int KCBPCDECL db2_no_xa_write(void *handle, char *str, int len)

{

    return(XAER_INVALFUNC);

}
```

## 6.2.7. 范例 2: ODBC1PC 的实现

ODBC1PC 是 KCBP 中实现的一个用 ODBC 方式访问数据库的接口。

实现代码包括 4 部分:

- KCBPXA 定义文件 KCBPXA.h
- 虚类定义 KCBPDatabase.h;
- 功能类定义 odbc1pc.h;
- 功能实现 odbc1pc.cpp。

### 6.2.7.1. KCBPXA.h

```
/*
Copyright (c) 2002 SZKingdom Co. Ltd.
All rights reserved
Author : Mr. Yuwei Du
*/
#ifndef XA_H
#define XA_H

#ifndef _TMPROTOTYPES
#define _TMPROTOTYPES 1
#endif

/*
 * Transaction branch identification: XID
 */
#ifndef XIDDATASIZE
#define XIDDATASIZE 128 /* size in bytes */
#define MAXGTRIDSIZE 64 /* maximum size in bytes of gtrid */
#define MAXBQUALSIZE 64 /* maximum size in bytes of bqual */
struct xid_t {
    long formatID; /* format identifier */
    long gtrid_length; /* value not to exceed 64 */
    long bqual_length; /* value not to exceed 64 */
    char data[XIDDATASIZE];
};
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
#endif
```

```
/*
 * Declarations of routines by which RM s call TM s:
 */

#if defined(__cplusplus)
extern "C" {
#endif

#ifdef _TMPROTOTYPES
extern int ax_reg(int, XID *, long);
extern int ax_unreg(int, long);
#else /* ifndef _TMPROTOTYPES */
extern int ax_reg();
extern int ax_unreg();
#endif /* ifndef _TMPROTOTYPES */

/*
 * XA Switch Data Structure
 */
#ifdef RMNAMESZ
#define RMNAMESZ 32          /* length of resource manager name, */
                          /* including the null terminator */
#define MAXINFOSIZE 256    /* maximum size in bytes of xa_info strings, */
                          /* including the null terminator */
struct KCBP_xa_switch_t {
    char name[RMNAMESZ];    /* name of resource manager */
    long flags;             /* resource manager specific options */
    long version;          /* must be 0 */
#ifdef _TMPROTOTYPES
    int (*xa_open_entry)(void *, char *, int, long); /* xa_open function pointer */
    int (*xa_close_entry)(void *, char *, int, long); /* xa_close function pointer*/
    int (*xa_start_entry)(void *, XID *, int, long); /* xa_start function pointer */
    int (*xa_end_entry)(void *, XID *, int, long); /* xa_end function pointer */
    int (*xa_rollback_entry)(void *, XID *, int, long); /* xa_rollback function pointer */
    int (*xa_prepare_entry)(void *, XID *, int, long); /* xa_prepare function pointer */
    int (*xa_commit_entry)(void *, XID *, int, long); /* xa_commit function pointer */
    int (*xa_recover_entry)(void *, XID *, long, int, long);
                          /* xa_recover function pointer*/
    int (*xa_forget_entry)(void *, XID *, int, long); /* xa_forget function pointer */
    int (*xa_complete_entry)(void *, int *, int *, int, long);
                          /* xa_complete function pointer */
#else /* ifndef _TMPROTOTYPES */
    int (*xa_open_entry)(); /* xa_open function pointer */
    int (*xa_close_entry)(); /* xa_close function pointer */
    int (*xa_start_entry)(); /* xa_start function pointer */

```

```
int (*xa_end_entry)(); /* xa_end function pointer */
int (*xa_rollback_entry)(); /* xa_rollback function pointer */
int (*xa_prepare_entry)(); /* xa_prepare function pointer */
int (*xa_commit_entry)(); /* xa_commit function pointer */
int (*xa_recover_entry)(); /* xa_recover function pointer */
int (*xa_forget_entry)(); /* xa_forget function pointer */
int (*xa_complete_entry)(); /* xa_complete function pointer */
#endif /* ifndef _TMPROTOTYPES */

int (*xa_init_entry)(void **);
int (*xa_exit_entry)(void *);
int (*xa_setopt_entry)( void *handle, char *option, char *str);
int (*xa_read_entry)( void *handle, char **str, int *len);
int (*xa_write_entry)( void *handle, char *str, int len);
int (*ax_reg_entry)(void *handle, int rmid, XID *xid, long flags);
int (*ax_unreg_entry)(void *handle, int rmid, long flags);
};
#endif

#if defined(__cplusplus)
}
#endif

/*
 * Flag definitions for the RM switch
 */
#ifndef TMNOFLAGS
#define TMNOFLAGS 0x0000000L /* no resource manager features
 * selected
 */
#define TMREGISTER 0x00000001L /* resource manager dynamically
 * registers
 */
#define TMNOMIGRATE 0x00000002L /* resource manager does not support
 * association migration
 */
#define TMUSEASYNC 0x00000004L /* resource manager supports
 * asynchronous operations
 */
#endif

/*
 * Flag definitions for xa_ and ax_ routines
 */
/* use TMNOFLAGS, defined above, when not specifying other flags */
#ifndef TMASYNC
#define TMASYNC 0x80000000L /* perform routine asynchronously */
```

```
#define TMONEPHASE    0x40000000L    /* caller is using one-phase commit
    * optimisation
    */

#define TMFAIL        0x20000000L    /* dissociates caller and marks
    * transaction branch rollback-only
    */

#define TMNOWAIT      0x10000000L    /* return if blocking condition exists */
#define TMRESUME      0x08000000L    /* caller is resuming association
    * with suspended transaction branch
    */

#define TMSUCCESS    0x04000000L    /* dissociate caller from transaction
    * branch
    */

#define TMSUSPEND     0x02000000L    /* caller is suspending, not ending,
    * association
    */

#define TMSTARTRSCAN  0x01000000L    /* start a recovery scan */
#define TMENDRSCAN    0x00800000L    /* end a recovery scan */
#define TMMULTIPLE    0x00400000L    /* wait for any asynchronous operation */
#define TMJOIN        0x00200000L    /* caller is joining existing
    * transaction branch
    */

#define TMMIGRATE     0x00100000L    /* caller intends to perform migration */
#endif
/*
 * ax_() return codes (transaction manager reports to resource manager)
 */
#ifndef TM_OK
#define TM_JOIN        2    /* caller is joining existing transaction
    * branch
    */

#define TM_RESUME     1    /* caller is resuming association with
    * suspended transaction branch
    */

#define TM_OK         0    /* normal execution */
#define TMER_TMERR    -1    /* an error occurred in the
    * transaction manager
    */

#define TMER_INVALID  -2    /* invalid arguments were given */
#define TMER_PROTO     -3    /* routine invoked in an improper context */
#endif
/*
 * xa_() return codes (resource manager reports to transaction manager)
 */
#define XA_RBBASE     100    /* The inclusive lower bound of the
```



```
        * rollback codes
        */
#define XA_RBROLLBACK    XA_RBBASE    /* The rollback was caused by an
        * unspecified reason
        */
#define XA_RBCOMMFAIL    XA_RBBASE+1 /* The rollback was caused by a
        * communication failure
        */
#define XA_RBDEADLOCK    XA_RBBASE+2 /* A deadlock was detected */
#define XA_RBINTEGRITY    XA_RBBASE+3 /* A condition that violates the integrity
        * of the resources was detected
        */
#define XA_RBOTHER    XA_RBBASE+4 /* The resource manager rolled back the
        * transaction branch for a reason not
        * on this list
        */
#define XA_RBPROTO    XA_RBBASE+5 /* A protocol error occurred in the
        * resource manager
        */
#define XA_RBTIMEOUT    XA_RBBASE+6 /* A transaction branch took too long */
#define XA_RBTRANSIENT    XA_RBBASE+7 /* May retry the transaction branch */
#define XA_RBEND    XA_RBTRANSIENT /* The inclusive upper bound of the
        * rollback codes
        */

#define XA_NOMIGRATE    9    /* resumption must occur where
        * suspension occurred
        */
#define XA_HEURHAZ    8    /* the transaction branch may have
        * been heuristically completed
        */
#define XA_HEURCOM    7    /* the transaction branch has been
        * heuristically committed
        */
#define XA_HEURRB    6    /* the transaction branch has been
        * heuristically rolled back
        */
#define XA_HEURMIX    5    /* the transaction branch has been
        * heuristically committed and rolled back
        */
#define XA_RETRY    4    /* routine returned with no effect and
        * may be re-issued
        */
#define XA_RDONLY    3    /* the transaction branch was read-only and
        * has been committed
```

```
*/
#define XA_OK      0 /* normal execution */
#define XAER_ASYNC -2 /* asynchronous operation already outstanding */
#define XAER_RMERR -3 /* a resource manager error occurred in the
                       * transaction branch
                       */
#define XAER_NOTA -4 /* the XID is not valid */
#define XAER_INVALID -5 /* invalid arguments were given */
#define XAER_PROTO -6 /* routine invoked in an improper context */
#define XAER_RMFAIL -7 /* resource manager unavailable */
#define XAER_DUPID -8 /* the XID already exists */
#define XAER_OUTSIDE -9 /* resource manager doing work outside */
                       /* global transaction */
#define XAER_INVALIDFUNC -100 /* invalid funtion, unimplemented */
#endif /* ifndef XA_H */
/*
 * End of xa.h header
 */
```

## 6. 2. 7. 2. 虚类定义文件 KCBPDatabase.h

```
#ifndef KCBPDATABASE_H
#define KCBPDATABASE_H

////////////////////////////////////
// KCBPDatabase.h: CKCBPDatabase class defined head
//
// database access abstract virtual base class for DB2 && MSSQL && SYBASE
// Mr. Yuwei Du
// Version 1.0 ,20010822
////////////////////////////////////

#define MAX_COLNUM 100

#define CMD_BUF_SIZE 1024*8

#define MAX_MESSAGE_LENGTH 1024
#define SQLSTATE_SIZE 5
#define MAX_DSN_LENGTH 32

#define MAXCOLNAMELEN 30
#define SQLMAXCHAR 256
```

```
#define MAXSERVERNAME 30
#define MAX_PARAMNUM 100
#define MAX_PARAMLEN 256
#ifndef max
#define max(a,b) (a > b ? a : b)
#endif

#define KDDBSUCC          0
#define KDDBSUCC_NORET   1
#define KDDBSUCC_NODATA  2
#define KDDBERR           3
#define DBSYSERR          4

typedef struct
{
    char Name[MAXCOLNAMELEN+1];           //列名
    short NameLen;                        //列名长度
    short Type;                            //列类型
    unsigned long MaxLength;              //列长度
    short Precision;                      //精度
} SDBCOL;

typedef struct
{
    unsigned char ErrorMessage[MAX_MESSAGE_LENGTH + 1];
    unsigned char SqlState[SQLSTATE_SIZE + 1];
    long SqlCode ;
    short Length;

    int severity;
    INT msgno;
    INT dberr;
    INT oserr;
    char msgtext[200];
    char dberrstr[200];
    char oserrstr[200];
} CDbErrorInfo;

typedef struct
{
    unsigned char ServerName[MAX_DSN_LENGTH + 1];
    unsigned char Description[256];
} CServerList;
```

```
enum DbmsOsType
{
    otNT = 0,
        otUnix,
        otNovell
};
enum DbmsVersion
{
    dvSQL65 = 1,
        dvSQL70 = 2,
        dvMicrosoft = dvSQL65 | dvSQL70,

        dvSystem10 = 0x10000,
        dvSystem11 = 0x20000,
        dvSystem12 = 0x30000,
        dvSybase = dvSystem10 | dvSystem11 | dvSystem12
};

class CKCBPDatabase
{
public:
    CKCBPDatabase(){return;};
    virtual DbmsOsType GetDbmsOsType()=0;
    virtual DbmsVersion GetDbmsVersion()=0;

    virtual bool IsOpen()=0;
    virtual bool ReOpen()=0;
    virtual bool Open(const char *servername, const char *loginID, const char *passWord,const char *appName =
NULL, const char *hostname = NULL, const char *characterSet=NULL)=0;

    virtual bool Use(const char *databaseName)=0;

    virtual int KDSTDRun(const char * formatString)=0;
    virtual bool ExecSql(const char * commandStr)=0;

    virtual bool MoveNext()=0;

    virtual bool EndTran(short int completioncode=0)=0;

    virtual bool Cancel()=0;

    virtual bool Close()=0;

    virtual const char * GetServerName()=0;
    virtual const char * GetLoginID()=0;
```

```
virtual const char * GetLoginPwd()=0;
```

```
virtual USHORT GetColCount()=0;
```

```
virtual const SDBCOL * GetColInfo(int)=0;
```

```
virtual const SDBCOL * GetColInfo(const char *)=0;
```

```
virtual char * GetColData(int)=0;
```

```
virtual char * GetColData(const char *)=0;
```

```
virtual ULONG GetRowCount()=0;
```

```
virtual int GetErrorCode()=0;
```

```
virtual const char * GetErrorMsg()=0;
```

```
virtual const CDbErrorInfo * GetDbErrorInfo()=0;
```

```
virtual USHORT GetServerList(CServerList * serverList)=0;
```

```
virtual void InstallMsgFunc(void (* MsgFunc)(CDbErrorInfo *)=0;
```

```
virtual bool Commit() = 0;
```

```
virtual bool RollbackTran(const char * TranName=NULL) = 0;
```

```
virtual bool BeginTran() = 0;
```

```
virtual bool SaveTran(const char * TranName) = 0;
```

```
virtual bool IsExistTran() = 0;
```

```
virtual int GetRowCountEffectd()=0;
```

```
virtual unsigned long GetStmtDiagRec( CDbErrorInfo * pDbInfo ) = 0;
```

```
virtual unsigned long MoreResults() = 0;
```

public:

```
virtual int BCPBatch()=0;
```

```
virtual int BCPDone()=0;
```

```
virtual bool BCPBind( unsigned char * const pData, int cbIndicator, int cbData, unsigned char * const pTerm,  
int cbTerm, int eDataType, int idxServerCol )=0;
```

```
virtual bool BCPColfmt( int idxUserDataCol, BYTE eUserDataTypes, int cbIndicator, int cbUserData, unsigned  
char * const pUserDataTerm, int cbUserDataTerm, int idxServerCol )=0;
```

```
virtual bool BCPCollen( int cbData, int idxServerCol )=0;
```

```
virtual bool BCPColptr( unsigned char *const pData, int idxServerCol )=0;
```

```
virtual bool BCPColumns( int nColumns )=0;
```

```
virtual bool BCPControl( int eOption, void* iValue )=0;
```

```
virtual bool BCPExec( long &pnRowsProcessed )=0;
```

```
virtual bool BCPInit( const char * szTable, const char * szDataFile, const char * szErrorFile, int eDirection )=0;
```

```
virtual bool BCPMoretext( int cbData, unsigned char *const pData )=0;
```

```
virtual bool BCPReadfmt( const char * szFormatFile )=0;
```

```
virtual bool BCPSendrow()=0;
```

```
        virtual bool BCPWritefmt( const char * szFormatFile )=0;
};

#endif
```

### 6.2.7.3. 功能类头文件 odbc1pc. h

```
////////////////////////////////////
// ODBClib.h: interface for the CODBClib class.
// class CODBClib for access database by ODBC
// version 1.0, Mr. Yuwei Du 200301020
////////////////////////////////////

#ifdef _ODBC
#ifndef __ODBCLIB_H
#define __ODBCLIB_H

#include "stdafx.h"

#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
#include <odbcss.h>
#include <stdlib.h>
#include "KCBPDatabase.h"

class CODBClib : public CKCBPDatabase
{

public:

    DbmsOsType m_dbmsOsType;
    DbmsVersion m_dbmsVersion;

    DbmsOsType GetDbmsOsType() {return m_dbmsOsType;};
    DbmsVersion GetDbmsVersion() {return m_dbmsVersion;};

    void InstallMsgFunc(void (* MsgFunc)(CDBErrorInfo *));

    CODBClib();
    virtual ~CODBClib();

    bool IsOpen();
```

```
    bool ReOpen();
    bool Open(const char *servername, const char *loginID, const char *passWord, const char *appName = NULL,
const char *hostname = NULL, const char *characterSet=NULL);
    bool Use(const char *databaseName);
    bool SetSchema(const char *schema);

    int  KDSTDRun(const char * formatString);
    bool ExecSql(const char * commandStr);

    bool MoveNext();
    bool Cancel();
    bool Close();
    bool EndTran(SQLSMALLINT completioncode=SQL_COMMIT);

    bool Commit();
    bool RollbackTran(const char * TranName=NULL);
    bool BeginTran();
    bool SaveTran(const char * TranName);
    bool IsExistTran();

    int GetRowCountEffectted();
public:
    int BCPBatch();
    int BCPDone();
    bool BCPBind( unsigned char * const pData, int cbIndicator, int cbData, unsigned char * const pTerm, int
cbTerm, int eDataType, int idxServerCol );
    bool BCPColfmt( int idxUserDataCol, BYTE eUserDataTypes, int cbIndicator, int cbUserData, unsigned char *
const pUserDataTerm, int cbUserDataTerm, int idxServerCol );
    bool BCPCollen( int cbData, int idxServerCol );
    bool BCPColptr( unsigned char *const pData, int idxServerCol );
    bool BCPColumns( int nColumns );
    bool BCPControl( int eOption, void* iValue );
    bool BCPExec( long &pRowsProcessed );
    bool BCPInit( const char * szTable, const char * szDataFile, const char * szErrorFile, int eDirection );
    bool BCPMoretext( int cbData, unsigned char *const pData );
    bool BCPReadfmt( const char * szFormatFile );
    bool BCPSendrow();
    bool BCPWritefmt( const char * szFormatFile );

private:
    bool Init();
    bool Login(const char *servername, const char *loginID, const char *passWord);
    bool Bind();
    SQLRETURN terminate( SQLRETURN rc);
    SQLRETURN DBconnect( SQLHANDLE henv, SQLHANDLE * hdbc, SQLCHAR *servername, SQLCHAR
```

```
*loginID, SQLCHAR *passWord);

    unsigned long GetDiagRec( SQLSMALLINT hType, SQLHANDLE hHandle, SQLRETURN rc, CDbErrorInfo
* pDbInfo);

    static void (* MyMsgFunc)(CDbErrorInfo *);

    bool m_IsExistTran;

    int m_errorCode;
    char m_errorMsg[MAX_MESSAGE_LENGTH];

    SQLHANDLE m_henv;                //环境句柄
    SQLHANDLE m_hdbc;                //连接句柄
    SQLHANDLE m_hstmt;               //执行句柄
    SQLRETURN m_rc;
    SQLCHAR m_stmt[CMD_BUF_SIZE];

    long m_counter;                  //存储过程调用计数

    bool m_loginOK;
    SQLSMALLINT m_colCount;           //列数
    SQLINTEGER m_rowCount;           //行数
    CDbErrorInfo m_dbErrorInfo;
    bool m_bStatementReady;
#ifdef _DB2_PARAMS_ARRAY
    SDBCOL m_colInfo[MAX_COLNUM];    //列信息
    struct DBCOLDATA{ SQLINTEGER pcbValue; //列输出长度
        SQLCHAR * rgbValue;           //列信息
    } m_column[MAX_COLNUM]; //行数据
#else
    SDBCOL m_colInfo[MAX_COLNUM];    //列信息
    struct DBCOLDATA{ SQLINTEGER pcbValue; //列输出长度
        SQLCHAR rgbValue[SQLMAXCHAR]; //列信息
    } m_column[MAX_COLNUM]; //行数据
#endif
    char m_serverName[MAXSERVERNAME + 1]; //server 名
    char m_loginID[MAXSERVERNAME + 1]; //用户
    char m_passWord[MAXSERVERNAME + 1]; //密码
    char m_schema[MAXSERVERNAME + 1]; //schema
    char m_databaseName[MAXSERVERNAME + 1]; //current database name
    char m_szEmptyStr[2];

public:
    const char * GetServerName();
    const char * GetLoginID();
```



```
const char * GetLoginPwd();

unsigned long GetStmtDiagRec( CDbErrorInfo * pDbInfo );
unsigned long MoreResults();

USHORT GetColCount();

const SDBCOL * GetColInfo(int);
const SDBCOL * GetColInfo(const char *);

char * GetColData(int);
char * GetColData(const char *);

ULONG GetRowCount();
int GetErrorCode();
const char * GetErrorMsg();
const CDbErrorInfo * GetDbErrorInfo();
USHORT GetServerList(CServerList * serverList);
};

#endif

#endif
```

#### 6.2.7.4. KCBPXA 实现代码 odbc1pc.cpp

```
/*

Copyright (c) 2003 Shenzhen Kingdom Tech. Co. Ltd.

All rights reserved

Author : Mr. Yuwei Du

odbc1pc.cpp

Version 1.0, 20031023

*/

#include "stdafx.h"
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Crypt.h"

#include "odbc.lib.h"

#include "KCBPXA.h"

#ifdef WIN32

#define DLLIMPORT __declspec(dllimport)

#define DLLEXPORT __declspec(dllexport)

#define KCBPCDECL __cdecl

#ifdef __cplusplus

extern "C" {

#endif

        DLLEXPORT struct KCBP_xa_switch_t * KCBPCDECL KCBP_ODBC1PC(void);

#ifdef __cplusplus

}

#endif

#else

#define DLLIMPORT

#define DLLEXPORT

#define KCBPCDECL

#ifdef __cplusplus

extern "C" {
```

```
#endif

struct KCBP_xa_switch_t * KCBP_ODBCIPC(void);

#ifdef __cplusplus
}

#endif

#endif

int odbc_no_xa_OnePhaseInit(void **pXACA);

int odbc_no_xa_OnePhaseExit(void * pXACA);

/*-----*/

/* function prototypes */

/*-----*/

int KCBPCDECL odbc_no_xa_open(void *, char *, int, long);

int KCBPCDECL odbc_no_xa_close(void *, char *, int, long);

int KCBPCDECL odbc_no_xa_start(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_end(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_rollback(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_prepare(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_commit(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_recover(void *, XID *, long, int, long);

int KCBPCDECL odbc_no_xa_forget(void *, XID *, int, long);

int KCBPCDECL odbc_no_xa_complete(void *, int *, int *, int, long);

int KCBPCDECL odbc_no_xa_setopt(void *handle, char *option, char *str);

int KCBPCDECL odbc_no_xa_read(void *handle, char **str, int *len);
```

```
int KCBPCDECL odbc_no_xa_write(void *handle, char *, int len);
```

```
int API_ReadField(char * pszInput, int nIndex, char * pszOutput, int nLen, char chTerm);
```

```
/*-----*/
```

```
/* global storage areas */
```

```
/*-----*/
```

```
DLLEXPORT struct KCBP_xa_switch_t noxaswitch = {
```

```
    "ODBC Single Phase",
```

```
    0,
```

```
    0,
```

```
    odbc_no_xa_open,
```

```
    odbc_no_xa_close,
```

```
    odbc_no_xa_start,
```

```
    odbc_no_xa_end,
```

```
    odbc_no_xa_rollback,
```

```
    odbc_no_xa_prepare,
```

```
    odbc_no_xa_commit,
```

```
    odbc_no_xa_recover,
```

```
    odbc_no_xa_forget,
```

```
    odbc_no_xa_complete,
```

```
    odbc_no_xa_OnePhaseInit,
```

```
    odbc_no_xa_OnePhaseExit,
```

```
    odbc_no_xa_setopt,
```

```
    odbc_no_xa_read,
```

```
        odbc_no_xa_write

};

#ifdef WIN32

BOOL APIENTRY DllMain( HANDLE hModule,

                      DWORD  ul_reason_for_call,

                      LPVOID lpReserved

                      )

{

    switch (ul_reason_for_call)

    {

        case DLL_PROCESS_ATTACH:

        case DLL_THREAD_ATTACH:

        case DLL_THREAD_DETACH:

        case DLL_PROCESS_DETACH:

            break;

    }

    return TRUE;

}

#endif

/*****

/*      KCBP customize initialisation function      */

*****/

int odbc_no_xa_OnePhaseInit(void **pXACA)
```

```
{

    *pXACA = (void *) new COBCLib;

    if(*pXACA==NULL) return 1;

    return(0);

}

int odbc_no_xa_OnePhaseExit(void * pXACA)

{

    if(pXACA==NULL) return 1;

    delete (COBCLib*) pXACA;

    return(0);

}

/*****

/*      KCBP_ODBC1PC()          -- entry point --          */

*****/

#if defined(__cplusplus)

extern "C"

#endif

DLLEXPORT struct KCBP_xa_switch_t * KCBPCDECL KCBP_ODBC1PC(void)

{

    return((KCBP_xa_switch_t *)&noxaswitch);

}

/*****

/*      odbc_no_xa_open()

*****/
```

```
*/

/*****

int KCBPCDECL odbc_no_xa_open(void *handle, char *str1, int i0, long i1)

{

    char szServerName[32];

    char szDatabase[32];

    char szUserId[32];

    char szCipherPassword[32];

    char szPassword[32];

    char szConnectName[32];

    int bRet;

    if(handle==NULL) return XAER_INVALID;

    /*

    sscanf(openstring, "%8[^\,],%8[^\,],%8[^\,],%8[^\,],%18s", database, user, password);

    */

    API_ReadField(str1, 1, szServerName, sizeof(szServerName), ',');

    API_ReadField(str1, 2, szDatabase, sizeof(szDatabase), ',');

    API_ReadField(str1, 3, szUserId, sizeof(szUserId), ',');

    API_ReadField(str1, 4, szCipherPassword, sizeof(szCipherPassword), ',');

    DecryptPassword(szUserId, szCipherPassword, szPassword, sizeof(szPassword)-1);

    API_ReadField(str1, 5, szConnectName, sizeof(szConnectName), ',');
```

```
bRet = ((CODBClib*)handle)->Open(szServerName, szUserId, szPassword) ;

if( bRet == FALSE )

{

    return ((CODBClib*)handle)->GetErrorCode();

}

bRet = ((CODBClib*)handle)->Use( szDatabase);

if( bRet == FALSE )

{

    return ((CODBClib*)handle)->GetErrorCode();

}

return(XA_OK);

}

/*****

/*      odbc_no_xa_close()
*/

/*****

int KCBPCDECL odbc_no_xa_close(void *handle, char *str1, int i0, long i1)

{

    if(handle==NULL) return XAER_INVALID;

    if( !((CODBClib*)handle)->Close() )

    {

        return ((CODBClib*)handle)->GetErrorCode();

    }

    return(XA_OK);

}
```



```
}

/*****

/*      odbc_no_xa_start()                                */

/*****

int KCBPCDECL odbc_no_xa_start(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      odbc_no_xa_end()                                  */
*/

/*****

int KCBPCDECL odbc_no_xa_end(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      odbc_no_xa_rollback()                            */

/*****

int KCBPCDECL odbc_no_xa_rollback(void *handle, XID *str1, int i0, long i1)

{

    if(handle==NULL) return XAER_INVAL;
```

```
        if(!((COBCLib*)handle)->EndTran(1))

        {

            return ((COBCLib*)handle)->GetErrorCode();

        }

        return(XA_OK);

    }

    /**
     *
     *      odbc_no_xa_prepare()
     *
     */

    /**
     *
     *      odbc_no_xa_prepare(void *handle, XID *str1, int i0, long i1)
     *
     */

    int KCBPCDECL odbc_no_xa_prepare(void *handle, XID *str1, int i0, long i1)

    {

        return(XA_OK);

    }

    /**
     *
     *      odbc_no_xa_commit()
     *
     */

    /**
     *
     *      odbc_no_xa_commit(void *handle, XID *str1, int i0, long i1)
     *
     */

    int KCBPCDECL odbc_no_xa_commit(void *handle, XID *str1, int i0, long i1)

    {

        if(handle==NULL) return XAER_INVALID;

        if(!((COBCLib*)handle)->EndTran(0))

        {
```

```
        return ((COBCLib*)handle)->GetErrorCode();

    }

    return(XA_OK);

}

/*****

/*      odbc_no_xa_recover()
*/

/*****

int KCBPCDECL odbc_no_xa_recover(void *handle, XID *str1, long i0, int i1, long i2)

{

    return(XA_OK);

}

/*****

/*      odbc_no_xa_forget()
*/

/*****

int KCBPCDECL odbc_no_xa_forget(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      odbc_no_xa_complete()
```

```
*/

/*****

int KCBPCDECL odbc_no_xa_complete(void *handle, int *i0, int *i1, int i2, long i3)

{

    return(XA_OK);

}

/*****

/*      odbc_no_xa_setopt()
*/

/*****

int KCBPCDECL odbc_no_xa_setopt(void *handle, char *option, char *str)

{

    return(XAER_INVALFUNC);

}

/*****

/*      odbc_no_xa_read()
*/

/*****

int KCBPCDECL odbc_no_xa_read(void *handle, char **str, int *len)

{

    return(XAER_INVALFUNC);

}
```

```
/*  
*****  
*/  
/*      odbc_no_xa_write()  
*/  
/*  
*****  
*/  
  
int KCBPCDECL odbc_no_xa_write(void *handle, char *str, int len)  
  
{  
  
    return(XAER_INVALIDFUNC);  
  
}
```

## 6.2.8. 范例 3: SEMARRAY 的实现

KCBP 提供了一个 semarray.dll, 这是一个 XA 资源, 用来配置一组信号量, 证券交易系统分布式交易类 LBM 可以使用其中的信号量进行同步。在下面程序中, 我们展现 KCBPXA+ 用法。KCBPXA+ 可利用 XA 的机制, 对非数据库资源进行管理, 这些机制包括初始化、退出、句柄取得、成员函数调用等。

### 6.2.8.1. 用法描述

### 6.2.8.2. XA 配置

参考下面行:

```
<xa entry="KCBP_SEMARRAY" name="sem1 "  
switchloadfile="semarray.dll" xaclose=""  
xaopen="size=2,initialcount=3,maxcount=3,name=sem1 " xaoption=""  
xaserial="all_operation"/>
```

其中 xaopen="size=2,initialcount=3,maxcount=3,name=sem1" 各项含义如下:

size 表示信号量数组中包含的信号量对象总数

initialcount 表示每个信号量的初始值

initialcount 表示每个信号量的最大值

name 表示信号量数组的名称

### 6.2.8.3. 操作

LBM 中对信号量的操作通过下面的抽象类完成:

```
class CSemArrayOp
{
public:
    CSemArrayOp(){}
    virtual ~CSemArrayOp(){}
    virtual int Lock(int i, int nTimeout = -1) = 0;
    virtual int Unlock(int i, int nCount = 1, int *pPrevCount = NULL) = 0;
    virtual int Count() = 0;
} *pSem;
```

抽象类的指针通过LBM API中新提供的函数KCBP\_GetXAHandle获取:

```
LBM API int LBMSTDCALL KCBP_GetXAHandle(LBMHANDLE
hHandle, char *szXAName, void **pDBHandle);
```

### 6.2.8.4. 例子程序

LBM 程序示范如下:

```
//...
hHandle = KCBP_Init(pCA);
```

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

CSemArrayOp *pSem

KCBP_GetXAHandle(hHandle, "sem1", (void **)&pSem); //信号量 XA
资源的名称为 sem1

if(pSem == NULL)

{ //未定义信号量资源返回

    //返回失败信息

    goto labelExit;

}

nRet = pSem->Lock(0, 5000); //5 秒超时

if(nRet == 0)

{ //超时取不到信号量

    //返回失败信息

    goto labelExit

}

//业务操作...

pSem->Unlock(0);

//...
```

#### 6.2.8.5. SemArray 实现代码

/\*

Copyright (c) 2005 Shenzhen Kingdom Tech. Co. Ltd.

All rights reserved

Author: Mr. Yuwei Du

SemArray.cpp, this module implement a semaphore array

Version 1.0, 20050530

\*/

```
#include "stdafx.h"
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <direct.h>
```

```
#include "KCBPXA.h"
```

```
#ifdef WIN32
```

```
#define DLLIMPORT __declspec(dllimport)
```

```
#define DLLEXPORT __declspec(dllexport)
```

```
#define KCBPCDECL __cdecl
```

```
#if defined(__cplusplus)
```

```
extern "C"
```

```
{
```

```
#endif
```

```
DLLEXPORT struct KCBP_xa_switch_t * KCBPCDECL KCBP_SEMARRAY (void);
```

```
#if defined(__cplusplus)
```

```
}
```

```
#endif
```

```
#else
```



```
#define DLLIMPORT

#define DLLEXPORT

#define KCBPCDECL

#if defined(__cplusplus)

extern "C"

{

#endif

struct KCBP_xa_switch_t *KCBP_SEMARRAY(void);

#if defined(__cplusplus)

}

#endif

#endif

int SemArray_OnePhaseInit(void **pXACA);

int SemArray_OnePhaseExit(void *pXACA);

/*-----*/

/* function prototypes */

/*-----*/

int KCBPCDECL SemArray_open(void *, char *, int, long);

int KCBPCDECL SemArray_close(void *, char *, int, long);

int KCBPCDECL SemArray_start(void *, XID *, int, long);

int KCBPCDECL SemArray_end(void *, XID *, int, long);

int KCBPCDECL SemArray_rollback(void *, XID *, int, long);
```

```
int KCBPCDECL SemArray_prepare(void *, XID *, int, long);

int KCBPCDECL SemArray_commit(void *, XID *, int, long);

int KCBPCDECL SemArray_recover(void *, XID *, long, int, long);

int KCBPCDECL SemArray_forget(void *, XID *, int, long);

int KCBPCDECL SemArray_complete(void *, int *, int *, int, long);

int KCBPCDECL SemArray_setopt(void *handle, char *option, char *str);

int KCBPCDECL SemArray_read(void *handle, char **str, int *len);

int KCBPCDECL SemArray_write(void *handle, char *, int len);

static int API_ReadField(char *pszInput, int nIndex, char *pszOutput, int nLen, char chTerm);

class CSemArrayOp
{
public:
    CSemArrayOp(){}

    virtual ~CSemArrayOp(){}

    virtual int Lock(int i, int nTimeout = -1) = 0;

    virtual int Unlock(int i, int nCount = 1, int *pPrevCount = NULL) = 0;

    virtual int Count() = 0;

};

class CSemArray : public CSemArrayOp
{
public:
    CSemArray();
```

```
~CSemArray();

private:

    int m_nSize;

    int m_nInitialCount;

    int m_nMaxCount;

    char m_szName[_MAX_PATH];

    HANDLE *m_apHandle;

public:

    int Open(int nSize = 1, int nInitialCount = 1, int nMaxCount = 1, char *pszName = NULL);

    int Close();

    int Lock(int i, int nTimeout = -1);

    int Unlock(int i, int nCount = 1, int *pPrevCount = NULL);

    int Count();

};

CSemArray::CSemArray()

{

    m_nSize = 0;

    m_nInitialCount = 0;

    m_nMaxCount = 0;

    memset(m_szName, 0, sizeof(m_szName));

    m_apHandle = NULL;
```

```
    }

int CSemArray::Open(int nSize, int nInitialCount, int nMaxCount, char *pszName)

{

    int i;

    int nLen;

    char *pName = NULL;

    char szPath[_MAX_PATH];

    char szName[_MAX_PATH];

    m_nSize = 0;

    m_nInitialCount = 0;

    m_nMaxCount = 0;

    memset(m_szName, 0, sizeof(m_szName));

    m_apHandle = NULL;

    if(nSize < 1 || nInitialCount < 0 || nMaxCount <= 0)

    {

        return 1;

    }

    m_nSize = nSize;

    m_nInitialCount = nInitialCount;

    m_nMaxCount = nMaxCount;

    if(pszName)

    {
```

```
        strncpy(m_szName, pszName, sizeof(m_szName));

    }

    _getcwd(szPath, _MAX_PATH);

    nLen = strlen(szPath);

    for(i = 0; i < nLen; i++)

    {

        if(szPath[i] == '\\')

        {

            szPath[i] = '/';

        }

    }

    m_apHandle = (HANDLE *) malloc(sizeof(HANDLE) * m_nSize);

    if(m_apHandle == NULL)

    {

        return 2;

    }

    memset(m_apHandle, 0, sizeof(HANDLE) * m_nSize);

    for(i = 0; i < m_nSize; i++)

    {

        if(pszName == NULL)

        {

            pName = NULL;
```

```
    }

    else

    {

        sprintf(szName, "%s/SemArray/%s/%010d", szPath, pszName, i);

        pName = szName;

    }

    m_apHandle[i] = CreateSemaphore(NULL, m_nInitialCount, m_nMaxCount, pName);

    if(m_apHandle[i] == NULL)

    {

        return 3;

    }

}

return 0;

}

int CSemArray::Close()

{

    int i;

    if(m_apHandle != NULL)

    {

        for(i = 0; i < m_nSize; i++)

        {

            if(m_apHandle[i] != NULL)
```

```
        {  
            CloseHandle(m_apHandle[i]);  
        }  
    }  
  
    free(m_apHandle);  
  
    m_apHandle = NULL;  
  
    }  
  
    return 0;  
  
    }  
  
CSemArray::~CSemArray()  
  
{  
  
    Close();  
  
    }  
  
int CSemArray::Lock(int i, int nTimeout)  
  
{  
  
    if(i >= m_nSize || i < 0 || m_apHandle == NULL || m_apHandle[i] == NULL)  
  
    {  
  
        return 0;  
  
    }  
  
    return WaitForSingleObject(m_apHandle[i], nTimeout);  
  
    }  
  
int CSemArray::Unlock(int i, int nCount, int *pPrevCount)
```

```
{  
  
    if(i >= m_nSize || i < 0 || nCount <= 0 || m_apHandle == NULL || m_apHandle[i] == NULL)  
  
    {  
  
        return 0;  
  
    }  
  
    return ReleaseSemaphore(m_apHandle[i], nCount, (long *)pPrevCount);  
  
}
```

```
int CSemArray::Count()
```

```
{  
  
    return m_nSize;  
  
}
```

```
/*-----*/
```

```
/* global storage areas */
```

```
/*-----*/
```

```
DLLEXPORT struct KCBP_xa_switch_t SemArrayswitch =
```

```
{  
  
    "SEMARRAY",  
  
    0,  
  
    0,  
  
    SemArray_open,  
  
    SemArray_close,
```



```
SemArray_start,  
  
SemArray_end,  
  
SemArray_rollback,  
  
SemArray_prepare,  
  
SemArray_commit,  
  
SemArray_recover,  
  
SemArray_forget,  
  
SemArray_complete,  
  
SemArray_OnePhaseInit,  
  
SemArray_OnePhaseExit,  
  
SemArray_setopt,  
  
SemArray_read,  
  
SemArray_write  
  
};  
  
#ifdef WIN32  
  
BOOL WINAPI DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID  
lpReserved)  
  
{  
  
    switch(ul_reason_for_call)  
  
    {  
  
        case DLL_PROCESS_ATTACH:  
  
        case DLL_THREAD_ATTACH:  
  
        case DLL_THREAD_DETACH:  
  
        case DLL_PROCESS_DETACH:
```

```
        break;

    }

    return TRUE;

}

#endif

/*****

/*      KCBP customize initialisation function      */

*****/

int SemArray_OnePhaseInit(void **pXACA)

{

    *pXACA = (void *) new CSemArray;

    if(*pXACA == NULL) return 1;

    return(0);

}

int SemArray_OnePhaseExit(void *pXACA)

{

    if(pXACA == NULL) return 1;

    delete (CSemArray *) pXACA;

    return(0);

}
```

```

/*****
/

/*      KCBP_SEMARRAY()          -- entry point --          */

/*****

#ifdef __cplusplus

extern "C"

#endif

DLLEXPORT struct KCBP_xa_switch_t *KCBPCDECL KCBP_SEMARRAY(void)

{

    return((KCBP_xa_switch_t *) &SemArrayswitch);

}

/*****

/*      SemArray_open()          */

/*****

int KCBPCDECL SemArray_open(void *handle, char *str1, int i0, long i1)

{

/*

str1 format 1: size=1,initialcount=1,maxcount=1,name=sem1

*/

    CSemArray *pCSemArray;
```

```
int nRet;

int i;

char szParamName[20];

char szParamValue[20];

int nSize = 0;

int nInitialCount = 0;

int nMaxCount = 0;

char szName[_MAX_PATH];

char szTmp[_MAX_PATH];

char *pszName;

if(handle == NULL) return XAER_INVALID;

pCsemArray = (CsemArray *) handle;

memset(szName, 0, sizeof(szName));

for( i = 1; ; i++)

{

    if(API_ReadField(str1, i, szTmp, sizeof(szTmp), ',') <= 0)

    {

        break;

    }

    if(API_ReadField(szTmp, 1, szParamName, sizeof(szParamName), '=') <= 0)

    {
```

```
        return XAER_INVALID;

    }

    if(API_ReadField(szTmp, 2, szParamValue, sizeof(szParamValue), '=') <= 0)

    {

        return XAER_INVALID;

    }

    if(stricmp(szParamName, "Size") == 0)

    {

        nSize = atoi(szParamValue);

    }

    else if(stricmp(szParamName, "MaxCount") == 0)

    {

        nMaxCount = atoi(szParamValue);

    }

    else if(stricmp(szParamName, "InitialCount") == 0)

    {

        nInitialCount = atoi(szParamValue);

    }

    else if(stricmp(szParamName, "Name") == 0)

    {

        strncpy(szName, szParamValue, sizeof(szName));

    }

}
```

```
    if(strlen(szName) == 0)

    {

        pszName = NULL;

    }

    else

    {

        pszName = szName;

    }

    nRet = pCSemArray->Open(nSize, nInitialCount, nMaxCount, pszName);

    if(nRet != 0)

    {

        return nRet;

    }

    return(XA_OK);

}

/*****

/*          SemArray_close()          */

*****/

/*****
```

```
int KCBPCDECL SemArray_close(void *handle, char *str1, int i0, long i1)

{

    int nRet;

    CSemArray *pCSemArray;

    if(handle == NULL) return XAER_INVAL;

    pCSemArray = (CSemArray *) handle;

    nRet = pCSemArray->Close();

    if(nRet != 0)

    {

        return nRet;

    }

    return(XA_OK);

}

/*****

/*      SemArray_start()      */

/*****

int KCBPCDECL SemArray_start(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}
```

```

/*****/

/*      SemArray_end()      */

/*****/

int KCBPCDECL SemArray_end(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****/

/*      SemArray_rollback()      */

/*****/

int KCBPCDECL SemArray_rollback(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****/

/*      SemArray_prepare()      */

/*****/
```



```
int KCBPCDECL SemArray_prepare(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      SemArray_commit()      */

/*****

int KCBPCDECL SemArray_commit(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****

/*      SemArray_recover()      */

/*****

int KCBPCDECL SemArray_recover(void *handle, XID *str1, long i0, int i1, long i2)

{

    return(XA_OK);

}
```

```

/*****/

/*      SemArray_forget()      */

/*****/

int KCBPCDECL SemArray_forget(void *handle, XID *str1, int i0, long i1)

{

    return(XA_OK);

}

/*****/

/*      SemArray_complete()      */

/*****/

int KCBPCDECL SemArray_complete(void *handle, int *i0, int *i1, int i2, long i3)

{

    return(XA_OK);

}

/*****/

/*      SemArray_setopt()      */

/*****/

```

```
int KCBPCDECL SemArray_setopt(void *handle, char *option, char *str)

{

    return(XA_OK);

}

/*****

/*      SemArray_read()      */

/*****

int KCBPCDECL SemArray_read(void *handle, char **str, int *len)

{

/*

    int nRet;

    CSemArray *pCSemArray;

    if(handle == NULL) return XAER_INVALID;

    pCSemArray = (CSemArray *) handle;

    nRet = pCSemArray->Receive(*str, *len);

    if(nRet != 0)

    {

        return nRet;

    }

*/

    return(XA_OK);

}
```

```

/*****
/

/*      SemArray_write()      */

/*****

int KCBPCDECL SemArray_write(void *handle, char *str, int len)

{

/*

    int nRet;

    CSemArray *pCSemArray;

    if(handle == NULL) return XAER_INVAL;

    pCSemArray = (CSemArray *) handle;

    nRet = pCSemArray->Send(str, len);

    if(nRet != 0)

    {

        return nRet;

    }

*/

    return(XA_OK);

}

```

## 7. 编程工具

KCBP 提供了 `kcbpcp`、`kcbpadd`、`debuglbm`、`lbmtest`、`kcbptest` 等开发工具，帮助程序员完成测试、调试、性能优化等工作。下面我们分别介绍各种编程工具的使用方法。

### 7.1. 程序调用工具 KCBPCP

`kcbpcp` 是 KCBP 的管理工具，同时，也具有程序调用功能。

`kcbpcp` 是 KCBP command line processor 的简写，顾名思义，这个命令处理器就是处理各种命令的，在它处理的命令当中，有一个命令是 `execute`，这是用来调 LBM 程序的，有了这个工具，程序员就可以不需要编写客户端程序，直接输入参数给 LBM、调用 LBM，打印 LBM 返回的结果。此外，这个命令也可以自动重复调用，通过它可以完成压力测试。

本节描述的内容是 `kcbpcp` 的一部分功能，更详细的信息，请参见《KCBP 用户手册》第 7.3 节。

#### 7.1.1. 单次调用 LBM

这个命令用法如下：

1. 运行 `kcbpcp`
2. 连接到 KCBP Server，使用命令“`connect to kcbp1 user XXX using 888888`”，其中 XXX 是用户名称，888888 是口令。这个命令需要在 Server 端定义用户 XXX，并且属于 `manage` 组。
3. 输入 `execute`
4. `kcbpcp` 提示输入 LBM 名称，这时输入要调用的 LBM 名称
5. `kcbpcp` 提示输入 LBM 参数，参数格式 `name1:value1,name2:value2,...`。  
如“`stkcode:600446,price:22.10,volume:100,flag:1`”，其中 `srkcode` 是变

量名称，600446 是变量值。

6. 输完参数后回车，kcbpcp 就向 KCBP Server 发起请求，调用 LBM，然后接受 LBM 返回的结果，如果结果是二维表，kcbpcp 就逐行显示二维表的内容，如果结果不是二维表，kcbpcp 就显示出整个 buffer。显示完结果，kcbpcp 会显示这次调用的时间开销。
7. kcbpcp 提示输入另一个 LBM 名称，这时可以输入 quit 返回。

### 7.1.2. 重复调用 LBM

命令语法如下：

```
EXECUTE [lbmname [WITH parameter-list] [AT nodename] [USER userid  
USING password] [LOOP number]]
```

例如：

```
execute buy with custid:123,stkcode:600446,price:22.1,volume:1000,flag:1  
at kcbp1 user XXXX using xxxxxx loop 10
```

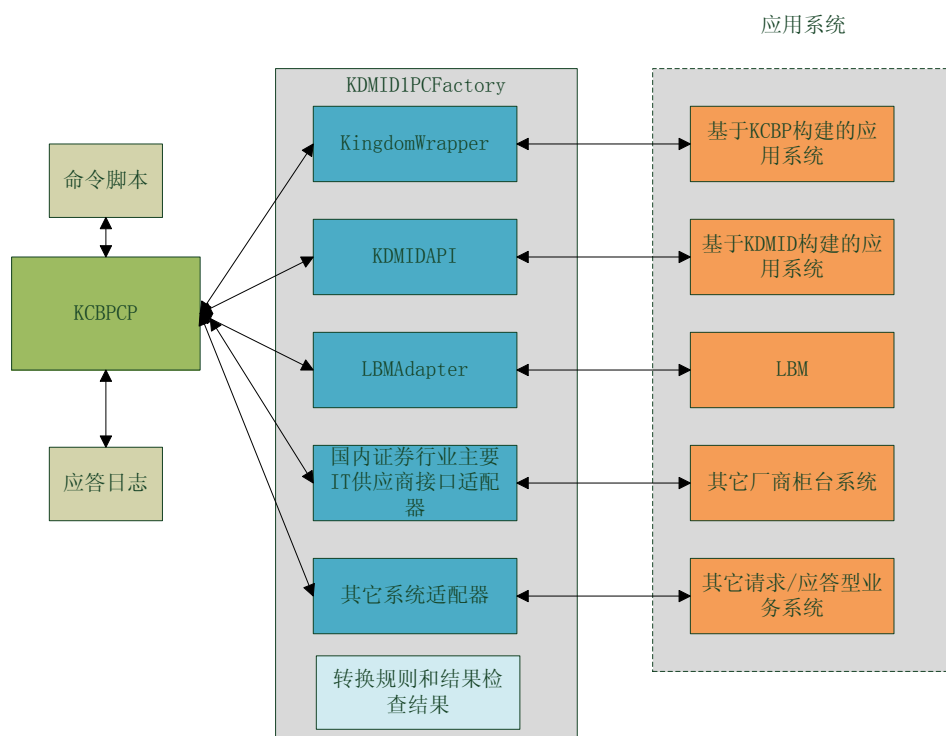
这时，调用 10 次 buy。

如果运行多个 kcbpcp，每个都调用一定数量的 lbm，就能实现压力测试。

### 7.1.3. 调试 LBM

这项功能可代替原来由 debuglbm 提供的功能。

kcbpcp，设置 kcbpxa 参数之后，再执行 execute 命令，将通过调用 KCBPCLI->KDMID1PCFactory->LBMAAdapter，加载和运行 LBM，供调试 LBM 用。当 LBM 程序运行到 DebugBreak 语句时，将进入调试状态。kcbpcp 加载和调试 lbm 原理图如下：



举例如下：

```
kcbpcp
```

```
set kcbpxa name=trade;protocol=KCBPXA;open=
rdc-duyw,master,jz30user,jPI4ybkZtf0Nm7rjKLM19w==,; close=;serial=
all_operation;option=

execute buy with custid:123,stkcode:600446,price:22.1,volume:1000,flag:1
at kcbp1 user XXXX using xxxxxx loop 10

quit
```

## 7.2. LBM 调试工具 DEBUGLBM

在 KCBPV3.0 版，DebugLBM 功能已经转移到 kcbpcp 中。以下内容适合 KCBP2.0 版本。

由于 LBM 程序是以动态库形式存在的，并且 LBM 要求 KCBP 作为特定的运行环境，因此，对它的调试方面需要一些技巧，下面的内容就是

对这些技巧的描述。

为了方便 LBM 程序的查错、调试，KCBP 提供了调试工具 `debuglbm`。`Debuglbm` 为 LBM 提供一个脱离 KCBP Server 的独立运行环境，它加载、执行 LBM，接收键盘输入的参数，并将参数传递给 LBM，最后将 LBM 返回的结果输出到屏幕。这个工具解除了 LBM 对 KCBP Server 运行环境的依赖，有了它，程序员再调试 LBM 时就不再需要运行起一整套环境，这就使调试 LBM 变成了一件简单的事。由于 `lbmapi` 能够支持 CICS，因此，程序员通过 `debuglbm` 也能调试 CICS 上的服务程序，提高 CICS 程序开发、调试效率。

`Debuglbm` 除了为 LBM 提供运行环境、输入、输出通道之外，还可以和调试器（如 `gdb`、`vc` 等）、LBM 交互作用，单步调试。单步调试需要断点触发机制，KCBP LBM API 中提供了一个 `KCBP_DebugBreak` 函数用来触发断点。当 LBM 程序调用 `KCBP_DebugBreak` 时，将产生调试中断，这个中断被调试器捕捉到，LBM 程序就停留在断点上，接着就可以在调试器上进行单步调试。

下面我们分别说明 `debuglbm` 在 Windows 和 Linux 上的用法。

### 7.2.1. 使用 `DEBUGLBM` 直接调用 LBM

`Debuglbm` 为 LBM 提供运行环境，提供输入，打印输出。

步骤如下：

1. 在 KCBP 的 BIN 目录下运行 `debuglbm`，`debuglbm` 启动时会读取 KCBP Server 的配置，建立起 LBM 的运行环境。
2. `debuglbm` 提示“`input lbm file full path =>`”，这时输入 `lbm` 所在动态库的全路径。
3. `debuglbm` 提示“`input lbm export function =>`”，这时输入 `lbm` 的入口名称。
4. `debuglbm` 提示“`input function argument =>`”，这时输入参数列表，参数格式 `name1:value1,name2:value2,...`。如“`stkcode:600446,price:22.10,volume:100,flag:1`”，其中 `srkcode` 是变量名称，`600446` 是变量值。



5. Debuglbn 执行 lbn, 将结果输出到屏幕上。
6. 输入 quit 退出

这时, Debuglbn 和 kcbpcp 比较如下:

1. kcbpcp 调用 LBM 时输入 LBM 名称, debuglbn 输入 LBM 路径和入口。
2. 二者输入参数格式相同
3. kcbpcp 要通过 KCBP Server 调用 LBM, 而 debuglbn 则不通过 KCBP Server, 它自己直接调用 LBM。
4. debuglbn 直接显示 lbn 返回的结果, 格式是 KCBP 通讯协议, 即裸格式; kcbpcp 输出二维表结果经过还原后逐行输出, kcbpcp 输出的非 2 维表格式与 debuglbn 相同, 也是裸格式。

### 7.2.2. WINDOWS 上使用 VC 和 DEBUGLBN 调试 LBN

Debuglbn 和 VC、LBN 配合进行单步调试, 步骤如下:

1. 首先需要在需要调试的 LBN 中适当的位置插入一行 KCBP\_DebugBreak, 设置断点, 编译出 Debug 版的 DLL。
2. 其次在 KCBP 的 BIN 目录下运行 debuglbn, debuglbn 启动时会读取 KCBP Server 的配置, 建立起 LBN 的运行环境。
3. 再次调试 debuglbn, 用 msdev -p debuglbn 进程号。
4. debuglbn 提示 "input lbn file full path =>", 这时输入 lbn 所在动态库的全路径。
5. debuglbn 提示 "input lbn export function =>", 这时输入 lbn 的入口名称。
6. debuglbn 提示 "input function argument =>", 这时输入参数列表, 参数格式 name1:value1,name2:value2,...。如 "stkcode:600446,price:22.10,volume:100,flag:1", 其中 stkcode 是变量名称, 600446 是变量值。
7. Debuglbn 执行 lbn, VC 调试器停留在 DebugBreak 代码行的反汇编代码处, 然后单步执行, 直到源代码出现, 然后关闭反汇编, 进

行单步源代码调试。

8. 结果输出到屏幕上。
9. 输入 quit 退出

### 7.2.3. LINUX 上使用 GDB+DEBUGLBM 调试 LBM

debuglbm 和 gdb、LBM 配合进行单步调试，步骤如下：

1. 首先需要在需要调试的 LBM 中适当的位置插入一行 KCBP\_DebugBreak，设置断点，编译出 Debug 版的动态库。**这里要特别提醒注意的是生产版本千万不要使用 KCBP\_DebugBreak，否则将造成灾难性的后果—kcbpas 进程被挂起，系统停止响应。**
2. 其次在 KCBP 的 bin 目录下运行 gdb debuglbm，然后输入 run 命令启动 debuglbm。debuglbm 启动时会读取 KCBP Server 的配置，建立起 LBM 的运行环境。
3. debuglbm 提示“input lbm file full path =>”，这时输入 lbm 所在动态库的全路径。
4. debuglbm 提示“input lbm export function =>”，这时输入 lbm 的入口名称。
5. debuglbm 提示“input function argument =>”，这时输入参数列表，参数格式 name1:value1,name2:value2,...。如“stkcode:600446,price:22.10,volume:100,flag:1”，其中 stkcode 是变量名称，600446 是变量值。
6. Debuglbm 执行 lbm，gdb 调试器停留在 DebugBreak 代码行，进行单步调试。
7. 结果输出到屏幕上。
8. 输入 quit 退出

## 7.2.4. DEBUGLBM 的局限性

目前, debuglbm 不支持代理类型程序的调试, 也不支持发布订阅程序的调试。如果你的程序是这类程序, 请直接调试 kcbp 进程或 kcbpas 进程, 如果是多进程方式运行 KCBP, 为了便于调试, 注意设置 kcbpas 初始化进程数目为 1 个。

## 7.3. LBM 提交工具 KCBPADD

kcbpadd 是一个提交和更新 LBM 程序的工具, 主要用在 makefile 文件中, 通过调用 kcbpcp 命令来向 KCBP Server 提交和更新 LBM 配置信息。

向 KCBP Server 上增加、修改、删除 LBM 的方式有 2 种: 静态方法和动态方法。

静态方法就是用 KCBPSetup 设置 Program, 或直接编辑 KCBPSPD.xml 文件, 需要重新启动 KCBP, 配置才能生效。

动态方法就是通过 kcbpcp 命令, 动态向 KCBP 发布增加、修改、删除等命令, 操作内容动态生效。

kcbpadd 是建立 kcbpcp 的一个调用工具, 除了能调用 kcbpcp 完成 insert、update、delete 等命令外, 还对 LBM 定义信息设置了缺省值, 并能够自动从 LBM 源代码中收集 LBM 相关定义参数。

### 7.3.1. 命令参数解释

kcbpadd 命令解释如下:

Usage:

```
kcbpadd -name value -acm value -cache value -module value -node value -originalnode value -path value -priority value -rsl value -timeout value -type value -userexitnumber value -xa value -concurrency value
```

Option explain:

-file, LBM source file, if specify it, kcbpadd will search source for module name, if more than one entry in a source file, the last will take affect; without default

- workdir, kcbpadd's work directory; without default
- update, Whether update LBM attribute, maybe enable or disable; default is disable
- name, LBM name; without default
- acm, access controll method, maybe public or private; default is public
- cache, LBM cache flag, maybe yes or no; default is no
- module, LBM entry function name, prefix with LBMEXPORTS; without default
- node, LBM transfer or deputy node identifier; default is 0
- originalnode, LBM transfer or deputy original node identifier; default is 0
- path, LBM dll or .so file path; without default
- priority, LBM priority, form 1-127; default is 1
- rsl, LBM resource security level, from 1-64; default is 1
- timeout, LBM maximum process timeout; default is 60
- type, LBM type, maybe biz, transfer, deputy; default is biz
- userexitnumber, LBM userexit function name list; default is 0
- xa, LBM related xa; default is 0
- concurrency, LBM concurrency, -1 no limit; default is -1

Example:

```
kcbpadd -name L9999999 -path kcbplbm.dll -module L9999999
```

```
kcbpadd -file L9999999.sqx -name L9999999 -path kcbplbm.dll -acm public -cache no -type biz
```

### 7.3.2. MAKEFILE 中 KCBPADD 使用举例

下面这个 makefile 是一个 LINUX 平台的编译 LBM 的通用 makefile, 它将当前目录下所有 LBM 程序 (\*.sqc 程序) 编译、连接到 libkcbplbm.so 动态库中, 并将 LBM 的相关信息增加 (或更新) 到 KCBP 中。makefile 使用的数据库是 DB2, 如果需要使用其它数据库, 只需要 makefile 的数

数据库连接命令。

```
#makefile for kcbplbm, Mr. Yuwei. Du , 20020719

PLATFORM = LINUX

CC = gcc

LBMTYPE = LBM

LBMAPI = lbmapl

KCBPPATH=/home/cts/kcbp

KCBPADD = $(KCBPPATH)/bin/kcbpadd

KCBPCLI = $(KCBPPATH)/bin/kcbpcp

#===== LINUX SPECIFIC OPTIONS =====

DB2PATH = /usr/IBMdb2/V7.1

CFLAGS = -c -Wall -g -fPIC -I. -I$(DB2PATH)/include -I../include/ -I$(KCBPPATH)/include

LIBS = -Wl,-rpath,. -Wl,-rpath,$(DB2PATH)/lib -L$(DB2PATH)/lib -L. -L$(KCBPPATH)/lib -ldb2
-I$(LBMAPI)

LINK = $(CC) -D$(PLATFORM) -fpic $(LDFLAGS)

PLATFORM_LIB_LINK_OPTIONS = -L/usr/lib -L/usr/local/lib

EXTRA_LINK_OPTIONS = -lc

SHLIBSUFFIX = .so

SLFLAGS = -shared -Wl,-soname,libkcbplbm.so

LINKSL = $(CC)

DB=sample

UID=

PWD=
```

ERASE= rm -f

SQC:= \$(wildcard \*.sqc)

SOURCE:= \$(patsubst %.sqc, %.c, \$(SQC))

BIND:= \$(patsubst %.sqc, %.bnd, \$(SQC))

OBJS:= \$(patsubst %.c, %.o, \$(SOURCE))

##\*\*\*\*\*

%.o : %.c

\$(CC) \$(CFLAGS) \$< -o \$@

\$(KCBPADD) -file \$< -path \$(KCBPPATH)/samples/server/lbm/libkcbplbm.so -type biz -timeout  
60

%.c : %.sqc

#db2 connect to \$(DB) user \$(UID) using \$(PWD)

db2 connect to \$(DB)

db2 prep \$\*.sqc bindfile

db2 bind \$\*.bnd

db2 connect reset

all: libkcbplbm

libkcbplbm : libkcbplbm.so.1.0.0

libkcbplbm.so.1.0.0 : \$(OBJS)

```
$(KCBPCLI) CONNECT TO kcbp1 user test using test

$(LINKSL) $(SLFLAGS) $(EXTRA_SLFLAGS) $(LIBS) $^ -o $@

$(ERASE) libkcbplbm.so

$(ERASE) libkcbplbm.so.1

ln -s -f $@ libkcbplbm.so.1

ln -s -f $@ libkcbplbm.so

#*****

cleanall : clean cleangen cleanso

clean :

$(ERASE) $(BIND)

cleangen :

$(ERASE) $(OBJS)

cleanso :

$(ERASE) libkcbp*.so libkcbp*.so.*
```

## 7. 4. KCBP 系统压力工具 KCBPTEST

KCBPTest 是金证公司开发的一个可扩展的、简洁易用、高效的通用并行压力测试工具。KCBPTest 的测试效果与原理与类似于业界负有盛名的 LoadRunner，但比 LoadRunner 更简单易用。KCBPTest 通过模拟大量用户实施并发负载及实时性能监测的方法来确认和查找问题，KCBPTest 能够对整个系统架构进行测试，找出其中写的不够优化的业务逻辑。通

过使用 KCBPTest，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。可以尽量避免难以预知的用户负载和复杂的网络、软硬件应用环境下导致用户响应速度过慢，系统崩溃等问题。

KCBPTest 运行在 Windows 系统上，可以同时进行多个工作任务压力测试，可同时测试多个应用系统。它具有开放的结构，系统由测试主程序及适配器构成。主程序启动、停止压力测试任务，通过图形界面显示每个压力任务的运行情况。适配器是动态库，用来完成与后端系统的通讯操作。对不同的后端系统，都需要一个与之配套的适配器，适配器采用 KDAPI 规范编写。KDAPI 规范由一组函数组成，这组函数完成建立连接、发送数据、接收数据、关闭连接等操作。KDAPI 此处不赘述，需要更进一步信息请联系研发中心。KCBPTest 的测试样本数据存放在文本文件中，每个压力任务可以按照自己的需求设置测试样本数据。样本中支持宏定义。样本加载方式支持顺序、乱序。支持循环测试，可自定义循环次数。KCBPTest 目前主要应用于证券行业的各种交易系统性能测试。由于其本身的开放特性，也适合其他行业软件系统的性能和功能测试

#### 7.4.1. 程序文件说明

主程序：KCBPTest.exe

配置文件：KCBPTest.ini

相关组件库：

Chart.dll KCBPcli.dll KCBPcrypt.dll KCXPapi.dll libdes.dll libeay32.dll  
libidea.dll libidea1.dll libzip.dll libzlib.dll libzlib1.dll SSLeay32.dll



## 7.4.2. 配置文件说明

KCBPTest 的配置文件是 KCBPTest.ini, 用户可以通过编辑器编辑它, 配置节 COMMON 存放公共参数, TASKn 存放每个任务的配置参数, 目前最多可定义 10 任务, 各项参数含义具体说明如下:

配置节	配置项	说明
COMMON	ADAPTER	适配器动态库名称, 缺省为空串, 表示使用用 KCBP 协议进行压力测试; 如设置 KDMIDAPI.DLL 可以对 KDMID 做压力测试
	SERVER_NAME	KCBP 服务器名称, 本地有效
	IP	KCXP 服务器的 IP 地址
	PORT	KCXP 服务器的端口
	SSL	SSL 参数, 空表示不用 SSL, 格式如下:  状态, 根证书, 证书, 密码, 算法表, 加密传输, 主动发起握手, 如: YES, sscCA.pem, client.pem, client, ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH, YES, YES
	PROXY	代理服务器参数, URL 格式, 空表示不用代理服务器, 有效格式如:  <a href="https://192.168.40.65:80">https://192.168.40.65:80</a> socks4://192.168.40.65:1080 socks5://guest:password@192.168.40.65:1080。
	REQUESTQ	请求队列名
	ANSWERQ	应答队列名
	USER_NAME	用户名
	PASSWORD	明文口令
	TIMEOUT	请求超时时间, 秒
COMPRESS	通讯压缩方法, 0, 不压缩, 其它压缩, 算法在 KCXPPlugin.Dat 中定义	

	CRYPT	通讯加密方法, 0, 不加密, 其它加密, 算法在 KCXPPlugin.Dat 中定义
	NULLPARAMETER	系统保留
	STEP	每个压力线程每次读取指令数
	LOOPCOUNT	数据样本的循环调用次数
	SAMPLEACCESSMODE	样本供给方式, 1 顺序, 0 随机, 缺省是 1
	CONCURRENCE	压力线程建立连接时的最大并发数, 用来控制 connect 调用的并发个数, 解决它多并发线程同时连接服务端存在的 listen backuplog 个数不够问题
	CONFIRM	通讯确认方式, 1 采用确认方式收发报文; 0 采用非确认方式收发报文, 通讯效率较前面方式快 1 倍, 需要与 2009 年版本的 KCXP 配合。
DEFINE	\$MACRO	宏定义, 可以在测试样本文件中引用, KCBPTest 程序执行时对样本中的宏变量进行替换, 这项设计可以帮助减小样本文件的长度:  \$VAR1 =custprop:, orgid:0001 \$ VAR2 =custprop:1, orgid:0002
TASK1	TASK_NAME	任务名称
	THREAD_NUM	线程数
	INPUT_FILE	指令文件
	GET_RESULT	是否接收执行结果 0-不接收, 1-接收, 结果存放到 data 目录下
	INTERAL	每笔指令时间间隔,单位毫秒
	MAX_NUM	每个线程每秒最多发送指令数

说明:

- 如果参数配置在 COMMON 节，那么作用域是所有任务，如果配置在各 Task 任务节，作用域是该节。
- 如要增加任务 2 或更多任务，配置方式参考任务 1 的配置即可，只需将抬头修改为[TASK2]。

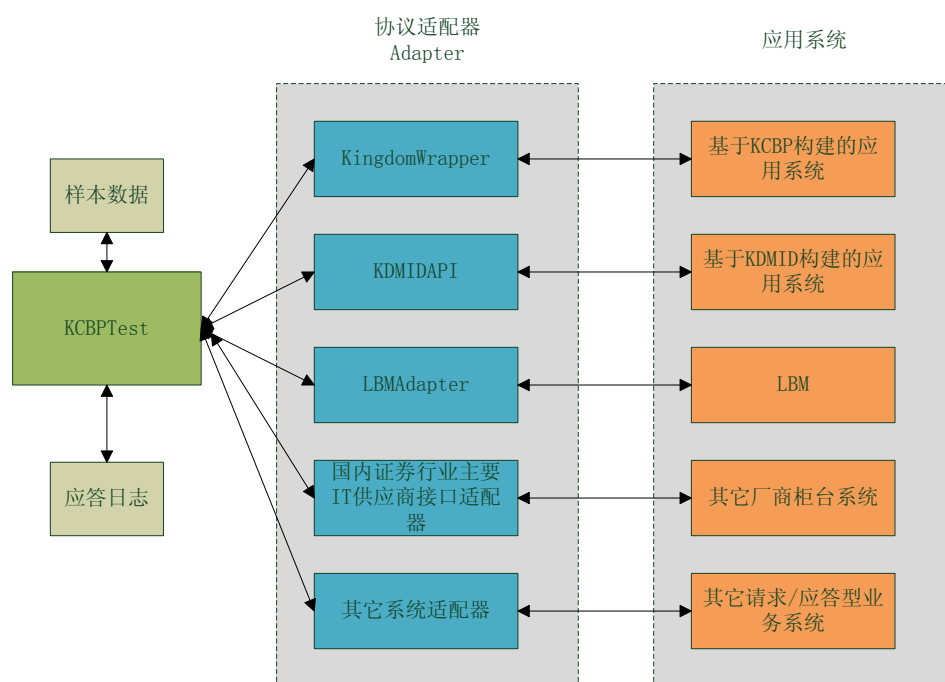
### 7.4.3. 指令文件格式

指令文件由一行或多行指令构成，每行存放一条指令，指令包括要调用的 LBM 名称及参数，LBM 名称何参数之间使用感叹号分隔，参数之间使用逗号分隔，参数名称和值之间使用冒号分隔，例如：

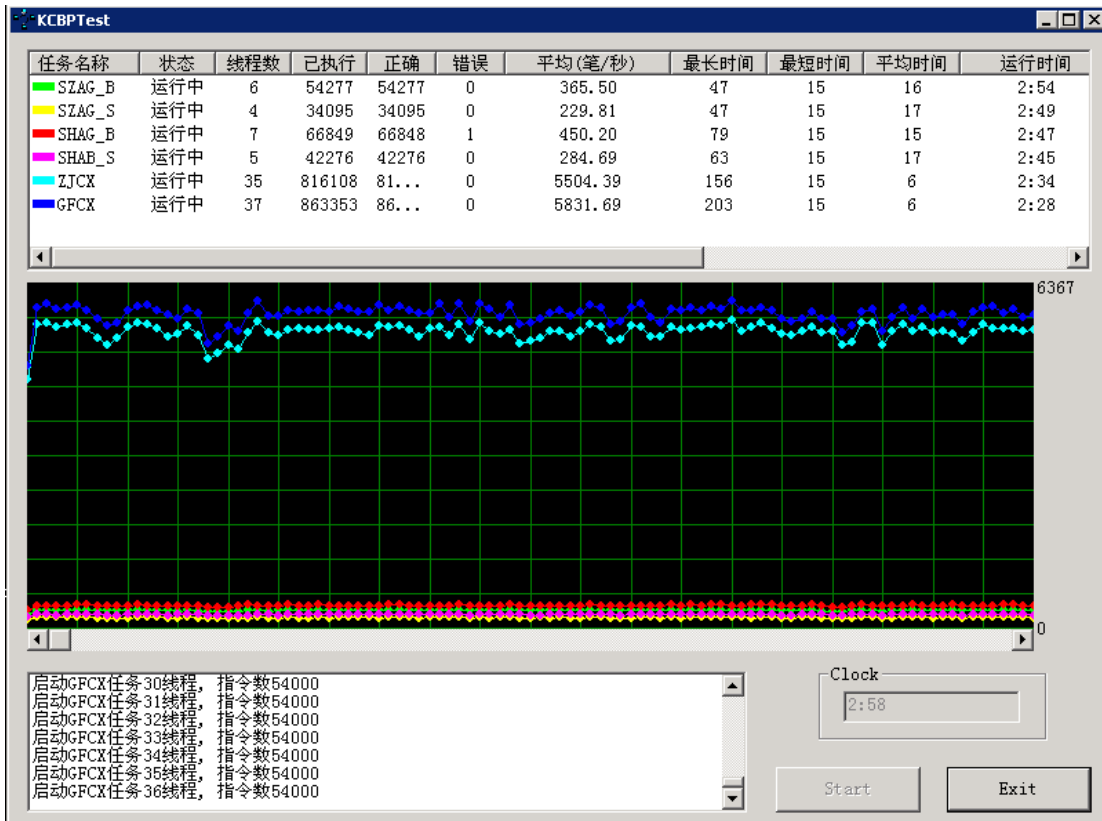
```
150101!USERTYPE:3,USERID:1,CUSTOMER:1001,MARKET:0,TRADE
_ACCT:C920001001,STK_INTL:900921,TRADE:0B,VOLUME:100,PRIC
E:0.41,BRANCH:0001,CHANNEL:0
```

### 7.4.4. 测试系统架构

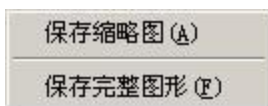
下图是 KCBPTest 测试系统关系图:



### 7.4.5. 运行界面



在这个界面按鼠标右键，出现如下菜单：

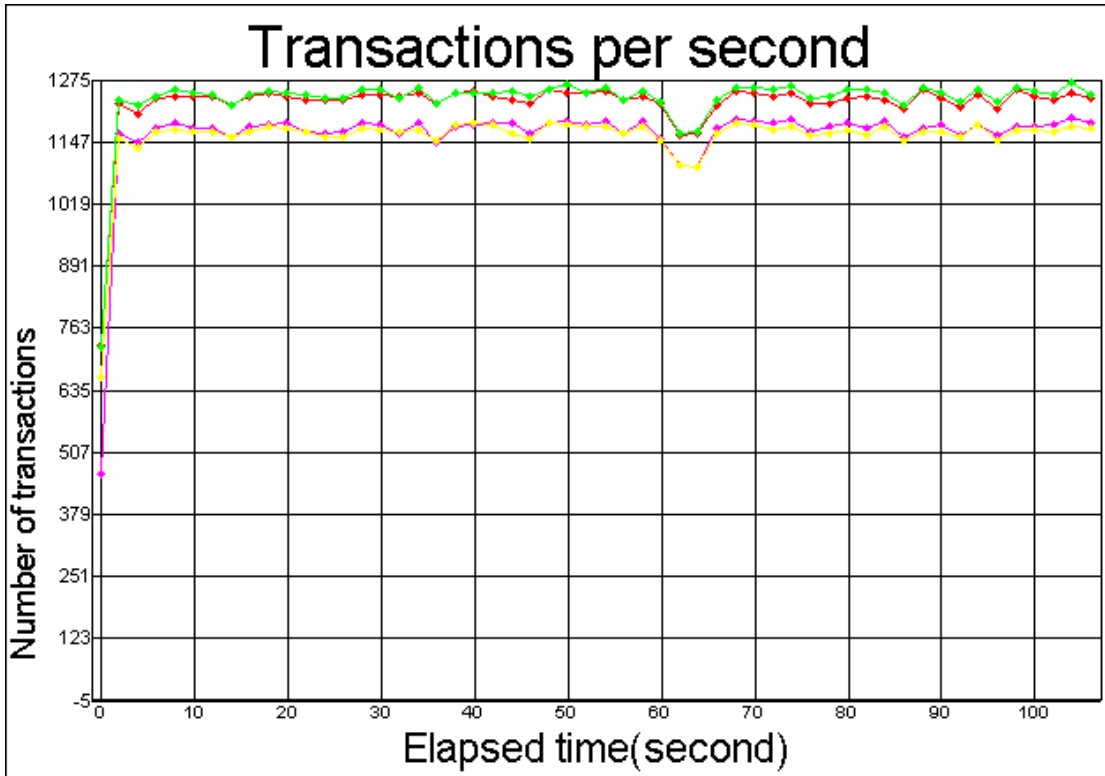


其中：

保存缩略图将按 X 轴 100 等分绘制 TPS 图形并保存；

保存完整图形将按 X 轴每秒一个数据点绘制 TPS 图形并保存。

下图是一个保存完整图形的例子，其中 X 轴每秒一个点：



KCBPTest 生成的 TPS 图

#### 7.4.6. 界面显示结果说明

列名	含义
任务名称	TASK 中定义的名称
状态	未执行、执行中、完成
线程数	任务的并发处理线程数
已执行	已经调用的 LBM 数目
正确	执行正确的 LBM 数目
错误	发生错误的 LBM 数目
平均（笔/秒）	每秒执行的 LBM 数目
最长时间	单个 LBM 执行最长时间，单位毫秒

最短时间	单个 LBM 执行最短时间，单位毫秒
平均时间	LBM 执行的平均时间，单位毫秒
运行时间	任务运行时间,格式 分：秒
失败	调用失败次数
每笔延时	每笔调用延时时间，来源于任务配置参数
每秒笔数	每秒调用的笔数，来源于任务配置参数
输入文件	指令文件，来源于任务配置参数
输出结果	是否输出结果
总指令数	指令文件中的总指令数

#### 7.4.7. 适配器接口函数列表

```

#ifndef_KDMIDAPI_H

#define_KDMIDAPI_H

#ifndef_KDMIDAPI_EXPORTS

#define_KDMIDAPI_API__declspec(dllexport)

#else

#define_KDMIDAPI_API__declspec(dllimport)

#endif

#define_KDMIDAPI_STDCALL__stdcall /* ensurestdcall calling convention on NT*/

typedef void*KDMIDAPIHANDLE;

#endif__cplusplus

```

```
extern "C" {

#ifdef

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_Init(KDMIDAPIHANDLE *hHandle);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_Exit(KDMIDAPIHANDLE hHandle);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_SetOption(KDMIDAPIHANDLE hHandle, int nIndex, char *pValue);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_GetOption(KDMIDAPIHANDLE hHandle, int nIndex, char *pValue, int nLen);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_Open(KDMIDAPIHANDLE hHandle, char *szIpAddress, char *szPort);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_Close(KDMIDAPIHANDLE hHandle);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_SendRequest(KDMIDAPIHANDLE hHandle, char *szReqName, char *szBuffer);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_GetReply(KDMIDAPIHANDLE hHandle);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_TypeOfResult(KDMIDAPIHANDLE hHandle);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_Read(KDMIDAPIHANDLE hHandle, int nIndex, char *pOut, int nLen);

KDMIDAPI_API int KDMIDAPISTDCALL KDMIDAPI_GetError(KDMIDAPIHANDLE hHandle, char *szError, int nLen);

#ifdef __cplusplus

}

#endif

#endif
```

#### 7.4.8. 应用案例：使用 KCBPTEST 对金证交易系统 V3.2 进行查询资金及股份的压力测试

对金证柜台 V3.2，适配器采用 KDMIDAPI.DLL

本次测试 KCBPTest 配置文件设置如下：

```
[COMMON] //通用配置项，对所有任务生效
USER_NAME = 9999 //测试柜员代码
```

```
PASSWORD= 888888 //测试柜员密码
TIMEOUT = 15 //超时时间，单位秒
CRYPT = 1 //通讯加密算法

[TASK1] //任务 1
TASK_NAME = 资金查询 //该任务的显示名称
ADAPTER = KDMIDAPI.DLL //适配器
IP = 192.168.80.88 //MID 服务器 1 的 IP 地址
PORT = 28946 //MID 服务器 1 的端口
THREAD_NUM = 75 //该任务的线程数
INPUT_FILE = .. \data\queryfund.txt //查询资金数据样本文件
GET_RESULT = 0 //是否记录执行结果 0-否，1-是
INTERAL = 0 //每笔指令时间间隔,单位：秒
MAX_NUM = 0 //每线程每秒最多发送的指令数
LOOPCOUNT = 1 //该任务数据样本的循环调用次数
SAMPLEACCESSMODE = 1 //样本供给方式，1 顺序，0 随机

[TASK2] //任务 2
TASK_NAME = 股份查询 //该任务的显示名称
ADAPTER = KDMIDAPI.DLL //适配器
IP = 192.168.80.90 //MID 服务器 2 的 IP 地址
PORT = 28946 //MID 服务器 2 的端口
THREAD_NUM = 75 //该任务的线程数
INPUT_FILE = .. \data\querystock.txt //查询股份数据样本文件
GET_RESULT = 0 //是否记录执行结果 0-否，1-是
INTERAL = 0 //每笔指令时间间隔,单位：秒
MAX_NUM = 0 //每线程每秒最多发送的指令数
LOOPCOUNT = 1 //该任务数据样本的循环调用次数
SAMPLEACCESSMODE = 1 //样本供给方式，1 顺序，0 随机
```

注意，前面配置中，为了展示系统的并行处理能力，每个业务由一个单独的服务器处理。

查询资金数据样本 queryfund.txt 格式如下：

```
20102906!khbs1x:Z, khbs:1000000001
```



20102906!khbslx:Z, khbs:1000000002

20102906!khbslx:Z, khbs:1000000003

……, 此处省略若干行

其中, 20102906 是业务请求编号, khbslx:Z 表示帐号是资金帐号, khbs:1000000001 表示资金帐号是 1000000001。

查询股份数据样本 querystock.txt 格式如下:

20102907!khbslx:Z, khbs:1000000001, zqbslx: , zqbs: , gdms:2, cxlx:0

20102907!khbslx:Z, khbs:1000000002, zqbslx: , zqbs: , gdms:2, cxlx:

20102907!khbslx:Z, khbs:1000000003, zqbslx: , zqbs: , gdms:2, cxlx:0

……, 此处省略若干行

数据样本可以使用 SQL 语句从柜台数据库中获取。

配置好之后, 就可以进行压力测试。

如果需要测试其他业务, 可根据《金证传统外围统一接口规范》编写测试数据样本。

如果需要对非金证柜台做压力测试, 只需更换适配器, 然后按照目标系统接口规范编写测试数据样本, 此处不再详述。

## 7.5. LBM 压力测试工具 LBMTEST

在 KCBPV3.0 版, LBMTEST 功能已经转移到 KCBPTest 中, 在 KCBPTest 上需要做的工作是配置 Adapter=LBMAdapter.dll。

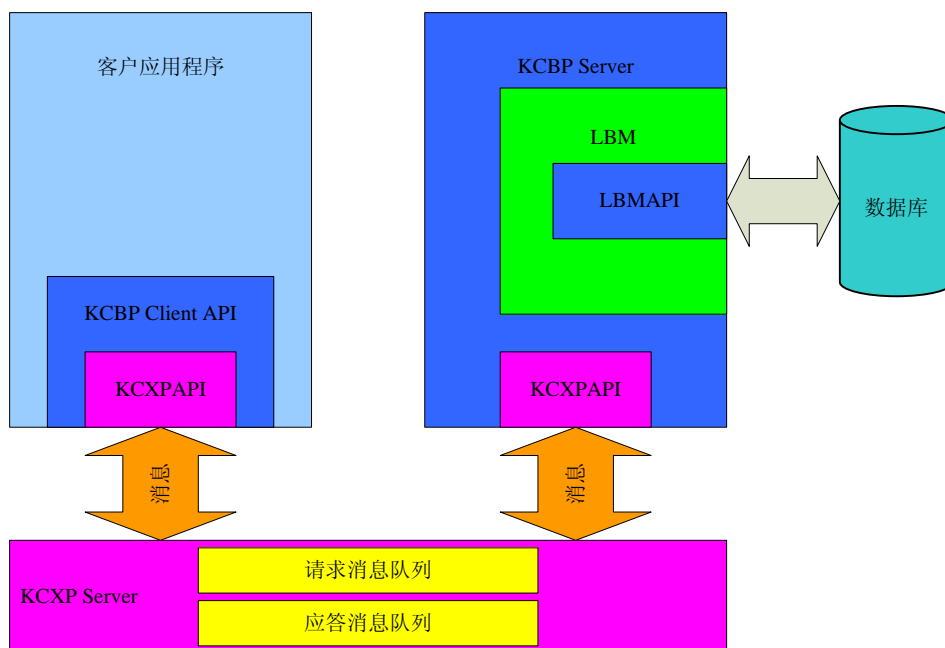
## 8. 部署方案

KCBP 和 KCXP 组合可以构建出实时、高稳定、可线形扩展、动态负载均衡的平台。

本部分描述的部署方案主要是总体性描述，一般不包含具体的配置信息，关于配置方面的详细说明请参考《KCBP 用户手册》。

### 8.1. KCBP 和 KCXP 的关系

在目前版本的 KCBP 中，通讯功能是由 KCXP 的队列技术完成的。KCXP 是金证公司开发的高性能消息中间件，它使用消息队列技术，和 KCBP 同样适合于对实时性要求高的企业应用系统。KCBP 运行成功的前提是 KCXP 已经正常启动。下图是非集群方式的 KCBP 和 KCXP 系统关系图。



从上图可见，KCXP 处于整个系统的底层，前端应用程序和 KCBP 都在上层，通过 KCXP 进行通讯。

编程时，程序员可以把 KCXP 和 KCBP 看作一个系统，KCXP 是 KCBP 的通讯子系统，KCXP 的特征被 KCBP 封装了，编程时不必在意 KCXP 通讯子系统的存在。实际部署和运行时，KCXP 是真实存在的，并且以独立系统的形式存在，KCXP 有自己运行程序、界面、管理工具。

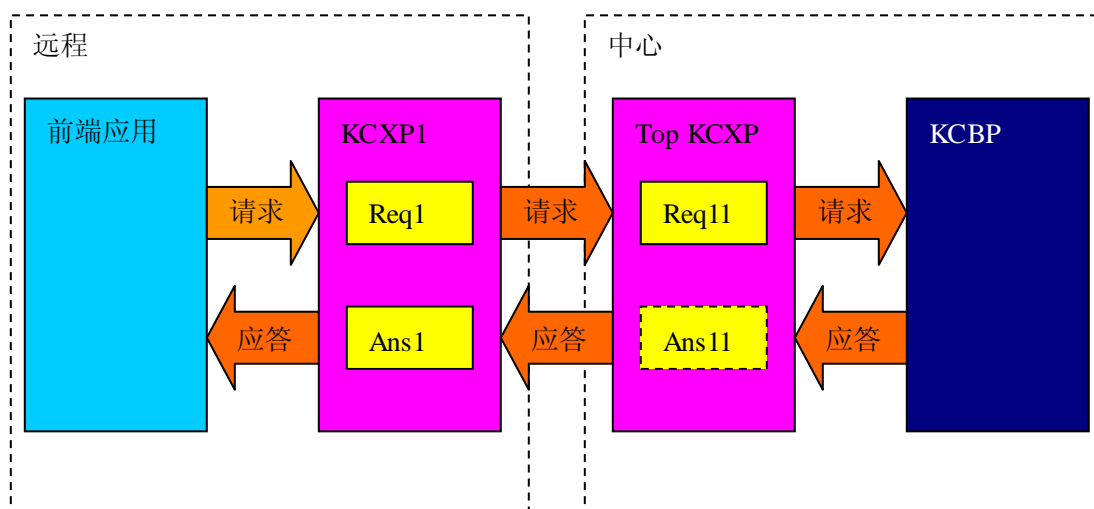
在 KCBP Client API 内部，所有的通讯操作，都是调用 KCXP API 完成的，KCBP Server 也通过 KCXP API 完成通讯操作的。系统中信息是通过消息形式传递的，消息是 KCBP 系统传输请求和应答数据的基本单位，KCBP 对数据进行了消息化处理，包括封装、编码、分组、压缩等操作。消息在整个系统中的处理流程描述如下：

1. 前端应用程序发起请求，调用 KCBP Client API；
2. KCBP Client API 调用 KCXP API 将请求转换成消息，传递到 KCXP Server 上的请求消息队列中存放；
3. KCBP Server 从 KCXP Server 上的请求消息队列中取得消息；
4. KCBP Server 根据消息的内容调用相应的 LBM（可访问数据库）
5. LBM 完成业务处理并返回结果给 KCBP Server；
6. KCBP Server 将结果以消息形式发送到 KCXP Server 上的应答消息队列中；
7. KCXP API 从 KCXP Server 上的应答消息队列中取得应答消息并将其返回给 KCBP Client；
8. KCBP Client 将消息转换成用户可识别的结果返回给前端应用程序。

## 8.2. KCXP 多级部署方式

### 8.2.1. 中心和远程两级 KCXP 部署

下图是一对一的 KCXP 两级部署示意图，由远程和中心两部分组成。



远程部分包括前端应用和远程 KCXP (KCXP1)，中心部分包括中心 KCXP (Top KCXP) 和 KCBP。这里要注意：中心的 Top KCXP 和远程的 KCXP 具有不同的节点编号，节点编号要求在集群中唯一，KCXP 的节点编号用来在 KCXP 集群中识别 KCXP 身份。其中，前端应用连接 KCXP1，KCBP 连接 TOP KCXP，KCXP1 和 Top KCXP 之间使用消息通道进行通讯。

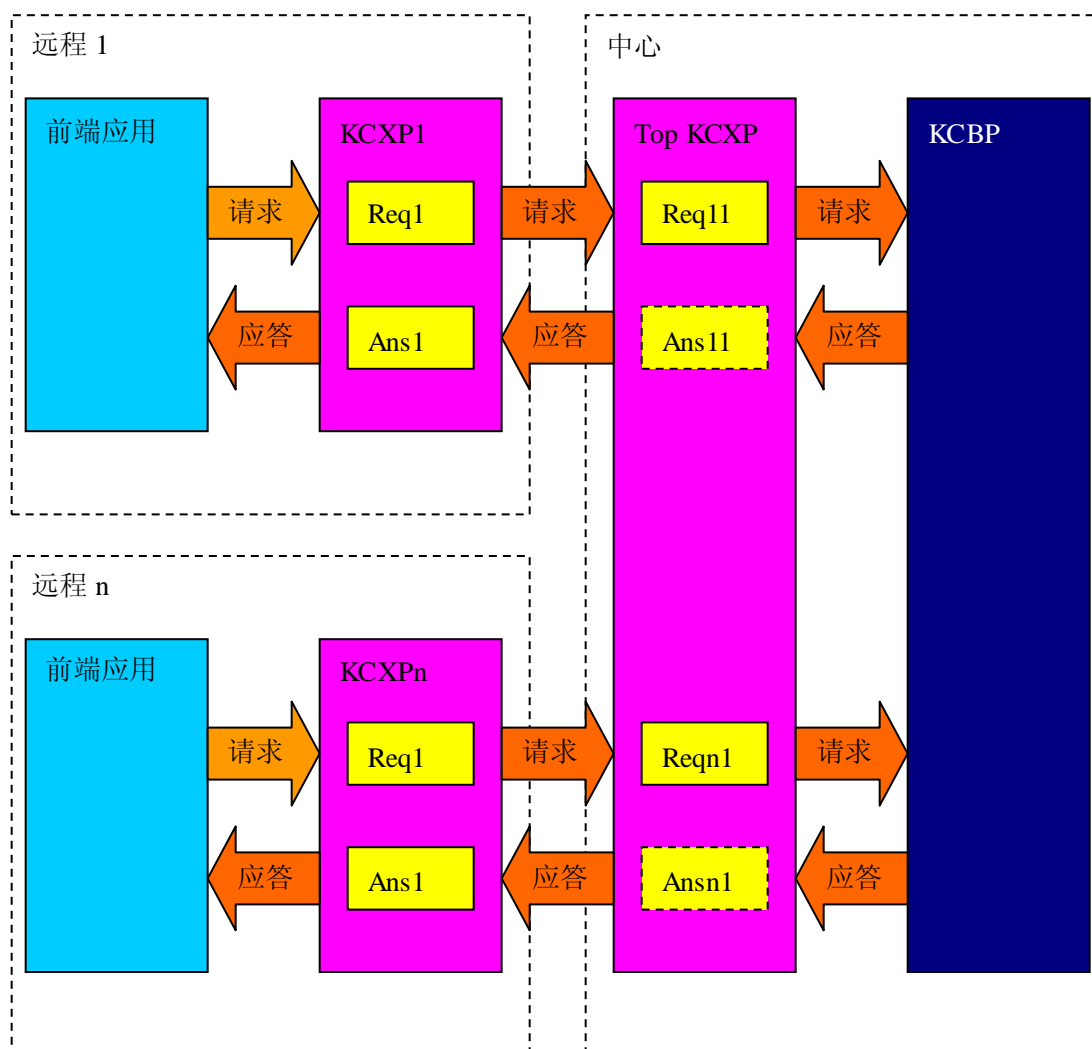
KCXP1 上面的请求队列 Req1 是一个远程队列，它对应 Top KCXP 上的本地队列 Req11，请求通过 Req1->Req11 传递给 KCBP。

Top KCXP 上的应答队列 Ans11 是一个远程队列，它对应 KCXP1 上的本地队列 Ans1，应答通过 Ans11->Ans1 传递给前端应用。

注意，Ans11 是虚线表示的。在 KCXP/Linux 版中，上述内容是恰当的，但在 KCBP/Win 版中有所不同：这时在 KCBP 上面需要配置 Ans11，但实际上不会用到它，因为 KCBP/Win 版在发送应答时采用了路由消息发送方式，就是根据请求消息中指定的应答 KCXP 节点和应答队列发送应答，所以实际上不会用到 Ans11。

## 8.2.2. 中心和 n 个远程两级 KCXP 部署

下图是一对多的 KCXP 两级部署示意图，由一个中心和 n 个远程组成。



远程部分可以有  $n$  ( $n \geq 1$ ) 个，每个远程部分都包括前端应用和远程 KCXPn，中心部分包括中心 KCXP (Top KCXP) 和 KCBP。其中，前端应用连接 KCXPn，KCBP 连接 TOP KCXP，KCXPn 和 Top KCXP 之间使用消息通道进行通讯。

KCXP1 上面的请求队列 Req1 是一个远程队列，它对应 Top KCXP 上的本地队列 Req11，请求通过 Req1->Req11 传递给 KCBP。Top KCXP 上的应答队列 Ans11 是一个远程队列，它对应 KCXP1 上的本地队列 Ans1，应答通过 Ans11->Ans1 传递给前端应用。

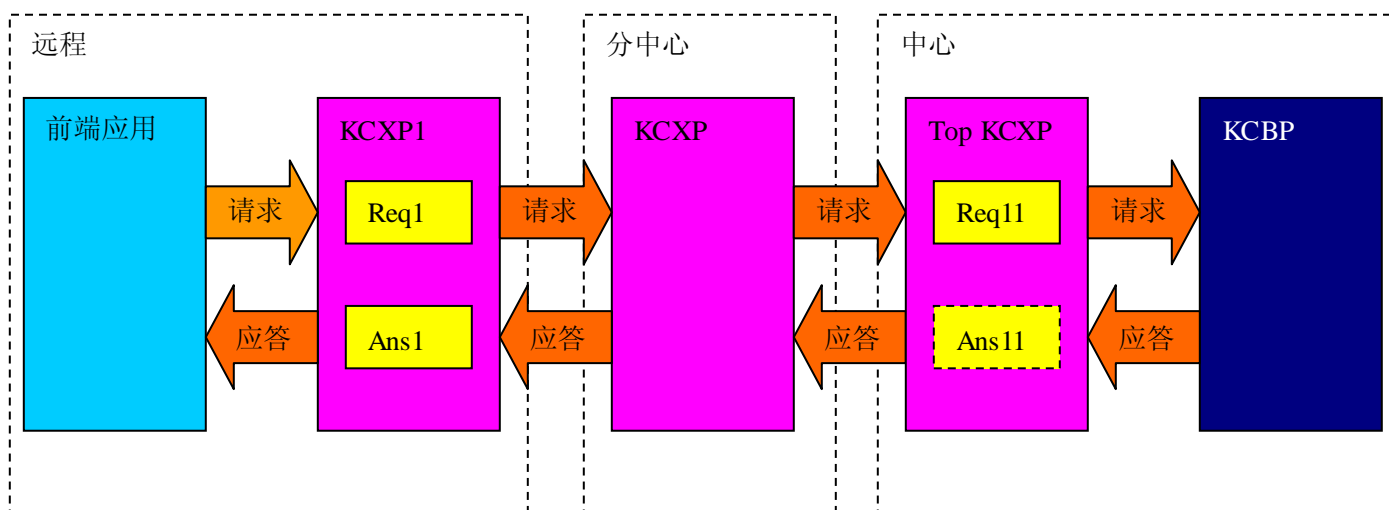
同理，KCXPn 上面的请求队列 Req1 是一个远程队列，它对应 Top KCXP

上的本地队列 Req<sub>n1</sub>，请求通过 Req<sub>1</sub>->Req<sub>n1</sub> 传递给 KCBP。Top KCXP 上的应答队列 Ans<sub>n1</sub> 是一个远程队列，它对应 KCXP<sub>n</sub> 上的本地队列 Ans<sub>1</sub>，应答通过 Ans<sub>n1</sub>->Ans<sub>1</sub> 传递给前端应用。

Ans<sub>n1</sub> 的注意事项同上节。

### 8.2.3. 中心、分中心、远程三级 KCXP 部署

下图是一对一的 KCXP 三级部署示意图，由中心、分中心和远程三部分组成。



远程部分包括前端应用和远程 KCXP (KCXP1)，分中心包括一个 KCXP、中心部分包括中心 KCXP (Top KCXP) 和 KCBP。其中，前端应用连接 KCXP1，KCBP 连接 TOP KCXP，KCXP1 通过分中心 KCXP 和 Top KCXP 通讯，分中心的 KCXP 为远程 KCXP 和中心 KCXP 提供消息路由功能。

KCXP1 上面的请求队列 Req<sub>1</sub> 是一个远程队列，它对应 Top KCXP 上的本地队列 Req<sub>11</sub>，请求通过远程 Req<sub>1</sub>->分中心->Req<sub>11</sub> 传递给 KCBP。

Top KCXP 上的应答队列 Ans<sub>11</sub> 是一个远程队列，它对应 KCXP1 上的本地队列 Ans<sub>1</sub>，应答通过 Ans<sub>11</sub>->分中心->Ans<sub>1</sub> 传递给前端应用。

Ans<sub>11</sub> 的注意事项同上节。

这种部署方式可以进一步扩展为多级分中心，原理和 3 级部署相同。

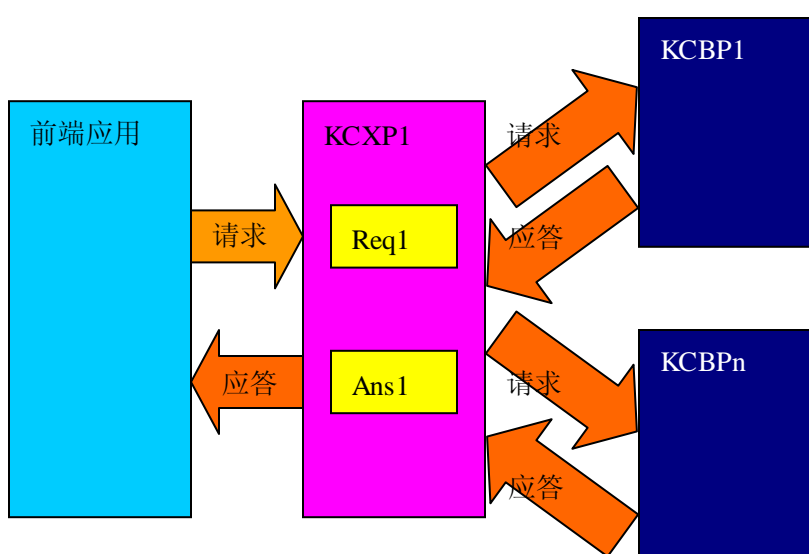
## 8.3. KCBP 动态负载均衡部署

本节中描述的 KCBP 部署一般是单级的，实际部署时可扩展为多级。

动态负载均衡的各 KCBP 节点完成相同的功能。

### 8.3.1. 单级单 KCBP+多 KCBP 动态负载均衡部署

示意图如下：



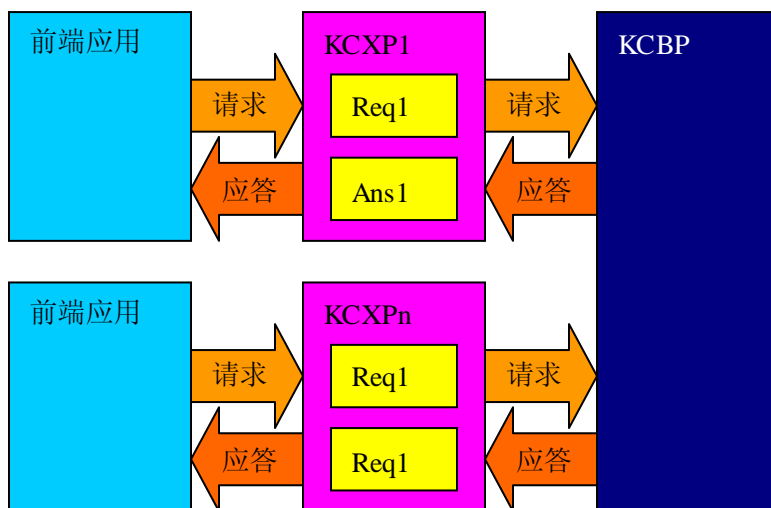
系统中存在多个功能相同的 KCBP 节点，这些节点连接同一个 KCXP 上，从同一个请求队列 Req1 中取请求，应答发送到 Ans1 上。

这种动态负载均衡方式称为 Multiple server single queue，简称 MSSQ。

MSSQ 的各 KCBP 节点部署在不同的机器上，提供相同的处理功能，各自的处理能力与硬件能力和 KCBP 软件设置相关。MSSQ 是一种典型的模糊动态负载均衡技术。

KCBP 中的请求队列是可扩展的，可有多个，每个 KCBP 可侦听多个请求队列。

### 8.3.2. 单级多 KCXP+单 KCBP 动态负载均衡部署

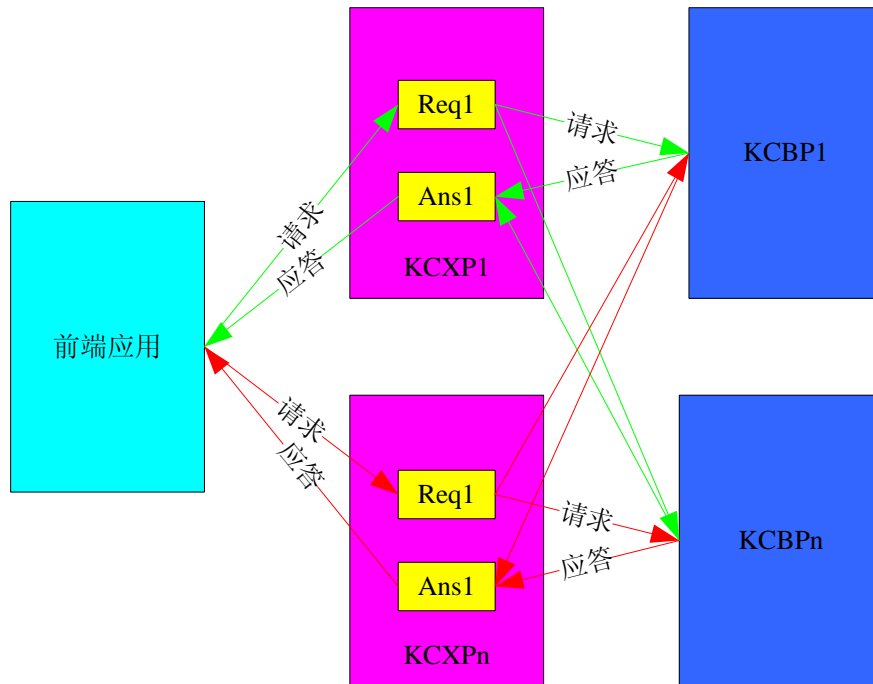


一个 KCBP 连接多个 KCXP。

这种部署方法中 KCXP 是并行的，可以解决 KCXP 的单点故障问题。



### 8.3.3. 单级多 KCXP+多 KCBP 动态负载均衡容错部署



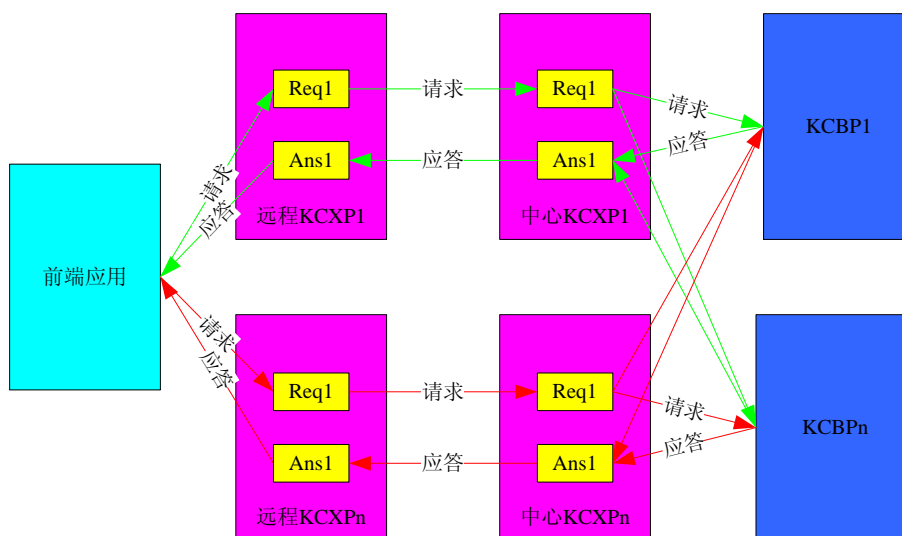
这种部署方法既可以容错，也可以动态负载均衡，总的来讲，这是一种理想的部署方法。

前端应用程序可以通过 `KCBPCLI_SetOptions` 的 `KCBP_OPTION_CONNECT` 选项设置多个连接参数（每个连接参数包括名称、IP、端口、队列等），这样，`KCBP Client API` 中的随机负载均衡算法会自动选择一个 `KCXP` 节点进行连接。如果前端应用有多个，和每个 `KCXP` 节点上的连接数概率均等，这样，即可容错、也可均衡。

多个 `KCXP` 节点之间可以互相备份，互相容错。在我们当前这种部署中，各 `KCXP` 是单独的，没有用到 `KCXP` 自身的集群负载均衡和容错机制。这里要提一点，`KCXP` 多节点之间也可配置成集群负载均衡和容错，有兴趣的读者，可以参阅 `KCXP` 的用户手册。

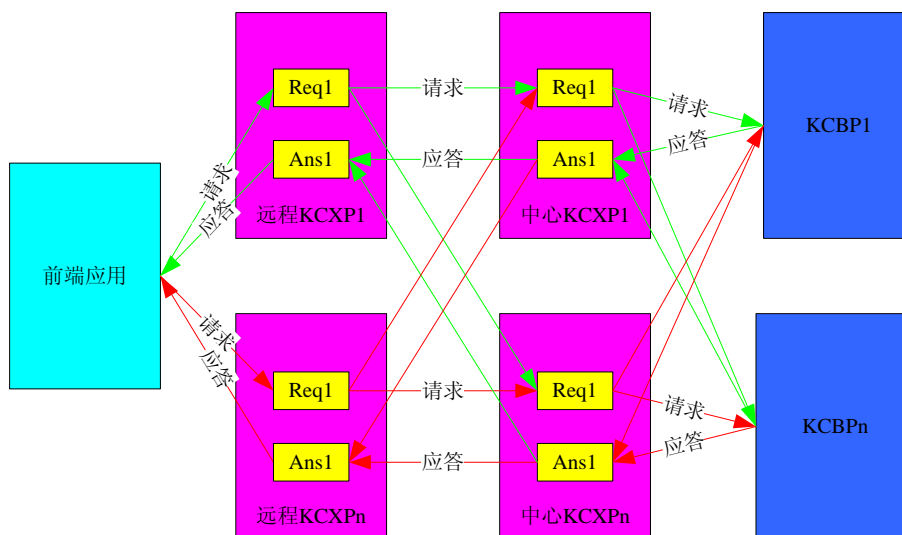
各 `KCBP` 节点是功能对等的，他们对外提供相同的服务。每个 `KCBP` 可以从多个 `KCXP` 并行获取请求，并行处理。

### 8.3.4. 多级多 KCXP+多 KCBP 动态负载均衡容错部署



这幅图与前面一节的图相比，多了一层远程 KCXP。远程 KCXP 和中心 KCXP 之间一对一连接。

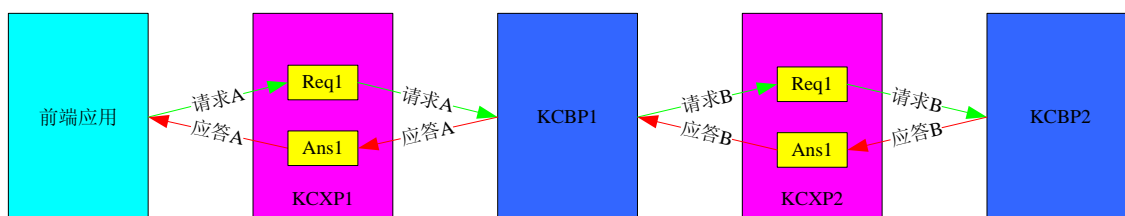
### 8.3.5. 多级 KCXP+多 KCBP 网状动态负载均衡容错部署



这幅图中，远程 KCXP 和中心 KCXP 之间采用交叉连接，容错能力更强，当 KCXP 节点多时，系统配置较复杂。

## 8.4. KCBP 转发部署

### 8.4.1. 一对一转发部署



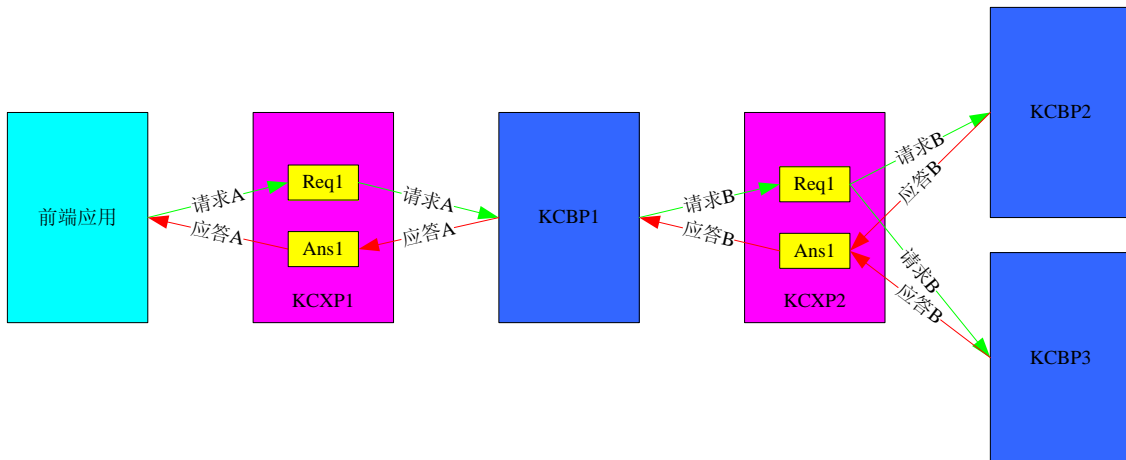
上图中，KCBP 节点 1 和 KCBP 节点 2 完成不同的功能。

转发的 2 个条件是：

- 客户端发起的业务 A，在 KCBP1 上调用 LBM 处理过程中，LBM 使用 KCBP\_CallProgram 或 KCBP\_CallProgramSys 调用 KCBP2 上的业务 B。如果使用 KCBP\_CallProgram，需要在 KCBP 的服务定义中一个代理类型的 LBM，它使用的 XA 名称对应 KCBP2。如果使用 KCBP\_CallProgramSys，当节点输入 0 时，处理流程与 KCBP\_CallProgram 相同；如果节点编号不是 0，则转发到同名定义的 KCBPXA 定义的 KCBP 节点上处理。
- KCBP1 上的业务 A 是 KCBP2 上的业务 A 的代理。需要在 KCBP 的服务定义中一个代理类型的 LBM，它使用的 XA 名称对应 KCBP2。注意此处 KCBP2 上 LBM 名称与 KCBP1 上的 LBM 名称相同。

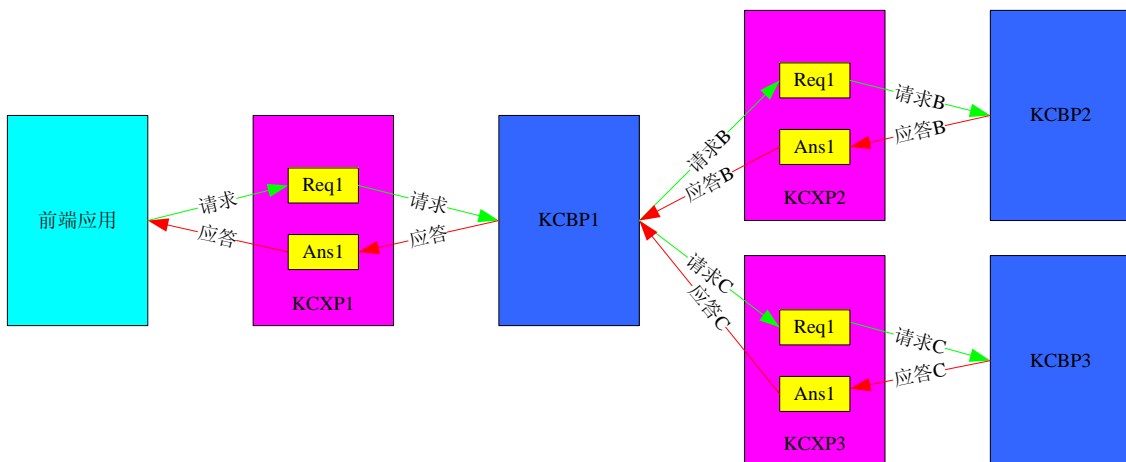
如果 KCXP1 和 KCXP2 是同一个 KCXP，KCBP1 和 KCBP2 需要使用不同的请求/应答队列。

### 8.4.2. 一对多转发部署 1



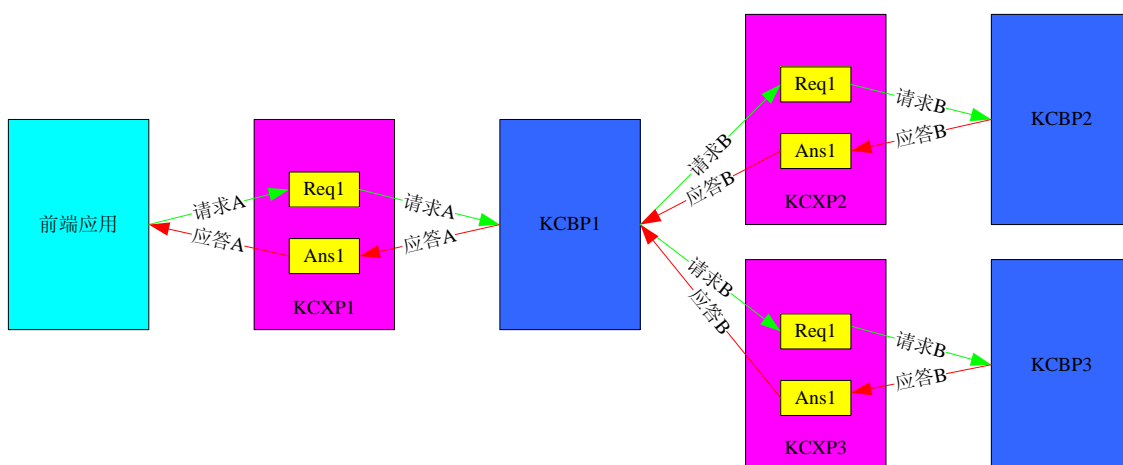
上图中 KCBP2 与 KCBP3 完成相同的功能，组成一个集群，对外统一提供服务。对于 KCBP1 来讲，它代理的是后面这个集群的业务。

### 8.4.3. 一对多转发部署 2



上图中 KCBP2 和 KCBP3 完成不同的功能，KCBP1 既代理了 KCBP2 的业务 B，也代理了 KCBP3 的业务 C，B 和 C 都配置在 KCBP1 的服务定义表中。

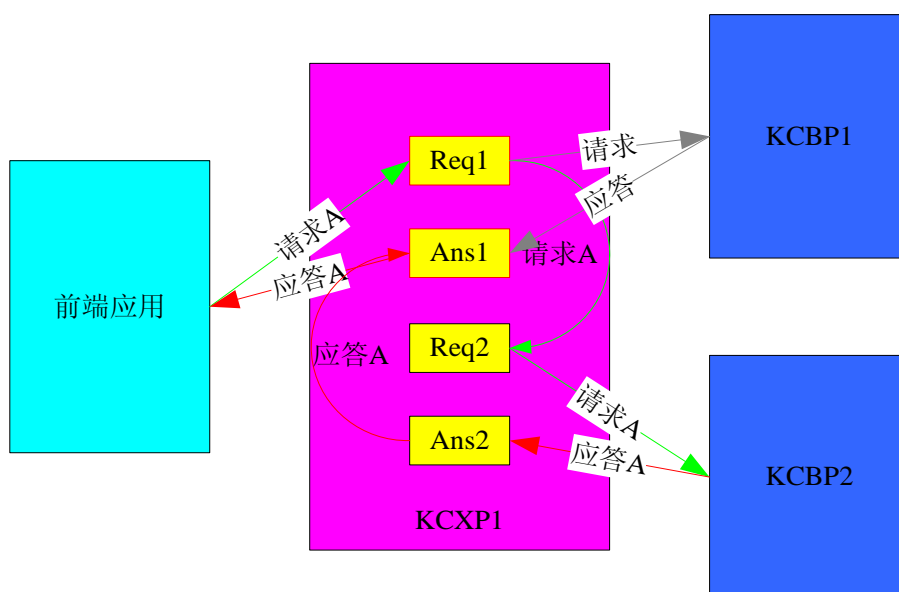
### 8.4.4. 一对多转发部署 3



KCBP2 上的业务和 KCBP3 上的业务内容实现不同（比如访问不同的数据库），名称皆为 B，KCBP1 代理了 KCBP2 和 KCBP3 上的业务 B，名称为 A。KCBP1 接到的请求 A 如何才能发送到正确的节点处理呢？这可以用获取代理名称的用户出口来实现。前提是前端应用需要传递表明请求处理节点特征的信息，可以在请求报文中传递，也可以用 KCBPCLI\_SetSystemParam 的 KCBP\_PARAM\_RESERVED 选项设置。此外，KCBP1 的 A 服务定义中，需要配置用户出口号为 22。

### 8.5. KCXP 的消息重定向

KCXP 插件也可以控制消息的转发。



KCXP 的 Exit.ini 中可配置转发条件，能实现按各种条件设置。上图中，前端应用发起的请求 A 送到 Req1 时，被插件重定向到 Req2 交给 KCBP2 处理，应答被插件由 Ans2 重定向到 Ans1 再返回给前端应用。

## 9. 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》
- ◆ 《TUXEDO 程序员手册》
- ◆ 《JMS1.1 规范》
- ◆ 《KCXP 应用程序编程参考书》