KCBP 用户手册

文档信息

项目名称	KCBP
标题	KCBP 用户手册
类别	文档
子类别	用户手册
摘要	
作者	杜玉巍
文档拥有者	KCBP 项目组
送交人员	用户
文件	KCBP 用户手册.doc- <文件类型：Microsoft Word>

修改历史			
版本号	日期	修改人	摘要
V1.0	2004-4-2	吴波	配置说明
V1.0	2004-7-26	吴波	升级后版本的配置说明
V1.2	2004-8-10	吴波	补充了错误处理说明。
V2.0	2005-8-21	杜玉巍	补充修订
V2.01	2005-9-20	杜玉巍	补充 KCBPDaemon 说明，参见 9.1 常见问题。 补充 kcbpcp 调用 LBM 时，如何输入系统变量说明，参见 7.3.2.3.3 KCBP 系统变量输入方法。
V2.02	2006-12-4	杜玉巍	增补 6.2.3.2.10 KDMID1PC 配置参数说明。 新增 6.2.3.2.5 KCBP1PCASYNC 配置说明。 新增 8.9.3 LBM 资源消耗过高问题定位。 新增 8.9.4 LBM 死锁问题处理方法。 新增 8.10 应用专题探讨，包括： 8.10.1 KCBP 部署规模估算一例 8.10.2 KCBP/Win 与 KCXP 系统间连接句柄数目优化 8.10.3 WIN 集中交易系统分离请求到备机优化方案 8.10.4 银证跨系统调用多机协同解决方案
V2.0.3	2007-12-10	杜玉巍	修改 8.10.3，增加保存和恢复参数 szGroupId，以适应异步请求与同步请求共用队列。
V2.5	2009-4-8	杜玉巍	增加 6.2.3.13Timer 属性页说明 增加 6.2.3.14IMDB 属性页说明 修改 6.2.3.1System 属性页说明 修改 6.2.3.9Program 属性页说明 修改 6.2.3.10KCXP 属性页说明 增加 7.3.2.4 循环调用 LBM 修改 7.6 压力测试工具 KCBPTest 相关内容 增加 7.8.3imdbcmd 使用说明 增加 8.1KCBP 高可用性介绍
V2.5.1	2009-6-10	杜玉巍	增加 KCBPTest 宏定义说明
V2.5.2	2009-9-6	杜玉巍	修改 Timer 例子说明，明确参数含义

V3.0	2010-7-8	杜玉巍	增加了用户出口 UserExit 配置页 LBM 增加了服务时间段及服务状态属性 废弃 LBMTest，将其功能合并到 KCBPTest 中 废弃 DebugLBM，将其功能合并到 kcbpcp 中
V3.1	2013/6/14	杜玉巍	Program 定义增加了 bpel 服务类型说明
V3.2	2015/10/12	陈洪涛	增加 listenmode=1,2,3,4 的处理方式
V2.90	2016/8/29	陈洪涛	增加 2016 优化版内容
V2.90	2016/11/24	李新革	1.增加 XA 并发控制的配置内容 详见章节 5.5.2.8 和 6.2.3.16 2.增加了 KCBPXARSL.xml 的介绍 详见章节 6.1
V2.90/ V2.81	2016/11/28	李新革	1.增加了 xa 并发控制的配置注意事项，详见章节 5.5.2.8.1

1. 引言

1.1 编写目的

这个手册是 KCBP/WIN 版的使用手册，为客户提供最新的、准确的、详细的 KCBP/WIN 使用指南、操作指导。

该手册从中间件的基本概念出发，详细描述了 KCBP 的运行环境、安装指导、操作指导，并有针对性地解答用户在应用中产生的各种疑问。

我们希望这本手册能够促进用户对交易中间件技术的把握，能促进用户对 KCBP 系统的理解，并能促进用户提高 KCBP 的应用水平。

我们力求手册内容的准确，但由于技术的迅猛发展和我们对新技术的掌握程度等方面的原因，我们还无法做到尽善尽美，因此我们欢迎用户提出的对 KCBP 系统和本手册的各种改进意见和建议，并争取将用户的宝贵建议付诸实践，改进 KCBP 系统。

1.2 背景

KCBP 是 Kingdom Core Business Platform 的简写，是金证公司开发的交易中间件。金证公司对中间件的开发，是从 90 年代初开始的，最早的中间件当时称作接口处理机，通过它来隔离前端应用程序和数据库，这种接口处理机，虽然不是完整的中间件，但已经具有中间件的朴素思想；金证公司真正意义的中间件，是在 1999 年开发的，叫做金证中间件 2.0，与金证 2.0 版证券交易系统配套，能完成通讯和业务处理功能，它提供一种简单的 Script 编程语言，能够描述业务处理逻辑，对复杂逻辑的处理，主要依赖数据库中的存储过程，Script 支持调用数据库中的存储过程。金证中间件 2.0 是金证的公司级中间件，金证公司的很多产品都是基于其上开发的，它是专用的中间件，运行在 Windows\Unix 平台，主要应用在证券业和银行业。在 2001 年，为了设计新一代的集中交易系统，金证公司开发部门成立了平台项目组，专门负责设计新一代的中间件，KCXP 和 KCBP 便属于新一代中间件的设计范围。

最著名的商业化交易中间件产品是 IBM 的 CICS 和 BEA 的 TUXEDO。金证公司在设计集中交易系统的过程中，也曾考虑过采用上述两个产品作为中间件。但由于上述两个产品是美国公司开发的，价格太高，势必增加系统建设成本；同

时，上面两个产品源代码不对外公开，遇到复杂一点的技术问题就需要国外解决，而国外响应周期长，证券交易系统又是实时系统，难以承担时间延误的风险，中国证券业的大客户更习惯使用具有源代码的产品，并且希望自己能够保留源代码，因为这样可以降低系统风险，而且更易于维护。中国证券行业的用户更看重软件供应商的服务。金证公司设计的系统，可以在与客户签订协议的基础上向客户提供源代码，包括中间件平台的源代码，此举是国外公司做不到、也不可能做的，这一点充分体现了金证公司把客户利益放到第一位的企业宗旨。另外，上述两个产品本身也有不足之处，由于他们主要是面向主机及 UNIX 平台设计的，因此在 Windows 平台上功能实现有缺陷，对 MS SQLServer 支持不足，在业务程序编程方式、服务进程运行状态控制、数据库断线恢复等方面都不完善，在 Windows 平台上的性能较低，难以满足证券业客户的要求。国内也有开发中间件产品的公司，但产品或多或少存在一些技术上和质量上的问题。金证公司在设计集中交易系统时，始终坚持硬件平台无关、操作系统平台无关、数据库无关等指导思想，为了更好的贯彻落实这些思想，决定自主开发交易系统的支撑平台，同时，金证公司经过 10 余年的发展，也具备了足够的技术储备和人力储备，公司上下也有坚定的信念。

KCBP 的设计目标是与业界最先进的中间件看齐，KCBP 的设计始终遵循商业中间件的技术标准，它是真正意义的交易中间件。KCBP 设计时充分考虑到了中国证券行业应用的实际情况，专门针对证券行业的特点进行针对性设计和特别优化，诸如为了提高业务程序访问多个数据库的速度，KCBP/WIN 单结点可以连接多个数据库、业务程序可以自由切换数据库、内存数据库技术等等。KCBP 的第一个版本是在 UNIX 平台上开发的，经过两年多的开发，KCBP 产品已经覆盖 OS/400、UNIX、LINUX、WINDOWS 等多个操作系统平台，并且已经在多个生产系统中投入运营，中信证券正是基于 KCBP/UNIX 开发了集中交易系统，并在 2003 年底投入生产运行。

国泰君安的集中交易系统，从 2000 年开始规划，经过近 3 年的论证、选型、测试，在 2003 年 8 月最终决定和金证公司合作，基于 WINDOWS 系统和 PC Server 构造，数据库采用微软公司的 SQLServer 2000。2003 年 7 月，为了验证系统的可行性，国泰君安证券组织集中交易系统招标入围厂家，在 INTEL 公司进行了集中交易系统架构可行性测试。金证公司应邀参加了这次测试。在此基础上，国泰君安与金证科技签订了共同开发新一代集中交易系统的合同。8 月中旬，KCBP/WIN 的开发工作全面启动，2004 年 3 月下旬，KCBP/WIN 正式投入生产。

1.3 定义

1. CTS：Central Trading System，中央交易系统。
2. BTS：Branch Trading System，分支交易系统。
3. KCBP：Kingdom Core Business Platform，金证核心业务平台。
4. KCXP：Kingdom Communication Exchange Platform，金证通讯交换平台。
5. LBM：Loadable Business Module，可装载的业务处理模块，它可以保证系统在不中断运行的情况下动态加载或卸载业务处理模块，实现系统功能的动态更新。
6. ACL：Access Control List，存取控制列表，KCBP 用于控制对资源的访问。
7. 应用程序

应用程序定义事务及在事务范围内访问的资源，每个应用程序指定一系列涉及资源（如数据库、文件系统）的操作。

8. 资源管理器

资源管理器管理计算机的特定共享资源，其他软件可以通过资源管理器提供的服务接口对资源访问。数据库管理系统（DBMS）、结构化文件系统（SFS，Structured File System）、消息管理器均可作为资源管理器。

9. 交易

联机交易处理（OLTP）的基本处理单位，也称事务交易。可以简单理解为一组操作的集合,操作的执行将使系统从一个状态转换为另一个状态。可能的操作包括数据通讯、业务逻辑处理和对多个共享资源的操作。

10. 交易(事务)特性

交易存在着四个基本特性（ACID），包括：

- 原子性（Atomicity）：一个交易所作的操作要么全部成功要么全部失败；
- 一致性（Consistency）：一个交易把一个合法对象（比如一条记录）从一种有效状态转变为另一种有效状态，如果该交易被放弃则此对象退回到交易开始前的有效状态；
- 独立性（Isolation）：一个交易对对象的操作效果在其被提交之前对于其它交

易是不可见的；

- 永久性（Durability）：一个成功的交易被提交后，其对对象的操作结果是永久性的，若要取消该操作结果必须通过另一个交易使它退回到原状态；

11. 全局事务

在分布式环境中，任何资源管理器都必须支持事务，以其自己的方式实现内部可恢复的一组操作，同时，这些资源管理器也有相应的功能支持跨资源管理器的一组操作，即全局事务。

12. 事务分支

一个全局事务包含一个或多个事务分支，一个分支是一个资源管理器上属于此全局事务的所有操作的总和。当应用程序在一个全局事务中对多个资源管理器进行操作时，此全局事务便包含多个事务分支。

13. 只读事务

没有写操作的事务分支称为只读事务。

14. 事务管理器

事务管理器管理全局事务，协调统一的提交、回滚以及故障恢复。应用程序通过调用事务管理器定义全局事务的起始点，事务管理器返回一标识符（XID）来标志此全局事务。各资源管理器接受这个事务标识符(XID)并对应到相应的本地事务分支，这样，当事务管理器要完成这个事务时，资源管理器可以知道该对那个本地事务做处理。

事务管理器和资源管理器之间通过 XA 接口交换事务信息。

15. XA 接口

X/OPEN 组织定义的一个交易管理中间件产品与资源管理系统（一般为数据库管理系统）之间的一个接口标准，该标准不直接提供给应用程序开发使用。

16. 动态事务复原

如果事务中的某个操作任务失败，未落实的修改将自动逆序恢复，将可恢复资源恢复到事务的起始状态或最近的同步点。

17. 交易通讯模式

KCBP 支持同步、异步和会话三种交易通讯模式。

- 同步模式：客户方发送一笔交易请求后必须等待服务方返回的应答信息，在收到服务方应答前不作其他任何处理；
- 异步模式：客户方发送一笔交易的请求后不必等待服务方回送应答，可以去执行其他任务，如发送另一笔交易请求或处理应用逻辑等，用户可以使用状态查询函数查询交易状态；
- 会话模式：通讯双方在进行通讯时，首先建立通讯连接，并交替进行数据的发送和数据的接收，在信息交互完成后拆除连接，会话结束；

18. 交易确认方式

    - CLIENT 控制：对资源的更改由 CLIENT 端通过请求方式控制提交/取消，SERVER 端在等待 CLIENT 端确认时要锁资源；
    - SERVER 控制：对资源的更改马上实现；

19. 交易描述信息

KCBP 为了能正确的控制一笔交易的执行，需要获取交易的基本描述信息，交易描述信息由应用程序填写在一组数据结构中。

20. 交易数据

KCBP 支持分布式客户/服务器结构的应用程序，客户方应用程序与其对应的服务方程序之间的具体交易数据以消息（即数据包或文件）形式在通讯网络上传递。交易数据的传输格式与通讯中间件无关。通讯中间件只是消息运载工具。应用递送的消息可以有两种组织方式：

- 字符流方式：数据格式由用户自定义，KCBP 不做解释；当然，用户可以使用 XML 来描述信息格式。
- 二进制位流方式：类似于 C 语言的 struct 数据类型，应用在使用二进制位流方式时要对结构中的数据类型进行描述，由 KCBP 核心进行解释并进行与字符流的自动转换，以保证应用数据的跨平台交互。

21. 命名服务

名字即交易码，也可以是请求编号，KCBP 提供按名字请求服务的方式。客户程序按名字提出服务请求，中间件根据名字服务表查找并分配服务请求，服务程序可以按优先级进行服务处理。KCBP 的名字服务采用交易码驱动方式实现，按照交易码名字来确定一笔交易的服务进程。客户程序在请求交易服务时给出交易码

信息，KCBP 根据交易码服务程序对照表查找相应的服务程序，交易码由用户自行定义。

22. 节点

一台虚拟机器，是一套由 KCXP 和 KCBP 构建起来的应用运行环境，一般与一个实际的业务机构相对应。在一个节点上可以有多个应用程序运行，多个节点可以在一台实际的机器上运行。节点的概念相当于 CICS 的 REGION。

23. 通道

通道是客户程序与 KCBP 服务器之间或者不同 KCBP 服务器之间消息交换的渠道。通道由消息中间件 KCXP 提供。

24. 通讯程序

通讯程序管理节点的连接，提供数据和文件的可靠传输，通讯程序按 KCBP 要求的接口规范提供接口函数，KCBP 当前使用的通讯程序是消息中间件 KCXP。

1.4 参考资料

文档名称	版本号	备注
国泰君安证券集中交易方案	V2.0	
KCBP 需求分析	V1.4	
KCBP 系统概要设计	V1.1	
KCBP 程序员手册	V2.2	
KCBP 系统介绍	V2.0	
KCBP/WIN 项目总结	V1.0	
KCBP/UNIX 用户手册	V1.0	

2. 中间件技术简介

2.1 中间件技术的产生

在过去的几千年中，人类创造了有两种最重要的文明：机械文明和信息文明。

机械文明，解放了人类的手脚；信息文明，解放了人类的大脑。机械文明已经有几千年的历史。信息文明则是在最近一个世纪才突飞猛进的，尤其是在计算机发明之后。计算机能存储程序，能按照人类的要求进行信息处理。计算机使人类能够在全世界范围内进行脑力劳动分工协作。信息文明则是建立在计算机技术之上的。

计算机技术主要包括计算机制造和计算机应用两类。计算机制造方面的发展主要体现在三个方面：处理速度的提高、信息存储容量提高、信息交换能力提高。

计算机应用技术的发展，也就是软件技术的发展。软件技术的发展，反映了人类认知能力的不断发展，人类对软件技术不断抽象和提炼，积累了越来越多的解决问题的模式，软件的种类越分越细，应用软件的开发越来越简单。

在计算机技术发展的早期，计算机是由硬件组成的；后来，硬件中的可变化部分抽象成了软件，通过软件来操控硬件进行处理的；再后来，随着人们认知能力的不断提高，逐渐意识到软件是可以分成不同类别的，其中相对恒定的是系统软件，可变的是应用软件，这样就形成了系统软件和应用软件两类；随着人们认知能力不断提高，又意识到，千差万别的应用软件也是可以归类的，每种类别都有各自的处理模式，并且处理模式可以单独提取出来，作为一类软件而存在，这类软件就是中间件。中间件是介于系统软件与应用软件之间的，它使人们可以按照一些预知的模式开发应用软件，这样做能缩短应用软件开发的周期，并能增加系统的稳定性和可靠性。

2.2 中间件的概念

中间件，与操作系统、数据库并列为三大基础软件平台。今天，中间件已经包括了许多分支，包括通讯中间件、交易中间件、对象中间件、安全中间件、数据访问中间件、应用服务器等等。

IDC 是这样定义中间件的：中间件是一种独立的系统软件和服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源，中间件位于客户机服务其

的操作系统之上，管理计算资源和网络通信。

实际上，在计算机软件系统的层次结构中，中间件位于操作系统、数据库、网络层之上，在应用程序之下，是一个中间层，因此得名。

中间件是技术发展和应用需求发展的必然产物。软件技术的发展，是有鲜明时代特征的，20 世纪 70 年代是操作系统的时代，80 年代数据库的时代，90 年代则是中间件迅猛发展的年代。

交易中间件是中间件的一个重要分支，主要用在 OLTP 中。90 年代以前建设的商业应用系统结构比较单一，主要以主机-终端模式及客户-服务器模式为主，客户-服务器模式也称为二层结构。到了 90 年代，随着计算计应用水平的飞速发展，企业信息化水平的不断提高，企业客户的不断增加，以及新业务的不断出现，越来越多的用户计算机应用系统在并发处理、关键业务联机交易处理、跨平台、跨广域网、多数据源等方面提出了更高的要求，为了解决这些实际需求，诞生了以交易中间件为基础框架的应用模式，这种架构业界称之为三层结构。三层结构的核心概念是利用中间件将用户界面、业务逻辑和数据逻辑分为三个不同的处理层。以中间件为基础的三层结构系统，不仅具备大型机的系统稳定、安全、处理能力高的优点，同时也具有开放式系统成本低、可扩展性强、开发周期短、易维护等优点。

2.3 中间件的分类

依据解决问题的范畴不同，中间件主要分为以下几类：

2.3.1 消息中间件

将数据从一个应用发送到另外一个应用，这就是消息中间件的主要功能。消息中间件主要用来解决跨操作系统及不同的网络环境的应用间通讯问题。它要负责建立网络通信的通道，进行数据的可靠传送、保证数据不重发，不丢失。消息中间件的一个重要作用是可以实现跨平台操作，为不同操作系统上的应用软件集成提供数据传送服务。它适用于进行非实时的数据交换，如银行间的结算数据传送。消息中间件的主要产品有：IBM MQSeries、BEA MessageQ、BEA Tuxedo/Q，Microsoft MSMQ、金证公司的 KCXP、东方通公司的 TongLink/Q。

2.3.2 交易中间件

交易中间件和消息中间件一样，也具有跨平台、跨网络的能力。他的主要功能是管理分布在不同计算机上的数据的一致性，协调数据库处理分布式事务，保障整个系统的性能和可靠性。交易中间件遵循的主要标准是 X/Open DTP 模型。它适用于连机交易处理系统，如银行的 ATM 系统，电信的计费系统。主要产品有 IBM CICS、BEA Tuxedo、KCBP、东方通 Tong Easy。

2.3.3 对象中间件

对象中间件也称 Object TP Monitor，它一般也具有交易中间件的功能，但它是按照面向对象的模式来组织体系结构的，在线的电子交易系统很适合采用这种类型中间件，因为这种类型的应用会被频繁修改，面向对象的体系结构可以保持足够的弹性来应付这种改动。面向对象的中间件，现在有 3 种 ORBs 结构：CORBA、COM、EJB。ORBs 是 ObjectRequestBrokers 的简写，他是一组协议或标准，现在的对象中间件一般都是按照其中的某一种来构造的。主要的产品有：Borland VisiBroker 、 Microsoft Transaction Server 、 IONA orbix 、 IBM ComponentBroker、东方通 Tong Broker，开放源代码的 TOMCAT。

2.3.4 应用服务器

应用服务器主要用来构造基于 WEB 的应用，它是企业实施电子商务的基础平台。它一般是基于 J2EE 体系结构的。它可以让网络应用的开发更加容易，使开发人员专注于业务逻辑。主要的产品有：IBM Websphere、BEA Weblogic、Borland Appserve，中科院网驰公司的 ONCE、金碟公司的 APUSUS。还有一些开发源代码的 J2EE 应用服务器，如 JBOSS。

2.3.5 企业应用集成

企业应用集成，Enterprise Application Integration（EAI）。

一个大企业内部往往有很多的计算机应用系统，EAI 可用于对这些系统进行有效的整合，使他们之间能够互相访问，实现互操作。EAI 所提供的上层开发工具或许是 EIA 和其他中间件的最大区别，他允许用户自定义商业逻辑和自动使数据对象符合这些规则。EAI 的典型用户是那些巨型企业的大量应用系统的整合。以前的 EAI 中间件一般都是在通讯基础上构建的，目前的 EAI 中间件开始

构建在 J2EE 中间件之上。

2.3.6 安全中间件

近几年，随着互联网的发展，信息安全越来越受到普遍关注，安全中间件也应运而生。安全中间件是一公钥基础设施（PKI）为核心的，建立在一系列相关的国际安全标准上的一个开放式应用开发平台，向上为应用系统提供开发接口，向下提供统一的密码算法接口及各种 IC 卡、安全芯片等设备的驱动接口。

2.3.7 其他类型的中间件

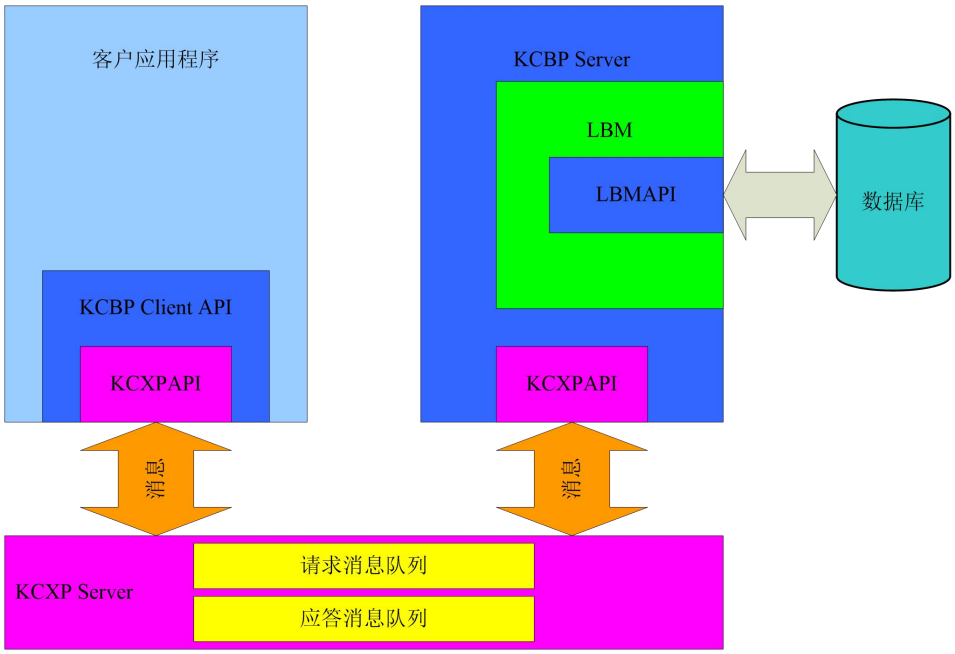
上面对中间件的分类描述并不是一个很严格的定义，只是一个大致的划分。此外，还有很多专业公司为客户提供某一领域的更符合客户需求的中间件产品，如清华北美的 THMT、JAVATS、青牛公司的 CT-USE 等。

3. KCBP 系统结构

为了便于理解以后各章的内容，我们在这章对 KCBP 的系统结构做一个简要介绍。

3.1 KCBP 应用系统结构图

KCBP 应用系统典型结构图如下：



整个系统由 KCBP 系统及 KCBP 相关系统构成。

KCBP 系统由以下几个子系统构成：

- KCBP Server
- KCBP Client API
- LBMAPI
- KCXPAPI
- KCXP Server

KCBP 相关系统包括：

- 客户应用程序

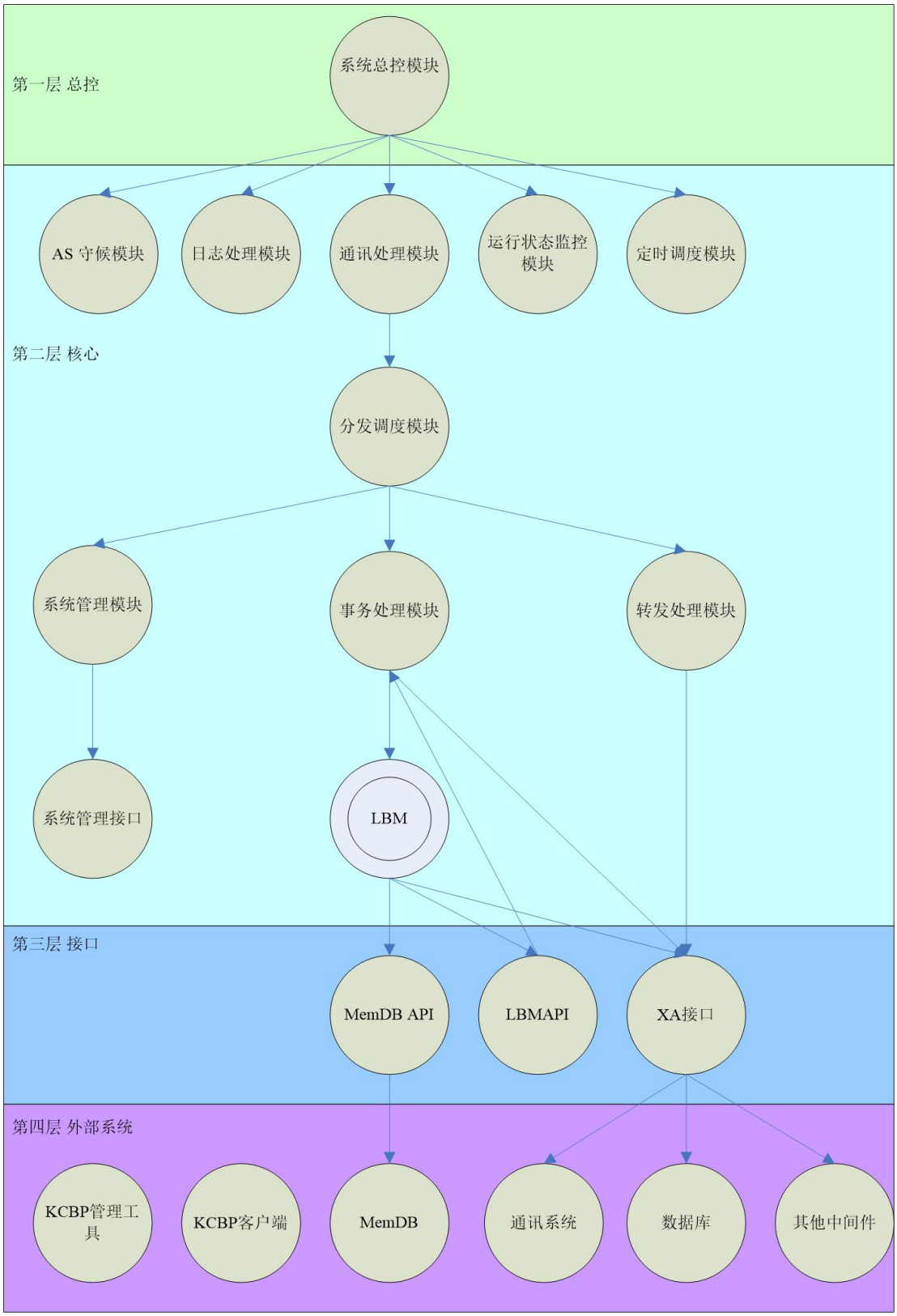
- LBM
- DB

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。KCXP 是 KCBP 的通讯子系统。KCXP 是独立系统，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。

3.2 KCBP 系统层次结构图



说明：

上图是 KCBP/WIN 的总体层次结构图，KCBP/LIUNX 与此在第二层不完全相同，此处要注意。

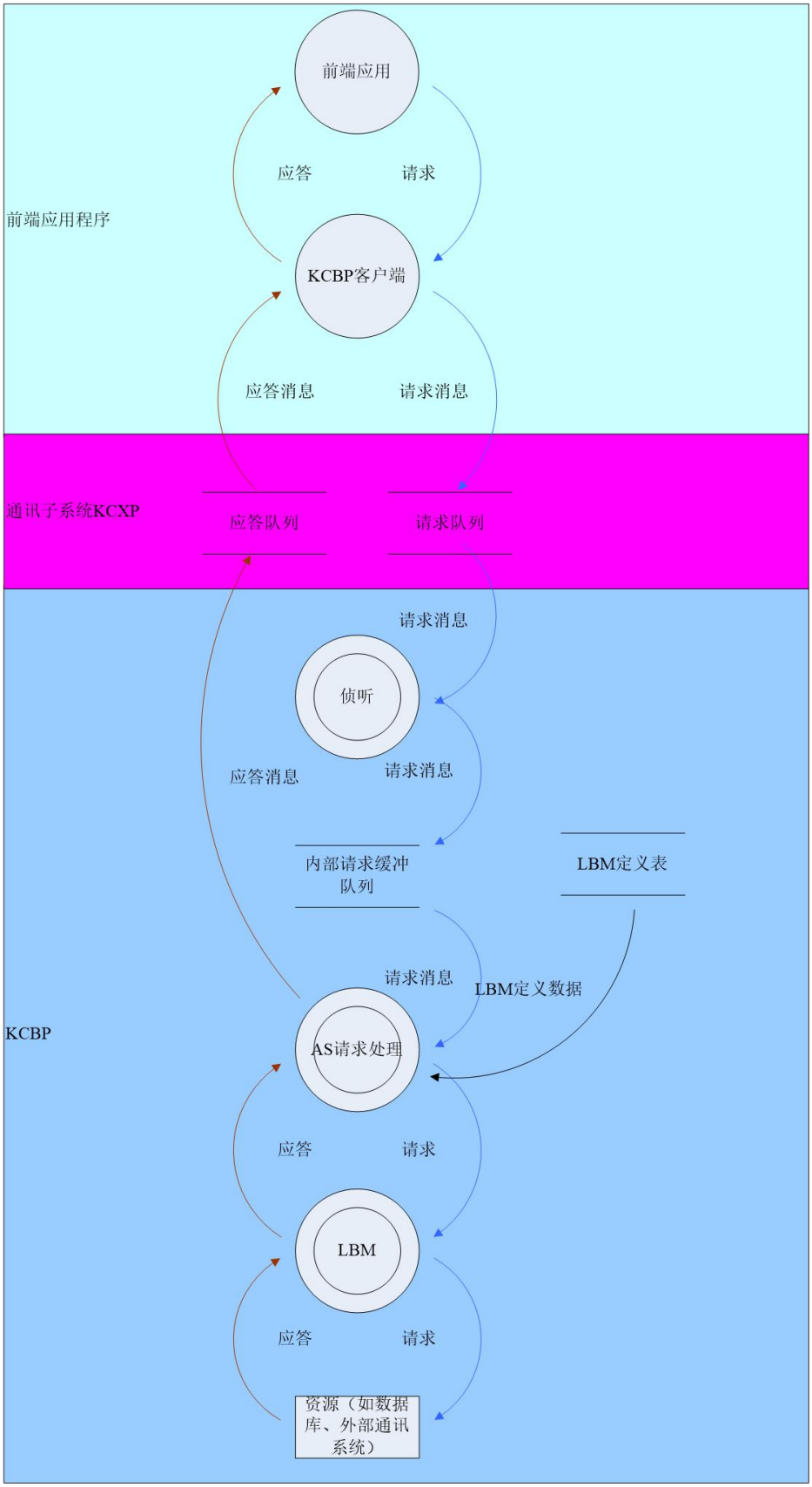
第一层，总控层，包括系统运行界面，负责启动、停止系统，观察、调控系统的运行状态。

第二层，核心功能执行层，完成系统的各种核心功能，包括通讯、分发调度、事务处理、第三方服务代理、转发、日志处理、系统管理(也负责 KCMM 交互)、定时任务调度、运行状态监控、统计信息收集、AS 进程守候等。

第三层，接口层，负责与 KCBP 核心、数据库、第三方交易中间件、通讯中间件、监控管理工具等交换信息。

第四层，外部系统，与 KCBP Server 相关的系统，包括数据库、第三方交易中间件、通讯中间件、监控管理工具、KCBP 客户端等。

3.3 KCBP 请求处理流程图



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。

4. KCBP 主要功能

KCBP/WIN 主要功能如下：

4.1 任务管理

KCBP 处理事务的机构：事务监控器+应用服务器。

事务监控也称作交易管理模块，它负责调度应用服务器进程，事务监控器自动分配各应用服务器中的客户请求。

事务监控器自动回收由于各种异常情况(如用户程序编制错误)导致崩溃的应用服务器进程占用的资源，并根据系统的压力情况自动动态调整应用服务器的规模。

事务监控器通过可恢复进程和落实协议保证分布式交易的 ACID 特性。

每个应用服务器是一个包含多个线程的进程，各线程分别完成不同的功能。

每个 KCBP 节点包含多个应用服务器进程。

应用服务器进程可以并发处理同一事务的不同实例。

应用服务器通过与一个或多个资源管理器交互来处理客户的请求。

应用服务器验证用户对资源的访问权限。

应用服务器成功完成用户请求后，落实数据的更改，终止该事务并释放资源以用于其它事务，如果事务失败，则将更改全部回滚。

在程序上，KCBP 主进程完成该部分功能。主进程由以下模块组成：

控制界面：完成系统运行控制功能，包括系统各模块启动、停止、运行状态显示等。

侦听模块：完成连接检测、接受请求、向内部缓冲队列传送请求等功能。

主控模块：完成侦听线程及应用服务器的启动、停止、异常监控功能。

故障恢复模块：完成崩溃进程资源回收、恢复应用服务器进程功能。

日志模块：完成日志收集、显示、记录等功能。

4.2 程序管理

KCBP 系统服务程序定义在 KCBPSPD.XML 文件中，SPD 是 Service program definition 的缩写。KCBP 系统运行时，服务程序定义存放在内存 XML RTDB 数据库中。共享内存中，有一份全局的程序定义表，全局表也称作公有表。每个应用服务器进程都有一份自己私有的服务程序定义表，这是全局表的一个备份。二者区别是公有表不保存 Cache 相关信息，私有表保存每个已经 Cache 的 LBM 文件句柄和模块入口地址。公有表和私有表都有各自的最后更改时间戳，如果私有表与公有表时间戳不一致，各服务进程执行公有表到私有表的同步操作。KCBP 通过表时间戳控制初始化、插入、删除等操作的全局表与局部表同步。表中每一行都有各自的行时间戳，通过它可以控制行更新同步。程序的添加、删除、修改等操作首先在公有表上完成，各应用服务器进程采用延迟更新算法更新各自的私有表。

KCBP 系统提供管理命令用来完成程序项添加、删除、修改等功能，客户端管理命令在 kcbpcp 中输入。

Server 端程序管理由初始化模块、XML 管理模块、RTDB 管理模块、应用服务器模块等共同完成。

4.3 系统资源管理

KCBP 系统的资源包括数据库资源和通讯资源。KCBP 通过 KCBPXA 接口规范对资源进行统一管理。

KCBPXA 接口规范是我们独创的，对 X/Open 的 XA 接口规范进行了再抽象，兼容线程和进程等调用方式。

KCBP 系统对于资源的访问全部基于 KCBPXA 接口。

KCBPXA 的强大功能：单个 KCBP 结点可以连接多个资源，对各资源的操作可以集成在一个事务处理服务器内，对于提高复杂系统处理效率具有重要意义。

KCBP 服务代理机制全部建立在 KCBPXA 接口规范之上，该部分内容将在后面专门叙述。

目前，已经实现的资源管理接口包括：

ODBC1PC.DLL：提供 ODBC 方式访问各种数据源的功能。

DB21PC.DLL：提供 ESQL/C 方式访问 DB2 的功能。

DB21PC_R.DLL：提供 ESQL/C 线程方式访问 DB2 的功能。

ORACLE1PC.DLL：提供 ESQL/C 方式访问 ORACLE 的功能。

ORACLE1PC_R.DLL：提供 ESQL/C 线程方式访问 ORACLE 的功能。

MQXA.DLL：提供访问 MQ 通讯资源的功能。

CICSXA.DLL：提供访问 CICS 资源的功能。

TUXEDO.DLL：提供访问 TUXEDO 资源的功能。

KCBPXA.DLL：提供访问其他 KCBP 系统资源的功能。

MEMDBXA.DLL：提供访问内存数据库的功能。

4.4 数据通信

该项功能是指 KCBP 提供与客户端通信的功能，不包括 KCBP 与其他 KCBP 结点和其他系统（如 CICS、MQ）通讯的功能。

目前，KCBP 与客户端通信，主要通过 KCXP 完成。有以下几种通信方式：

- 同步调用：又分为
  - Server 端确认
  - Client 端确认
- 异步调用
- 发布/订阅：该部分功能在后面叙述。

该部分功能由 KCBP Client API 模块和应用服务器模块完成。

4.5 安全性管理

KCBP 系统的安全性包括以下几个方面：

- 客户安全性

  用户定义在文件 KCBPUsr.xml 中，系统运行时，用户定义信息装入 RTDB 中，与安全相关的设计包括。

  - 口令加密保存。
  - RSLKey 定义用户对资源的访问权限。
  - 客户端连接时，口令采用密文传输。

- 资源访问安全性

  资源访问口令加密保存。

  授权访问资源：通过服务程序的 ACL 控制，具有 PUBLIC 和 PRIVATE 属性，对于 Private 资源，需要定义该资源的 RSL key，该资源只限拥有该 RSL Key 的用户访问。

- 服务器间通讯安全认证

  认证发起请求的远程系统。

  认证初始发起请求的客户(可控制到每笔交易)。

  授权其对资源的访问。

- 应用安全性

  KCXP 提供 SSL 及多种加密算法(用户可定制)保证传输安全、完整。

  通过用户出口进行加/解密处理。

系统安全管理相关模块较多，这里不一一列举。

4.6 服务代理

代理是指 KCBP 将 CICS、TUXEDO、其他 KCBP 结点等交易中间件作为 KCBP 的资源，从 KCBP 的角度看，不论 CICS，还是 TUXEDO，

他们与数据库一样，都是具有某种特性的资源管理器。

通过代理机制，KCBP 解决了多 KCBP 系统互联、异种中间件互连问题，可以有效协调发生在各异种中间件的分布式交易，对于已经建成的系统进行功能扩充具有重要意义。

KCBP 通过各被代理者的编程接口操作其资源，KCBP 的代理机制是 KCBP SERVER 端的功能， CLIENT 不必关心。

被代理者有自己的商业逻辑，KCBP 提供符合其商业逻辑的编制规则的 API 用来设计其商业逻辑，如对于 CICS，KCBP 提供 LBMAPICICS，对于 TUXEDO，KCBP 提供 LBMAPITUXEDO。

通过调用 LBMAPI 编制出的业务程序具有跨中间件平台无缝移植的特点，移植时，不需要修改业务程序源代码，只要经过重新编译，生成的代码就可以运行在其他中间件上。

目前，KCBP 已经实现的代理模块包括：

CICS 代理、TUXEDO 代理、KCBP 代理。

KCBP 系统设计中，有转发模块一项内容，现在更名为 KCBP 代理模块，在 kcbpxa.dll 中实现。

4.7 负载均衡

负载均衡的作用：

- 优化分布式环境的工作负载
- 增强应用的可用性、连续性和可扩展性
- 充分利用现有的计算能力
- 提供在不间歇情况下维护和升级系统的解决方案
- 节省硬件费用

KCBP 的负载均衡实现方式：

- IP 负载均衡
- KCXP 通讯流量负载均衡

- KCBP 集群负载均衡

客户端：

- 方法一：建立一个客户端连接，连接时制定多个 Server 地址，KCBP 采用随机算法选择其中一个 Server。
- 方法二：建立多个客户端连接，应用程序自己依照 ROUND ROBIN(轮询)或 BIASING(加权轮询)算法通过已经建立的连接发送请求。

SERVER 端：

- KCBP+KCXP 多节点均衡容错方案（MSSQ 技术）

4.8 MemoryDB

目标：减少 DISKIO，提高应用系统处理效率。

CACHE 表特征：数据量小、访问频繁、只读。

存放位置：表数据存放在共享内存中。

装入方式

- 系统初始化装入。
- 按需装入，由 LBM 调用 API 装入。

刷新方式

- 定时刷新，时间间隔可配置。
- 按需刷新。

访问方式

- LBM 可通过 API 装入表，操作表。

该部分功能由 Memdbxa.dll 实现，它是 KCBP 的一个资源管理模块，表缓冲由用户的 LBM 实现，Memdbxa 提供管理内存数据库的 XA 接口，它采用 KCBPXA 规范，表访问由 LBM 调用 IMDB API 完成。

4.9 发布 / 订阅

KCBP 应用服务器具有消息 BROKER 功能，提供按主题发布/订阅功能。

角色

- 消息订阅者(SUBSCRIBER)。
- 消息发布者(PUBLISHER)。

发布过程

- 主题登记：消息发布者向 KCBP 做一笔发布主题登记交易
- 发布：消息发布者向 KCBP 做一笔消息发布交易
- 撤销主题：撤销 BROKER 中的发布主题

其中，主题登记和撤销可选。

订阅过程

- 订阅：订阅者向 KCBP 做一笔订阅交易，按主题订阅消息
- 接收
  - 通过 KCXP 消息触发消息处理程序
  - 通过 KCBP API 接收消息
- 撤销订阅：订阅者向 KCBP 做一笔撤销订阅交易，撤销订阅

发布订阅动能由 kcbpsys.dll 完成，客户端通过发布/订阅 API 操作。

Kcbpsys.dll 中包含 KCBP 系统自用的几个 LBM。

4.10 日志管理

日志是按时间先后顺序记录的系统运行情况流水，利用日志可以完成系统故障跟踪、系统调试、系统审计、维护事务完整性等功能。KCBP 提供系统运行日志记录系统各方面的运行情况。

日志管理模块在主控进程中运行，是主控进程的一个线程。系统其他模块使用日志模块提供的 API 向日志管理模块发送日志信息。日志管

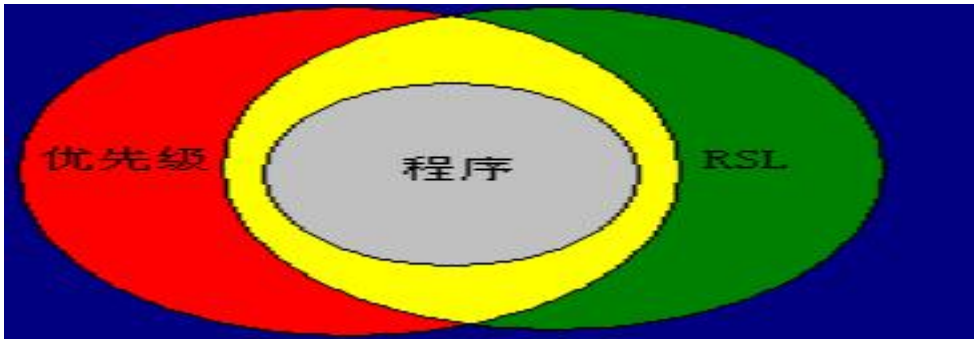
理模块负责日志的显示和记录。

4.11 事务并发性控制

KCBP 提供三种并发控制方式：

- 按 LBM 控制
- 按优先级(Priority)控制
- 按 RSL 控制

三种控制方法可以共同存在，相互关系如下图：



并发控制模块在 KCBP 系统应用服务器中实现。是否采用并发控制、采用何种并发控制方式可以通过参数控制，参数通过 KCBP 系统配置程序设置。

优先级并发限制定义在 KCBPPrio.XML 中，RSL 并发限制定义在 KCBPRSL.XML 中，程序并发限制定义在 KCBPSPD.XML 中。

并发检查功能由应用服务器模块完成。

4.12 对外接口

KCBP 提供两种对外接口：Server API 和 Client API。

CLIENT 端提供 C API。头文件是 KCBPCLI.h，接口程序模块是 KCBPCLI.DLL，相关程序文件是 KCXPAPI.DLL、KCBPCRYPT.DLL、KCXP 的插件 LIB*.DLL 及插件定义文件 KCXPPLUGIN.DAT。

Client API 接口，分为两层，高层 API 和底层 API，高层 API 现在广泛使用，它建立在底层接口之上，封装了表操作和交易调用等高级功

能；底层接口，又称作 KCBP CLIENT ECI 接口，是 KCBP CLIENT API 的最核心实现，设计简洁，只有一个调用函数，但功能强大，能完成各种高层接口无法完成的功能，目前 KCBP CLIENT ECI 接口未对外公布。

Server 端 API 也称作 LBMAPI，它是 C 语言 API，它支持用 ESQL/C 技术或数据库平台的 NATIVE METHOD（如 DB2 CLI、ORACLE OCI、ODBC 等）访问数据库，并提供访问 KCBP 系统资源的函数。LBM 是 Loadable Business Module 的缩写，它是 KCBP 服务端业务处理程序，C 或 C++编写，以动态连接库形式存在，相关属性(包括路径、export 函数名称等)定义在 KCBPSPD.xml 中。客户端采用按名方式调用 LBM。客户端程序与服务端程序通过 KCBP 的通讯区交换数据。

SERVER：KCBP 提供 makefile 范例,使用 gmake 或 nmake 编译，makefile 中自动调用 kcbpadd 工具向 kcbp 系统增加服务程序。

除了 CLIENT API 和 LBMAPI 之外。我们还提供了 DBAPI，用来访问数据库。数据库访问接口是一个抽象类，与核心层分离，它的调整不会对核心层产生影响。DBAPI 分为 2 层，下层是基本 API，上层是高层 API。DBAPI 目前的一个实现是基于 ODBC 封装的，已经全面投入运行。该部分功能由 ODBC1PC.DLL 实现。

KCBP 还提供适用于报盘 API 接口及银证 API 接口的外部通讯 API。该部分功能由 KCXPXA.DLL 实现。

内存数据库模块提供外部访问 API，该部分功能在 IMDBG.DLL 中实现。

为了方便 LBM 程序的编写，系统还提供了 GeneralLBMAPI.DLL 供 LBM 调用，其中封装了上述各种 API 的功能。

4.13 管理界面

KCBP 提供 2 种管理界面：

- 字符界面：由命令处理器 kcpbcp 提供，使用时需要与 KCBP Server 端建立连接，本身完成命令输入、请求发送、接收结果、显示结果等功能，修改过的参数动态生效。kcbpcp 命令风格类似 db2 命令。

- 图形界面：C++编写的静态管理界面，由 KCBPSetup.exe 提供，单独使用，不需要与 KCBP Server 配合，直接操作 KCBP 的配置文件，对处在运行中的 KCBP 进程不产生作用。

5. 安装及运行

5.1 获取最新安装程序

最新的 KCBP 安装程序、KCXP 以及相关程序可以通过 Internet 下载，下载网址是 http://www.szkingdom.com。

5.2 安装 KCXP

下载后，运行 Setup.exe，选择完全安装即可。

安装后，双机桌面上的 KCXP 队列管理器图标 

运行队列管理器，选择系统=>安装 KCXP 服务菜单，将 KCXP 安装为 Windows 服务程序。

5.3 安装 KCBP

KCBP 是绿色安装包，下载后拷贝到目标目录，就可以运行了。如果需要使用以前的配置文件，可以把以前版本 Bin 下的*.ini,*.xml 拷贝过来。

5.4 KCBP 目录结构说明

KCBP/WIN 的目录和文件列表如下：

5.4.1 BIN 目录

文件名称	说明
cicsxa.dll	CICS one phrase commit 接口动态连接库
cygintl-1.dll	系统动态库
cygpcre.dll	系统动态库
cygwin1.dll	系统动态库

db21pc.dll	DB2 一阶段提交接口库
db21pc_r.dll	DB2 一阶段提交接口库（多线程版）
DumpLUW.exe	LUW 浏览程序
GeneralLBMAPI.dll	LBMAPI 高层接口
grep.exe	系统工具，用于模式搜索
IMDBg.dll	内存数据库动态库
Item.xml	统计项定义 XML 文件
KCBP.exe	KCBP 主程序
kcbpadd.exe	LBM 命令行添加工具
kcbpas.exe	KCBP 应用服务器程序
KCBPCli.dll	KCBP 客户端 API
kcbpcli.xml	KCBP 客户端配置文件
kcbpcmd.exe	KCBPCP 命令运行环境
KCBPConf.xml	KCBP 配置文件
kcbpcp.exe	KCBP 命令处理器
kcbpcrypt.dll	加密动态库
KCBPCWA.dat	KCBP CWA 文件影像
KCBPDaemon.exe	KCBP 进程守候者
KCBPError.xml	KCBP 错误定义文件
kcbpkill.exe	KCBPAS 服务进程清除工具
KCBPPacketOpApi.dll	KCBP 报文处理接口库
KCBPPrio.xml	KCBP 优先级定义文件
KCBPPub.xml	KCBP 消息发布登记文件

KCBPRSL.xml	KCBP 资源访问级别定义文件
KCBPSetup.dll	图形设置界面动态库
KCBPSetup.exe	图形设置界面主程序
KCBPSPD.xml	KCBP 服务程序定义文件
KCBPSub.xml	KCBP 消息订阅登记文件
kcbpsys.dll	KCBP 系统自用 LBM 动态库
kcbptest.exe	KCBP 压力测试工具
kcbptest.ini	KCBP 压力测试工具配置文件
KCBPUsr.xml	KCBP 用户定义文件
kcbpxa.dll	KCBP 结点间通讯文件
kcxpapi.dll	KCXP 接口库
kcxpapi.ini	KCXP 接口配置文件
kcxpplugin.dat	KCXP 插件定义文件
kcxpuser.dat	KCXP 用户定义文件
kcxpxa.dll	KCXP 通讯资源接口库
kill.exe	系统工具
lbmapi.dll	LBM 接口库
libdes.dll	KCXP 插件库
libeay32.dll	KCXP 插件库
libidea.dll	KCXP 插件库
Libidea1.dll	KCXP 插件库
libzip.dll	KCXP 插件库
libzlib.dll	KCXP 插件库

Libzlib1.dll	KCXP 插件库
mqxa.dll	MQ 通讯资源接口库
odbc1pc.dll	ODBC 一阶段提交接口库
oracle1pc.dll	ORACLE 一阶段提交接口库
oraclelbm.dll	ORACLE 演示 LBM
publish.exe	发布演示程序
SSLeay32.dll	KCXP 插件库
subscribe.exe	订阅演示程序
tuxedoxa.dll	TUXEDO 资源访问接口库
xerces-c_2_7_0.dll	XML 处理接口库
xerces-depdom_2_7_0.dll	XML 处理接口库
lbmadapter.dll	以动态库形式按 KCBPXA 规范提供 KCBPAS 全部功能，主要与 KCBPTest、KCBPCP 配合使用。

5.4.2 DOCS 目录

文件名称	说明
KCBP 程序员手册.pdf	编程手册
update.txt	版本升级说明文件
KCBP 用户手册.pdf	用户手册。
KCBP 编程补充资料.txt	列出系统参数和取请求包包头里的参数
KDMID1PCFactory 用户手册.pdf	KDMID1PCFactory 用户手册

memhq 编程说明.txt	memhq 编程说明
内存行情系统 MemHQ 操作说明.pdf	内存行情系统 MemHQ 操作说明

5.4.3 INCLUDE 目录()

文件名称	说明
KCBPCli.h	KCBP 客户端 C API 头文件
kcxpapi.h	KCXP 客户端 API
lbmapi.h	LBMAPI 头文件
KCBPXA.h	KCBPXA 接口头文件
Imdbdll.h	内存数据库接口头文件
KCBPPacketOpExport.h	KCBP 报文处理函数库头文件
KCBPDatabase.h	基于 odbc1pc 的数据库访问接口抽象类
contools.hpp	kcxpxa 的接口头文件
GetHq.h	Memhq 的头文件
GeneralDblib.h	
GeneralLBMAPI.h	

5.4.4 LIB 目录

文件名称	说明
KCBPCli.lib	KCBP 客户端 lib 文件
Kcxpapi.lib	KCXP 客户端 lib 文件
lbmapi.lib	LBMAPI 的 lib 文件

IMDBg.lib	内存数据库的 lib 文件
kcxpapi.lib	kcxp 的 lib 文件
GeneralLbmapi.lib	GeneralLbmapi 的 lib 文件

5.4.5 samples\server\onephrase 目录

文件名称	说明
LBMODBC.cpp	用 ODBC 操作数据库的例子程序
LBMTest.cpp	LBM 例子程序

5.4.6 samples\server\userexit 目录

文件名称	说明
UserExit.cpp	用户出口的例子程序
UserExit.h	用户出口头文件

5.4.7 samples\client 目录

文件名称	说明
kcbpcli_async.cpp	客户端异步操作的例子程序
kcbpcli_gen.cpp	客户端同步操作的例子程序
kcbpcli_sql.cpp	客户端 SQL 风格函数的例子程序
kcbpcli_xa.cpp	客户端使用 KCBPXA 的例子程序
publish.cpp	KCBP 发布的例子程序
subcribe.cpp	KCBP 订阅的例子程序

5.4.8 JNIAPI 目录

文件名称	说明
kcbpclijni.jar	Java 接口包
KCBPCliJNI.dll	Java 接口 JNI 动态库
KCBPTest.class	测试程序
Test.java	测试程序

5.5 配置简介

5.5.1 KCXP 配置简介

在 KCXP 管理器界面选择 系统=>KCXP 配置向导=>自动完成即可。界面如下：

经过自动配置后，KCXP 使用所在机器的 21000 端口和客户通讯。

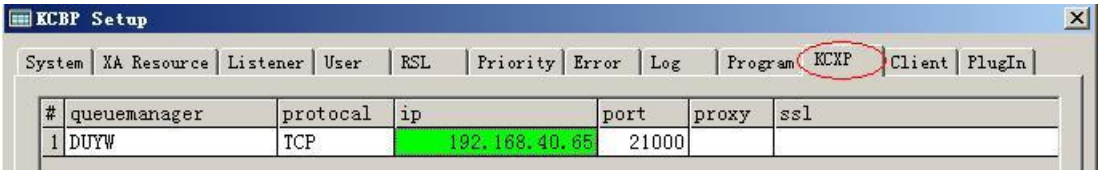
5.5.2 KCBP 配置简介

5.5.2.1 启动 KCBP 配置界面

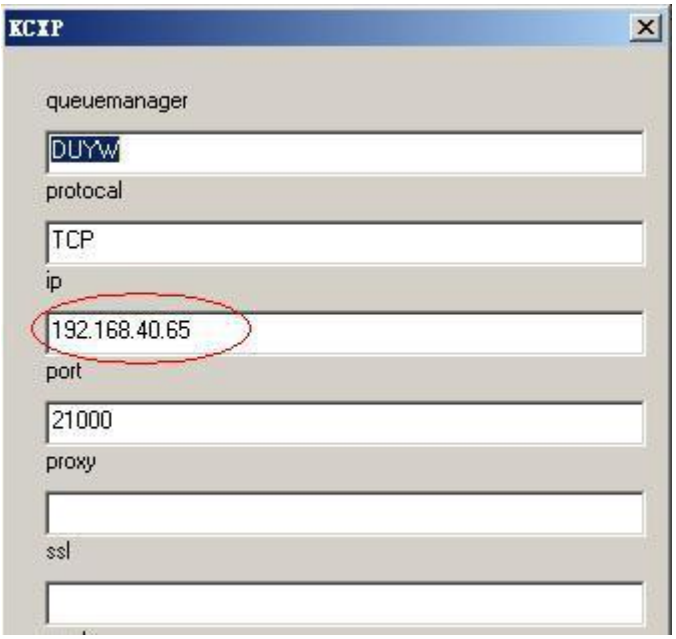
双击桌面上的 

图标，启动 KCBP，选择工具->图形管理器菜单，启动图形管理器。

5.5.2.2配置 KCXP 通讯参数



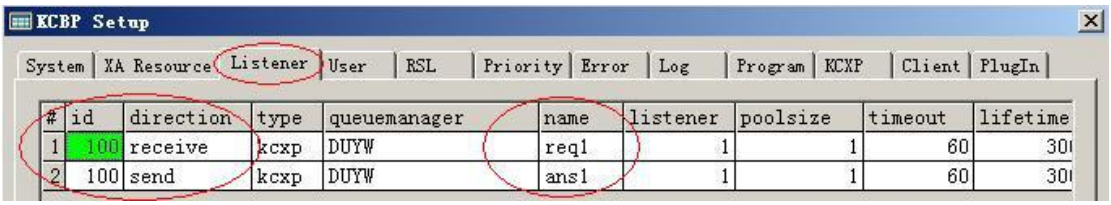
选择 KCBP Setup 的 KCXP 属性页（红圈标识），然后在 192.168.40.65 处双击鼠标左键，弹出下面对话框：



在 ip 地址输入行（红圈标识）输入 KCXP 服务器的 ip 地址。

5.5.2.3配置 KCBP 侦听队列

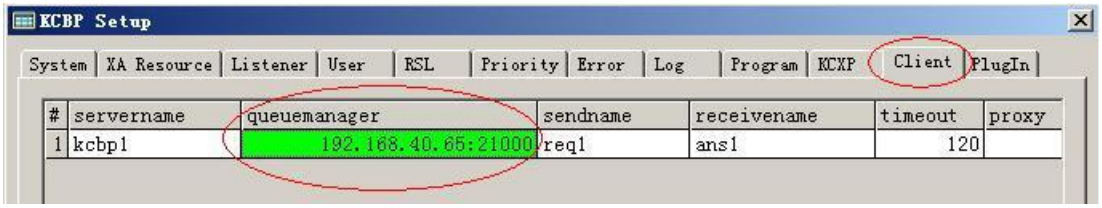
KCBP 安装时，缺省配了一组队列 req1 和 ans1，这个参数不用修改。



注意红圈里面的内容：两个队列 id 相同，direction 相反，name 不同。Queuemanger 名称与前面 KCXP 参数配置页面的 queuemanager 相同。

5.5.2.4配置 Client 连接参数

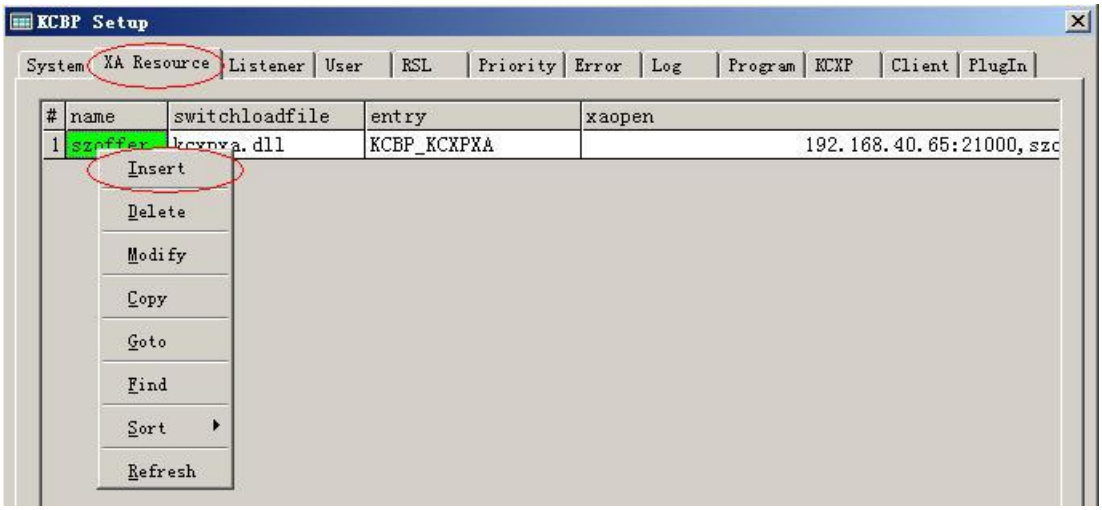
在 Client 属性页面（红圈标识）的 192.168.40.65:21000 行双击鼠标，修改 queuemanager 中的 ip 地址为 KCXP 服务器的 ip 地址，或将 queuemanager 设置为 DUYW。



Client 定义的参数给 kcbpcp 使用。

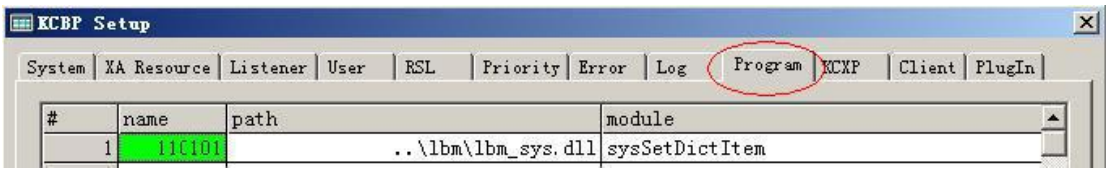
5.5.2.5配置 XA Resource

如果不访问数据库或其它资源，此步可跳过。如果连接数据库，需要增加 XA Resource，方法是在 XA 页面点击鼠标右键，选择弹出菜单的 Insert 菜单项，然后再对话框中设置 XA 属性。关于 XA 属性的具体含义见后面配置章节的 XA Resource 页面配置说明。



5.5.2.6配置 Program

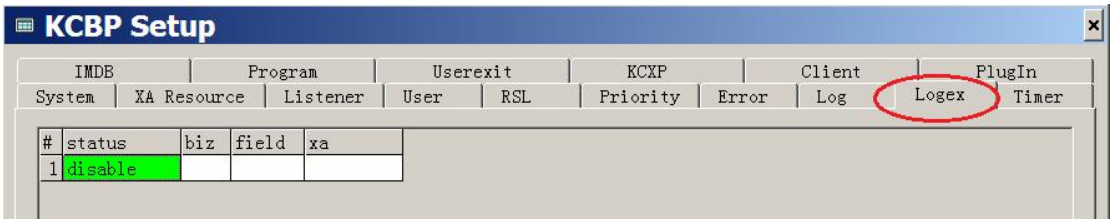
用户可以在 program 属性页(红圈标识)增加新的 LBM。



经过上面的配置后，KCBP 就可以运行了。

5.5.2.7配置 logex

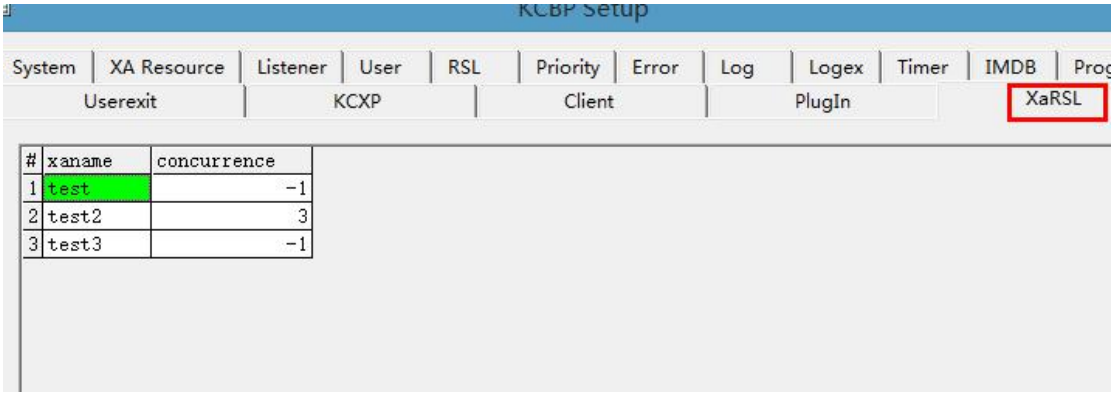
配置扩展日志功能。可以对日志做过滤设置。



5.5.2.8配置 xarsl

配置 xarsl 功能。当通过 XA 调用外部系统时,可以按照 XA 名称进行并发控制，防止整个系统不可用。

concurrence 表示要控制的并发数，-1 表示不控制并发(xa 名称未在此配置的也表示不控制并发)



在此处修改后，需要重启 KCBP 才能生效。也可以通过 kcbpcp 命令在 KCBP 运行时修改配置，不用重启 KCBP，命令如下所示:**注意字符之间要有空格**

```
kcbp => select * from xarsl
  Retrieved 3 node.
  Node 1:
    Table : xarsl
    concurrence='-1' xaname='test'
  Node 2:
    Table : xarsl
    concurrence='2' xaname='test2'
  Node 3:
```

Table : xarsl
concurrence='3' xaname='test5'

kcbp => insert into xarsl (xaname,concurrence) values (test6,3)
Insert 1 node.
kcbp => update xarsl set (concurrence=4) where xaname=test2
Update 1 node.
kcbp => delete from xarsl where xaname=test6
Deleted 1 node.
kcbp =>

5.5.2.8.1 xarsl 配置注意事项：

1.xarsl 中所有配置的并发总数，应该小于界面上 min as 的数量
就是 xarsl 的配置中，所有 concurrence 大于 0 的数值加起来，应该小于 KCBP 的 min as 配置数量。
2.通过 kcbpcp 动态修改 xarsl 的原理是：向 KCBP 发送 soap 请求，KCBP 处理 soap 请求，修改内存信息。极端情况下，假如 KCBP 的所有 AS 都被占用，那么通过 kcbpcp 也动态修改不了 xarsl 配置，此时如果要修改 xarsl，要手动修改 KCBPXARSL.xml 的配置，然后重启 KCBP.

5.6 运行

系统启动顺序是先启动 KCXP 通讯服务，再启动 KCBP。

5.6.1 运行 KCXP

1. 启动 KCXP 服务
   - 运行 kcxp 管理程序（kcxpmanager.exe）,在系统托盘区有一个托盘程序，如图所示：（运行状态指示为红色）

   - 在托盘程序图标上点击鼠标右键，在弹出式菜单中选择“启动 KCXP 服务”，如图所示：

- 启动 KCXP 服务成功以后，KCXP 托盘程序的图标会变为如下图所示：（运行状态指示为绿色）

- 如果启动服务不成功，则会报错，可检查 KCXP 管理程序的配置文件。

2. 停止 KCXP 服务
   - 在托盘程序图标上点击鼠标右键，在弹出式菜单中选择“停止 KCXP 服务”，如图所示：

   - 停止服务成功以后，托盘程序的图标的运行状态指示由绿色变为红色，如图所示：

3. 检查 KCXP 的运行状态
   - 当托盘程序的图标的运行状态指示为绿色时，表示 KCXP 服务已经启动。如图所示：

   - 当托盘程序的图标的运行状态指示为红色时，表示 KCXP 服务已经停止。如图所示：

   - 此时如果要启动 KCXP 服务，可按照第 1 点中的说明进行。

5.6.2 运行 KCBP

5.6.2.1 启动 KCBP

- 在 KCBP 界面中执行“系统”菜单下的“启动”子菜单，或直接点击起动按钮，则启动 KCBP。如图：

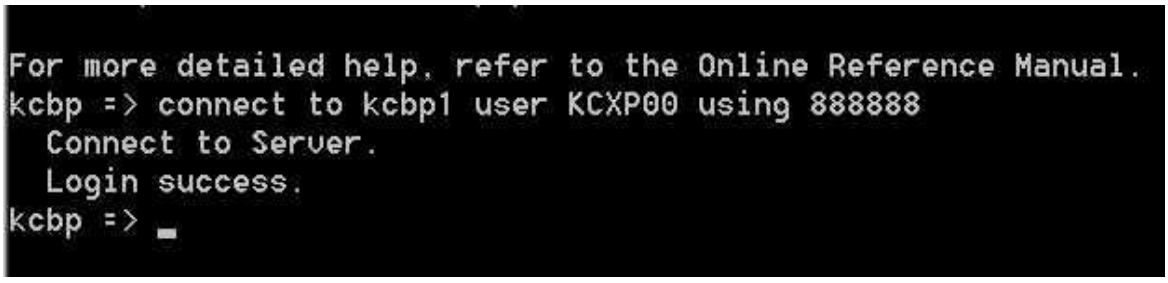
- 当 KCBP 启动成功后，暂停和停止按钮由灰色变为彩色，（通过此特征可判断 KCBP 是否正常启动）。如图：

5.6.2.2 退出 KCBP

- 执行“系统”菜单下的“退出”子菜单，退出 KCBP
- 直接点击“退出”按钮。
- 直接关闭 KCBP 窗口。

5.6.2.3 检测 KCBP 是否正常工作

- 执行“管理”菜单下的“字符管理器”子菜单，出现 KCBP 命令管理器窗口。
- 在该窗口中，执行如下命令，登陆到 KCBP：connect to kcbp1 user KCXP00 using 888888，如果登陆成功，则说明 KCBP 正常工作。如下图所示：



5.6.2.4KCBP 启动时常见错误及解决办法

1) 错误现象：连接 KCXP 队列失败；
处理办法：查看 KCXP 服务是否已经启动。
2) 错误现象：连接数据库失败；
处理办法：检查是否能够连接到数据库。
3) Kcbpcp 连接报错：Login error : Authentication error。
处理办法：Connect 使用的用户需要在KCBP服务端的user中配置group为manage。KCBP 的 User 支持 manage 和 application 两个组，manage 有管理权限，application 有 LBM 调用权限。

6. 配置

6.1 配置文件说明

文件名	用途
KCBPConf.xml	KCBP 主配置文件，存放系统参数、XA、Listener、Log 等参数
Kcxpusr.xml	存放 KCBP 用户信息
KCBPSPD.xml	存放 LBM 定义信息
KCBPError.xml	存放 KCBP 内部错误代码和名称对照信息
KCBPPub.xml	存放 KCBP 消息发布信息
KCBPSub.xml	存放 KCBP 消息订阅信息
KCBPRSL.xml	存放 KCBP 权限控制信息
KCBPPrio.xml	存放 KCBP 优先级定义信息
KCBPplugin.xml	存放 KCXP 界面插件信息
KCBPcli.xml	存放 KCBPCP 配置信息
Refer.xml	存放 KDMID1PC 处理规则
KCBPcli.ini	KCBP 客户端配置信息，系统保留，不对外提供
Kcxpapi.ini	KCXP 配置信息
KCBPtest.ini	KCBPtest 压力测试工具配置信息
KCBPUserExit.xml	存放用户出口定义信息
KCBPXARSL.xml	存放 XA 并发控制信息

6.2 图形化配置工具 KCBPSetup 用法

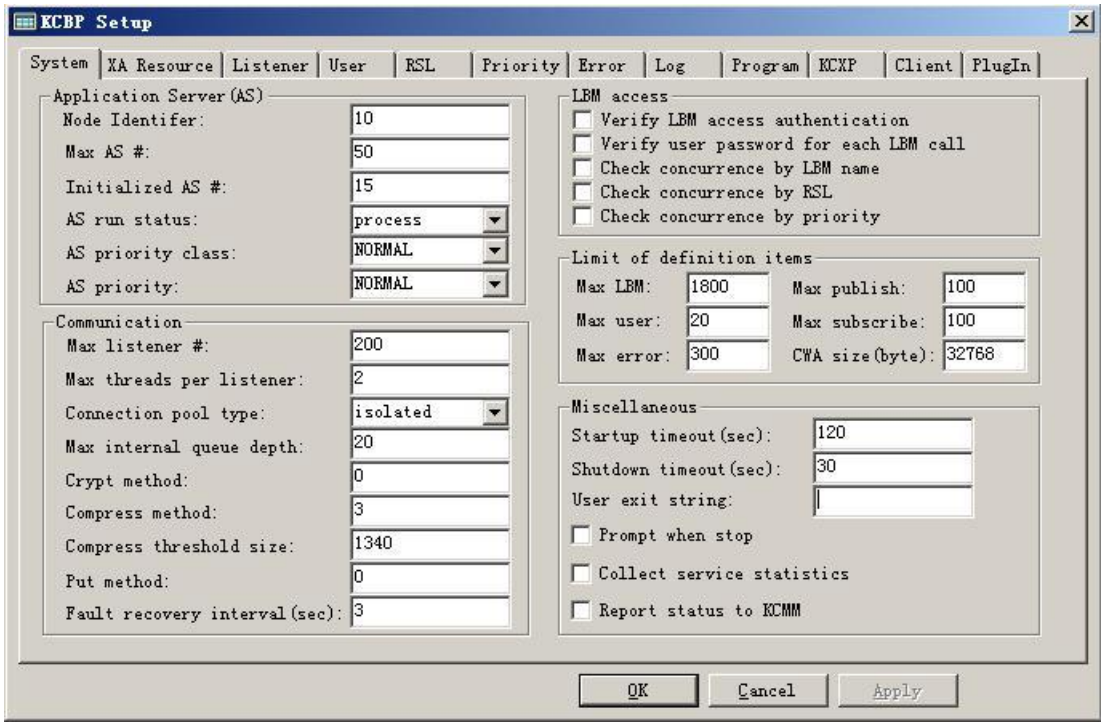
6.2.1 简介

KCBP 图形管理器用图形界面来管理 KCBP 的配置文件，包括：

- KCBPConf.xml
- KCBPSPD.xml
- KCBPUsr.xml
- KCBPError.xml
- KCBPPrio.xml
- KCBPRSL.xml.
- KCXPAPI.ini
- KCBPCli.xml
- KCBPPlugin.xml
- KCBPUserExit.xml

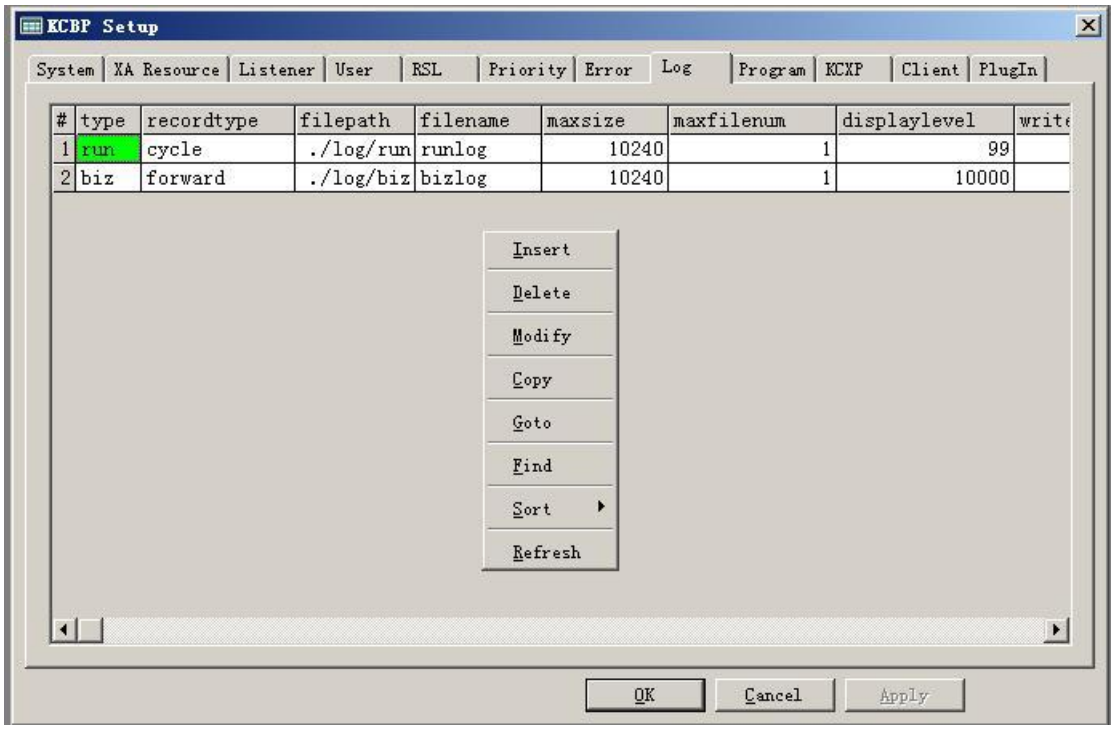
- KCBPXARSL.xml

KCBP 管理界面包含多个属性页，每个属性页包含一个配置表。



属性页名称	配置文件	节点名
System	KCBPConf.xml	System
XA Resoruce	KCBPConf.xml	Xa
Listener	KCBPConf.xml	Externalqueue
User	KCBPUsr.xml	User
RSL	KCBPRSL.xml	Rsl
Priority	KCBPPrio.xml	Priority
Error	KCBPError.xml	Error
log	KCBPConf.xml	Log
Timer	KCBPTimer.xml	task
Program	KCBPSPD.xml	Program
Kcxp	KCXPAPI.xml	Kcxp
Client	KCBPCli.xml	Externalqueue
Plugin	KCBPPlugin.xml	Plugin
UserExit	KCBPUserExit.xml	userexit
XaRsl	KCBPXARSL.xml	xarsl

提供了增加，删除，修改等功能，通过在显示表格的属性页面上**点击鼠标右键弹出这些功能菜单**。在每条记录上双击鼠标，则可以显示该条记录的详细内容，并可以修改各属性的值。



KCBP 图形管理器包含 KCBPSetup.exe 和 KCBPSetup.dll，它是完全独立于 KCBP 主服务程序的。

6.2.2 启动

在 KCBP 主控程序界面的管理菜单下选择图形管理器菜单项(图 3)，或者点击 ToolBar 上的图形管理器图标(图四以红色圈起)，就可以打开 KCBP 图形管理器。

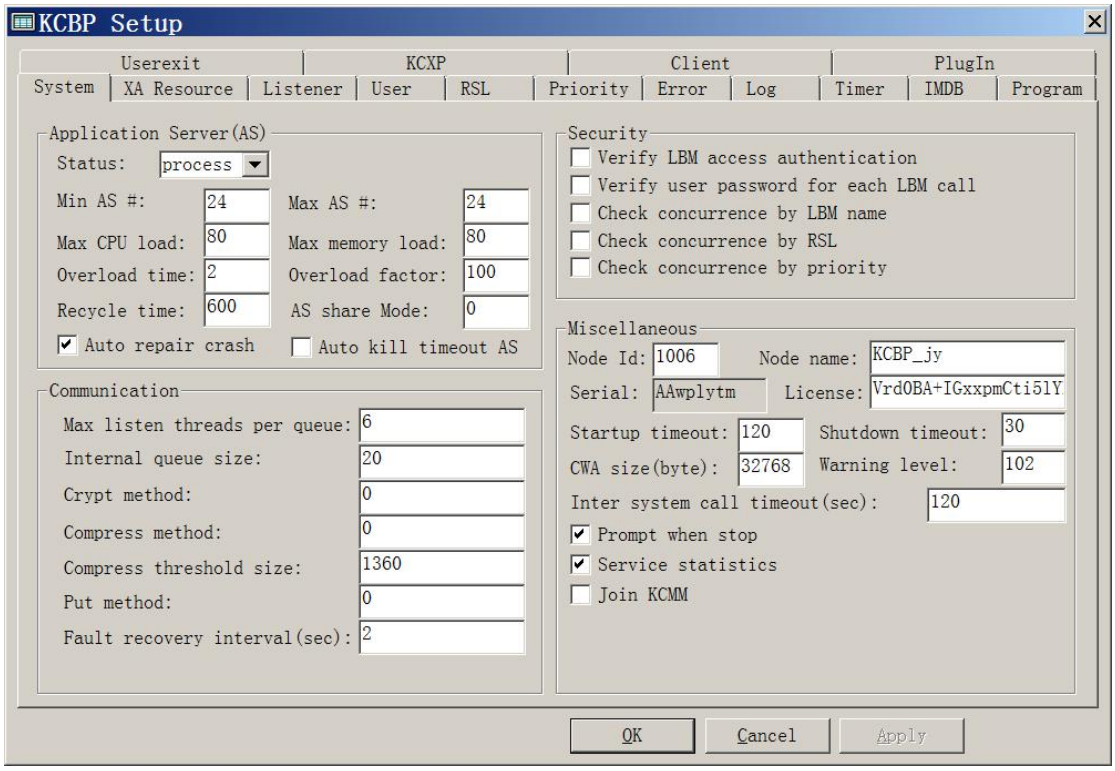
图 3

6.2.3 属性页面配置说明

6.2.3.1System 属性页

下图是 System 属性页，用来配置 KCBP 系统的运行参数，这些参数存放在 KCBPConf.xml

中，节点名称是 system。



配置共分为五类:

- Application Server（AS）
- Communication
- Security
- Miscellaneous

页面配置项与 KCPBConf.xml 节点属性对照表：

类别	界面名称	XML 属性名	说明
Application Server	Max AS #	maxworker	应用服务器进程数目上限，取值范围 1-1024
	Min AS #	maxas	应用服务器进程数目，取值范围 1-1024。如果是进程方式运行 KCBP，这个数目是当前正在运行的 kcbpas.exe 进程的数目，并可在任务管理器中看到。这个参数影响系统的性能，一般来讲，当 CPU 有空闲时，这个参数值越大，并发处理能力越强，如果 CPU 没有空闲，调大这个参数，将增加系统调度开销，可能降低系统性能。

	Max cpu load	maxcpuload	允许 AS 进程自动伸缩的最大 CPU 负载百分比
	Max memory load	maxmemoryload	允许 AS 进程自动伸缩的最大内存负载百分比
	Overload factor	overloadfactor	触发 AS 进程自动增加的有效负载百分比
	Overload time	overloadtime	触发 AS 进程自动增加的负载持续时间，单位秒
	Recycle time	maxidletime	触发 AS 进程自动收缩的触发时间，单位秒
	AS run status	runas	应用服务器运行状态，有进程和线程两种状态。进程方式运行优点是稳定，单个 LBM 崩溃不会影响到整个系统的运行，并且 KCBP 能恢复崩溃的进程；线程方式运行的优点是速度快，处理性能比进程方式快大约 10%，缺点是没有进程方式稳定，LBM 崩溃将造成整个 KCBP 系统崩溃。我们推荐使用进程方式运行 KCBP。
	AS priority class	asprioritycla ss	AS 进程优先级，取值范围： REALTIME High Normal Idle。 这个参数控制 KCBPAS 进程的优先级，缺省值是 Normal。一般不用调整这个参数。如果需要调整这个参数，需要先了解 WINDOWS 操作系统对进程和线程优先级的控制机制。
	AS priority	aspriority	AS 线程优先级，取值范围： TIME_CRITICAL HIGHEST ABOVE_NORMAL NORMAL BELOW_NORMAL LOWEST IDLE 这个参数控制 KCBPAS 进程的优先级，缺省值是 Normal。一般不用调整这个参数。如果需要调整这个参数，需要先了解

			WINDOWS 操作系统对进程和线程优先级的控制机制。
	Auto kill timeout AS	autoreclaimer	自动终止超时业务，yes 或 no，慎用 yes。
	Auto reapir crach	Selfrepair	KCBP 进程崩溃自动修复，yes 或 no，推荐 yes。
	AS share mode	listenmode	AS 侦听方式。 0：由侦听线程接收，收到请求后放入内部队列里缓存。推荐方式为 0。 1：AS 直接侦听，按 kcbpas 序号根据队列 listener 数分配。 2：AS 直接侦听，按 kcbpas 序号循环侦听每个队列。 3：和方式 1 相同，有等待控制。 4：和方式 2 相同，有等待控制。
Communication	KCBPConf.xml 文件中配置	maxinput	侦听队列最大数目，取值范围 1-1000。侦听队列在 listener 中配置，方向标志是 receive。侦听队列和应答队列成对出现，二者有相同的 id，但方向不同。
	Max listen threads per queue	maxlistener	每个侦听队列可配的最大侦听线程数目。取值范围 1-32。这个参数用来控制 listener 中侦听队列属性 listener 的最大值，用于防止用户误操作。
	KCBPConf.xml 文件中配置	pooltype	连接池类型，取值范围：isolated 和 combined 。 缺 省 值 是 combined，这时，每个 kcbpas 进程对 listener 中配置的同一个 队列管理器只建立一个 socket 连接。如果是 isolated，每个 kcbpas 进程对 listener 中配置的 每个队列都建立一个 socket 连接。Combined 可以减少系统中 socket 句柄的数目，在关闭 KCBP 然后重新启动时，不会遇到 socket 连接错误，而 Isolated 可能遇到这个错误，这个错误的产生的原因是关闭 KCBP 时，KCBP 中建立的 Socket 有 2 分钟的 TIME_WAIT 保护时间。

	Internal queue size	maxqueuedepth	内部请求缓冲队列长度。KCBP 内部采用了一个请求缓冲队列，用于缓存所有 listener 队列的请求。这个值建议≥Initialized AS #。
	Crypt method	crypt	通讯加密算法，取值范围 0-3。缺省值 3。0 表示不加密。"1"，"2"，"3"表示三种加密方式；"1"是 idea 方式，"2"是 des 方式，"3"是 idea1 方式。
	Compress method	compress	通讯压缩算法，取值范围 0-3。缺省值 3。0 表示不压缩。"1"，"2"，"3"表示三种压缩方式；"1"是 zip 方式，"2"是 zlib 方式，"3"是 zlib1 方式。压缩算法一般会消耗 30%CPU。
	Compress threshold size	compressthreshold	压缩阀值，单位为字节。缺省值 1340 字节。当通讯压缩算法不等于 0 时，KCBP 只对大于阀值的数据报文进行压缩。
	Put method	put	发送应答时采用的通讯算法，值为 0-2。缺省值为 0。值为 0 时使用 KCXP_Put 时发送应答，值为 2 时使用 KCXP_Put2 时发送应答，KCXP_Put2 比 KCXP_Put 速度快大约 25%，但这是有代价的，值为 2 时如果发生 KCXP 断线情况，当恢复连接后，每个 AS 进程发送的应答会被 Windows 丢失一笔，这时前端会返回 2011 超时错，值为 0 时没有这个问题。
	Fault recovery interval	retryinterval	断线重连间隔时间，单位为秒。缺省值 3 秒。这个值越小断线恢复越快，但当连接不能恢复时，由于单位时间内调用次数多，所以 CPU 利用率会高。
LBM access	Verify LBM access authentication	verifiylbmauthentication	调用 LBM 时进行权限检查，值为 yes 或 no。当值为 yes 时，调用 LBM 时检查权限。KCBP 客户端使用客户名和密码登录，客户名称在 user 表中定义，每个 user 可有一个或多个 rslkey，program 中定义的每个 LBM 都

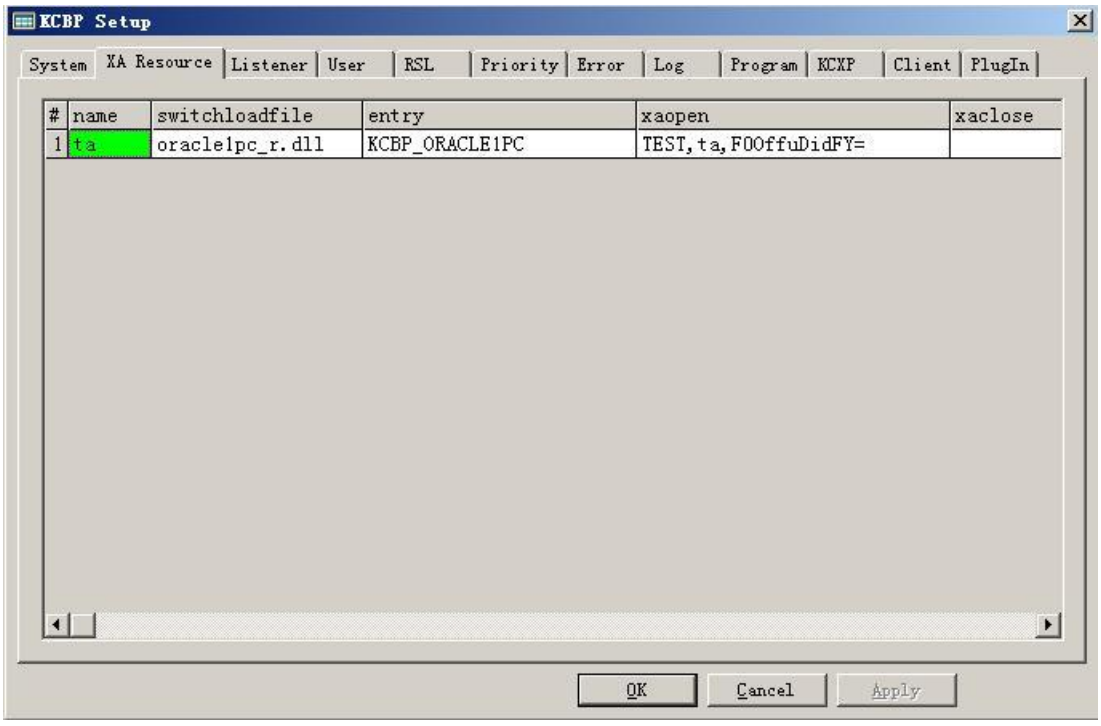
			有一个 acm 属性和一个 rsl 属性，当 acm 属性值为 private 时，该 LBM 只能被有该 rsl 的用户调用。
	Verify user password for each LBM call	bizverifypassword	调用 LBM 时检查用户及密码合法性。值为 yes 或 no。当值为 yes 时，每次调用 LBM 时都验证用户及口令，这时系统更安全，但由于每次调用都进行一次用户口令检查，会降低系统的效率。
	Check concurrence by LBM name	limitprogramconcurrence	根据 LBM 名称控制并发数开关。值为 yes 或 no。当值为 yes 时，将按照 LBM 的 concurrence 数目控制并发，Concurrence 如果是-1，不控制并发数。
	Check concurrence by RSL	limitrslconcurrence	根据 LBM 的 RSL 控制并发数开关。值为 yes 或 no。当值为 yes 时，将按照 LBM 的 RSL 编号分组控制并发，并发数在 rsl 表中定义，concurrence 如果是-1，不控制并发数。如果存在一组具有相同 rsl 值的 LBM，这种方式可以实现按组控制并发数。
	Check concurrence by Priority	limitpriorityconcurrence	根据 LBM 的 priority 控制并发数开关。值为 yes 或 no。当值为 yes 时，将按照 LBM 的 priotiry 数目控制并发，并发数在 priority 表中定义，concurrence 如果是-1，不控制并发数。如果存在一组具有相同 priority 值的 LBM，这种方式可以实现按组控制并发数，这是对按 rsl 分组控制的一种补充。
Limit of definition items	Max LBM	maxspditem	Program 表有效行数，这个参数用于控制 KCBP 初始化时预分配内存数目。
	Max User	maxuser	User 表有效行数，这个参数用于控制 KCBP 初始化时预分配内存数目。
	Max Error	maxerror	Error 表有效行数，这个参数用于控制 KCBP 初始化时预分配内存数目。
	Max publish	maxpublish	Pulish 表有效行数，这个参数用

			于控制 KCBP 初始化时预分配内存数目。
	Max subscribe	maxsubscribe	Subscribe 表有效行数，这个参数用于控制 KCBP 初始化时预分配内存数目。
	CWA size	cwasize	CWA 内存长度，单位字节。缺省 4096。CWA 内存用于 AS 进程间共享数据。
Miscellaneous	Node ID	node	Node identifier, 节点编号，全局唯一，取值范围 1-99999
	Node Name	name	节点名称，辅助信息。和节点编号一起显示在状态栏的右下角
	User exit string	userexit	允许使用的用户出口列表。用户出口部分参见 KCBP 程序员手册扩展编程。
	Prompt when shutdown	shutdownprompt	KCBP 关闭时是否提示。值为 yes 或 no。当为 yes 时，关闭 KCBP 时会弹出一个确认框，这种方式可以防止用户误操作关闭 KCBP。
	Collect service statistics	cleanuptimeout	统计开关。值为 yes 或 no。当值为 yes 时，KCBP 系统统计服务调用信息，这些信息可以在 KCBP->View->服务统计窗口中查看。
	Report status to KCMM	usekcmm	KCMM 监控开关。值为 yes 或 no。当值为 yes 时，KCBP 向 KCMM 发送运行信息，这时 KCBP 主动被 KCMM 监控。
	License	license	KCBP 授权号。
	isctimeout	isctimeout	跨系统通讯超时设置，单位秒。
	Prompt when shutdown	shutdownprompt	关闭提示，yes 或 no
	Startup timeout	startuptimeout	启动超时时间，单位秒
	Cleanup timeout	cleanuptimeout	关闭等待时间，单位秒
	Warning level	warninglevel	警告信息的日志级别，缺省 102
		xa2phase	是否启用 2 阶段提交，yes 或 no，缺省 no
		tranlogpagesize	交易日志文件页面大小，单位 K

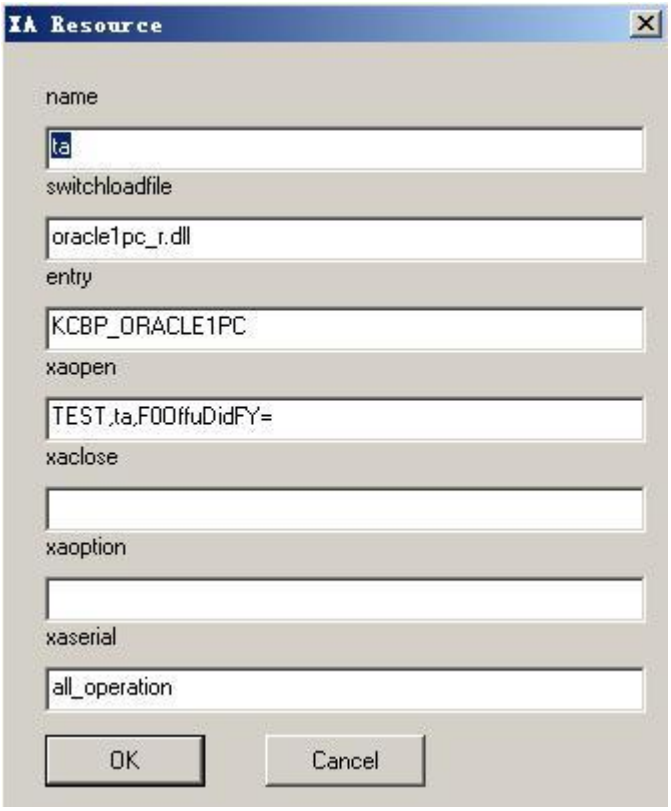
提示： KCBP 系统在启动时，对于未配置的属性，会给出警告信息（以粉红色表示），并自动使用缺省值。
注：如果需要对 Limit of definition items 相关参数动态调整，需要直接编辑配置文件进行设置。
AS 进程自动伸缩算法简要说明：
自动增长条件：当 AS 进程最小数小于最大数，并且在过去的 Overloadtime 时间内忙于出处理业务的 AS 进程数占当前总 AS 进程数的比例高于 Overloadfactor，并且 CPU 利用率小于 maxcpuload 及内存利用率小于 Maxmemoryload，则增加一个 AS 进程。
自动收缩条件：当前 AS 进程数大于最小ＡＳ数，并且，在过去的 Maxidletime 时间内，忙于工作的ＡＳ进程数小于最小 AS 数，则收缩ＡＳ进程个数到最小ＡＳ数。

6.2.3.2XA Resource 属性页

XA 表定义各种资源，包括数据库资源、通讯队列资源等。XA 表存放在 KCBPConf.xml 中，节点名称为 xa。LBM 通过 XA 接口访问数据库、通讯服务器、其他 KCBP 节点等资源。



XA 表的属性如下图：



属性说明如下：

类别	界面名称	说明
XA Resource	Name	XA 名称，同一节点内名称不能重复。
	switchloadfile	XA 接口动态库路径
	entry	XA 入口函数名称
	xaopen	xaopen 参数
	Xaclose	xaclose 参数
	xaoption	xaoption 参数
	xaserial	xaserial 参数，保留

6.2.3.2.1 ODBCXA 配置

ODBCXA 提供通过 ODBC 方法操作数据库。

界面名称	配置说明
name	XA 名称，同一节点内名称不能重复
switchloadfile	odbc1pc.dll
entry	KCBP_ODBC1PC
xaopen	参数格式：DSN,DBNAME,DBUSER,CipherPassword,ConnectName,option 如：spb-run,master,sa,AO5R0uytxn0=,, 生成密码的方法有两种：一是配置界面上密码字段输入 encrypt(DBUser,PlainPassword)；二是 kcbpcp 命令行输入 encrypt 命令，key 输入用户名。 各参数意义是：

	DSN ODBCDSN 名 DBNAME 数据库名 DBUSER 数据库用户名 CipherPassword 数据库加密密码 ConnectName 连接名，可为空 option 选项，支持 commit:0 不自动提供事务，commit:1 自动提交事务，缺省值
xaclose	无
xaoption	cachetable=yes,trancheck=all,isolationlevel=ReadCommitted cachetable 是否缓存指定的后台表，需要授权支持。缺省否。 trancheck 查询是否有事务时，是否执行 SQL 语句查询一下。all 时总是执行 SQL 语句查询。其它值时内部事务计数大于 0 时才执行 SQL 语句查询。缺省是””. isolationlevel 设置事务隔离级别,无值时不设置。
xaserial	all_operation

例子：

<xa entry="KCBP_ODBC1PC" name="trade" switchloadfile="odbc1pc.dll" xaclose="" xaopen="spb-run,master,jz30user,jPI4ybkZtf0Nm7rjKLM19w==,," xaoption="cachetable=yes" xaserial="all_operation"/>

6.2.3.2.2 DB21PC 配置

界面名称	配置说明
name	XA 名称，同一节点内名称不能重复
switchloadfile	db21pc.dll，db21pc_m.dll 带_m 的可以提供多数据库连接
entry	KCBP_DB21PC
xaopen	参数格式： DBNAME,DBUSER,CipherPassword 如：kgdbas,kgdb,vyf+5MISU3s= DBNAME 实例名 DBUSER 用户名 CipherPassword 加密后的密码
xaclose	无
xaoption	无
xaserial	all_operation

例子：

<xa name="security" xaopen="kgdbas,kgdb,vyf+5MISU3s=" entry="KCBP_DB21PC" xaserial="all_operation" xaoption="" xaclose="" switchloadfile="db21pc.dll"/>

6.2.3.2.3 ORACLE1PC 配置

界面名称	配置说明
name	XA 名称，同一节点内名称不能重复
switchloadfile	Oracle1pc.dll 或 oracle1pc_m.dll
entry	KCBP_ORACLE1PC
xaopen	参数格式：TNSNAME,DBUSER,CipherPassword 如：TEST,ta,F0OffuDidFY= TNSNAME 实例名 DBUSER 用户名 CipherPassword 加密后的密码
xaclose	无
xaoption	无
xaserial	all_operation

例子：

<xa entry="KCBP_ORACLE1PC" name="ta" switchloadfile="oracle1pc_r.dll" xaclose="" xaopen="TEST,ta,F0OffuDidFY=" xaoption="" xaserial="all_operation"/>

6.2.3.2.4 KCBPXA 配置

KCBPXA 用于 KCBP 系统互联，KCBPXA 使用同步通讯模式。

界面名称	配置说明
name	XA 名称，同一节点内名称不能重复
switchloadfile	kcbpxa.dll
entry	KCBP_KCBPXA
xaopen	参数格式： qmgrname,sendqname:receiveqname,userid,cipherpassword,proxy,ssl 其中 qmgrname 代表 KCXP 队列管理器名称,可以是 kcxpapi.ini 中的队列管理器，也可以是 ipaddress:port 格式。 proxy 代表代理服务器地址，采用 URL 格式，如果不通过代理服务器连接，该项空白。 ssl 表示 ssl 参数，如果不使用 ssl，该项填空。 格式 1 的 ssl 和 proxy 选项在 kcxpapi.ini 中配置。 如： kcxpzb,req1:ans1,, 192.168.40.65:21000,req1:ans1,KCXP00, GV2gODkBbGg=,,
xaclose	无
xaoption	以下选项参数，可以在初始化串中使用，也可以在运行中调用 setoption/getoption 使用。在初始化若不设置，使用缺省的值。 参数格式： sendtime=30,receivetime=30,lifetime=30,timeout=30,crypt=1,compress=2,can

	cel=1,msgid=,groupid=,retryinterval=3,reopen,ignorecorrelid=1/0,clearserial=1/0 解释： sendtime,receivetime,lifetime,timeout 这四个参数对应同一个值，用在发送超时，接收超时，消息生命周期，调用超时。单位秒，缺省 120 秒。 crypt 加密方式，缺省 0，不加密。请参考 KCBP 配置中的加密方式说明。 compress 压缩方式，缺省 0，不压缩。请参考 KCBP 配置中的压缩方式说明。 cancel 在运行中使用，若值不等于 0，则关闭 KCXP 连接。 msgid 在运行中使用，设置 MSGID。 groupid 设置 groupid retryinterval 设置断线重试间隔，保留 reopen 在运行中使用，重新连接 KCXP。 ignorecorrelid 忽略 correlid，1 忽略，0 不忽略。缺省不忽略。 clearserial 清除请求包里的 SERIAL，1 清除，0 不清除。缺省不清除。
xaserial	All_operation

例子：

<xa entry="KCBP_KCBPXA" name="kcbp2" switchloadfile="kcbpxa.dll" xaclose="" xaopen=" kcxpzb,req1:ans1,," xaoption="" xaserial="all_operation"/>

6.2.3.2.5 KCBP1PCASYNC 配置

KCBP1PCASYNC 用于 KCBP 系统间互联。与 KCBPXA 不同，KCBP1PCASYNC 使用了异步通讯模式。异步通讯模式的运用，使 KCBP 在处理存在 KCBP 系统间通讯的 LBM 时，可以采用单线程多任务的方式调度 LBM。所谓单线程多任务，指的是一个 KCBPAS 线程（或进程），可以同时加载、调度多个 LBM 并发运行。单线程多任务是 KCBP 的专有技术。在这项技术的基础上，可以采用 KCBP 多层集群，建设复杂系统。只支持 KCBP/Win32 系统。

界面名称	配置说明
Name	XA 名称，同一节点内名称不能重复
Switchloadfile	KCBP1PCAsync.dll
Entry	KCBP_KCBP1PCASYNC
Xaopen	参数格式： qmgrname,sendqname:receiveqname,userid,cipherpassword,proxy,ssl 其中 qmgrname 代表 KCXP 队列管理器名称,可以是 kcxpapi.ini 中的队列管理器，也可以是 ipaddress:port 格式。 proxy 代表代理服务器地址，采用 URL 格式，如果不通过代理服务器连接，该项空白； ssl 表示 ssl 参数，如果不使用 ssl，该项填空。 格式 1 的 ssl 和 proxy 选项在 kcxpapi.ini 中配置。 如： kcxpzb,req1:ans1,,

	192.168.40.65:21000,req1:ans1,KCXP00, GV2gODkBbGg=,,
Xaclose	无
Xaoption	选项参数。在初始化中若不设置，使用缺省的值。 参数格式： timeout=30,crypt=1,compress=2,cancel=1,msgid=,groupid=,retryinterval=3 解释： timeout 用在发送超时，接收超时，消息生命周期，调用超时。单位秒，缺省 120 秒。 crypt 加密方式，缺省 0，不加密。请参考 KCBP 配置中的加密方式说明。 compress 压缩方式，缺省 0，不压缩。请参考 KCBP 配置中的压缩方式说明。 cancel 在运行中使用，若值不等于 0，则关闭 KCXP 连接。 msgid 在运行中使用，设置 MSGID。 groupid 设置 groupid retryinterval 设置断线重试间隔，单位秒。
Xaserial	All_operation

例子：

<xa entry="KCBP_KCBP1PCASYNC" name="kcbp2" switchloadfile="KCBP1PCAsync.dll" xaclose="" xaopen=" kcxpzb,req1:ans1,," xaoption="" xaserial="all_operation"/>

6.2.3.2.6 KCXPXA 配置

KCXPXA 为 LBM 其它系统互联提供 KCXP 通讯队列。

界面名称	配置说明
Name	XA 名称，同一节点内名称不能重复
Switchloadfile	kcxpxa.dll
Entry	KCBP_KCXPXA
Xaopen	参数格式包括两种： format1: qmgrname,qname format2: ip:port, qname,userid,cipherpassword 其中 qmgrname 代表 KCXP 队列管理器名称,qname 代表队列名称； 目前版本的 KCXPXA 格式 1 支持 proxy 和 sslproxy，格式 2 不支持 proxy 和 ssl。 proxy 代表代理服务器地址，采用 URL 格式，如果不通过代理服务器连接，该项空白； ssl 表示 ssl 参数，如果不使用 ssl，该项填空。 格式 1 的 ssl 和 proxy 选项参考在 kcxpapi.ini 中的配置。 如： kcxp1,q1 192.168.40.65:21000,q1,KCXP00, GV2gODkBbGg=
Xaclose	无

Xaoption	参数格式： sendtime=30,receivetime=30,lifetime=30,timeout=30,putmode=0 解释： sendtime 发送超时，单位秒缺省 20 秒。 receivetime 接收超时，单位秒。缺省 0 秒。 lifetime 消息生命周期，单位秒，缺省 10800 秒。 putmode put 方式，0 时使用 KCXP_Put，2 时使用 KCXP_Put2。缺省 0.
Xaserial	All_operation

例子：

```
<xa entry="KCBP_KCXPXA" name="shagoffer" switchloadfile="kcxpxa.dll" xaclose=""
   xaopen="192.168.70.31:21000,shagoffer,KCXP00,GV2gODkBbGg="  xaoption=""
   xaserial="all_operation"/>

<xa entry="KCBP_KCXPXA" name="szoffer" switchloadfile="kcxpxa.dll" xaclose=""
   xaopen="192.168.70.31:21000,szoffer,KCXP00,GV2gODkBbGg="  xaoption=""
   xaserial="all_operation"/>
```

6.2.3.2.7 MQXA 配置

MQXA 为 LBM 其它系统互联提供 MQ 通讯队列。MQXA 与 KCXPXA 类似。

界面名称	配置说明
Name	XA 名称，同一节点内名称不能重复
Switchloadfile	Mqxa.dll
Entry	KCBP_MQXA
Xaopen	参数格式：qmgrname,qname 如 MQGR1,q1
Xaclose	无
Xaoption	无
Xaserial	All_operation

例子：

```
<xa entry="KCBP_MQXA" name="shagoffer" switchloadfile="mqxa.dll" xaclose=""
   xaopen=" MQGR1,q1" xaoption="" xaserial="all_operation"/>
```

6.2.3.2.8 CICSXA 配置

CICSXA 用于 KCBP 与 CICS 互联，这种方式连接时，CICS 服务程序需要采用 lbmapi 规范编写。KCBP 提供了一个 lbmapicics.dll 的动态库，可用于编写 CICS 服务程序。

界面名称	配置说明

Name	XA 名称，同一节点内名称不能重复
Switchloadfile	Cicsxa.dll，在 KCBP\bin 目录下
Entry	KCBP_CICSXA
Xaopen	ServerName, UserId, CipherPassword
Xaclose	无
Xaoption	无
Xaserial	All_operation

例子：

<xa entry="KCBP_CICSXA" name="CICS" switchloadfile="cicsxa.dll" xaclose="" xaopen="CICS01,KCXP00,GV2gODkBbGg=" xaoption="" xaserial="all_operation"/>

6.2.3.2.9 TUXEDOXA 配置

TUXEDOXA 用于 KCBP 与 TUXEDO 互联，这种方式连接时，TUXEDO 服务程序需要采用 lbmapi 规范编写，KCBP 提供了一个 lbmapitudexo.dll 的动态库，可用于编写 TUXEDO 服务程序。

界面名称	配置说明
Name	XA 名称，同一节点内名称不能重复
Switchloadfile	Tuxedoxa.dll，在 KCBP\bin 目录下
Entry	KCBP_TUXEDOXA
Xaopen	无（可以根据客户需要调整程序）
Xaclose	无
Xaoption	无
Xaserial	All_operation

例子：

<xa entry="KCBP_TUXEDOXA" name="TUXEDO" switchloadfile="tuxedoxa.dll" xaclose="" xaopen="" xaoption="" xaserial="all_operation"/>

6.2.3.2.10 KDMID1PCFactory 配置

KDMID1PCFactory 用于 KCBP 与第三方中间件互连，包括 KCBP 之间的互联，可在 XML 中配置两种中间件报文转换关系。

界面名称	配置说明
name	XA 名称，同一节点内名称不能重复
switchloadfile	KDMID1PCFactory.dll
entry	KCBP_KDMID_FACTORY

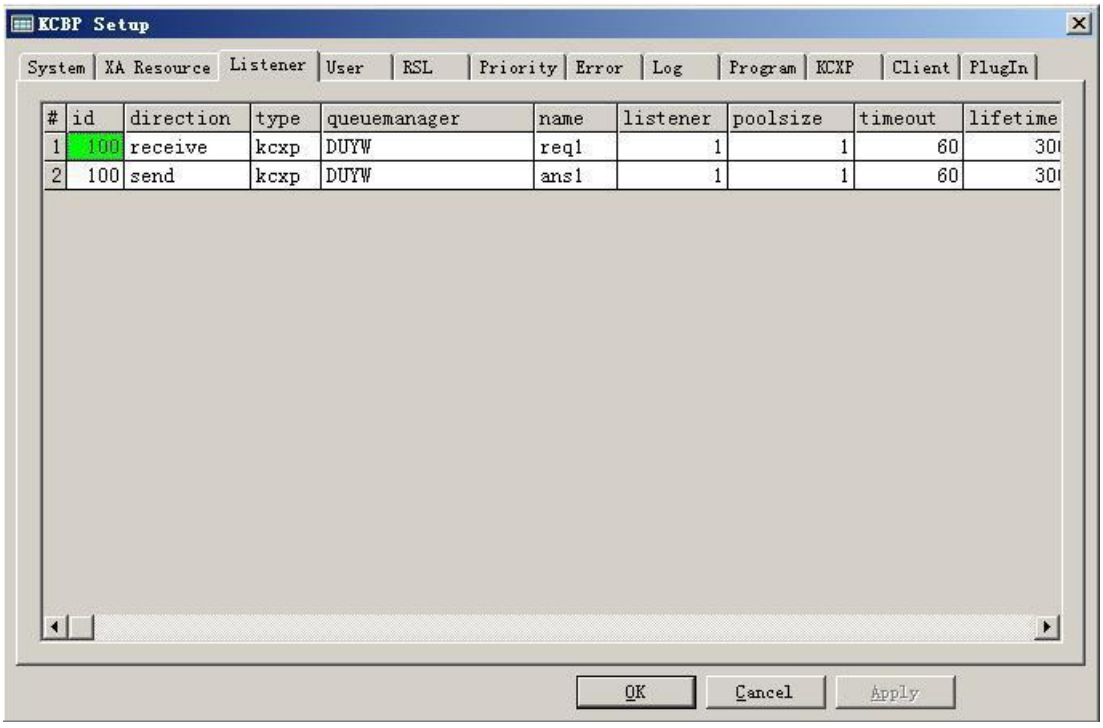
xaopen	格式：ip,port,adapter,rule,param1,param2,... 如：ip=127.0.0.1,port=21000,adapter=KingdomWrapper.dll,rule=testwin.xml,connecttimeout=30,userid=KCXP00,password=GV2gODkBbGg=,sendqueue=req2,receivequeue=ans2,timeout=300 解释： ● adapter 对应不同中间件的通讯适配器；rule 转换规则文件名；其它参数通讯适配器用。 ●
xaclose	无
xaoption	无
xaserial	All_operation

例子：

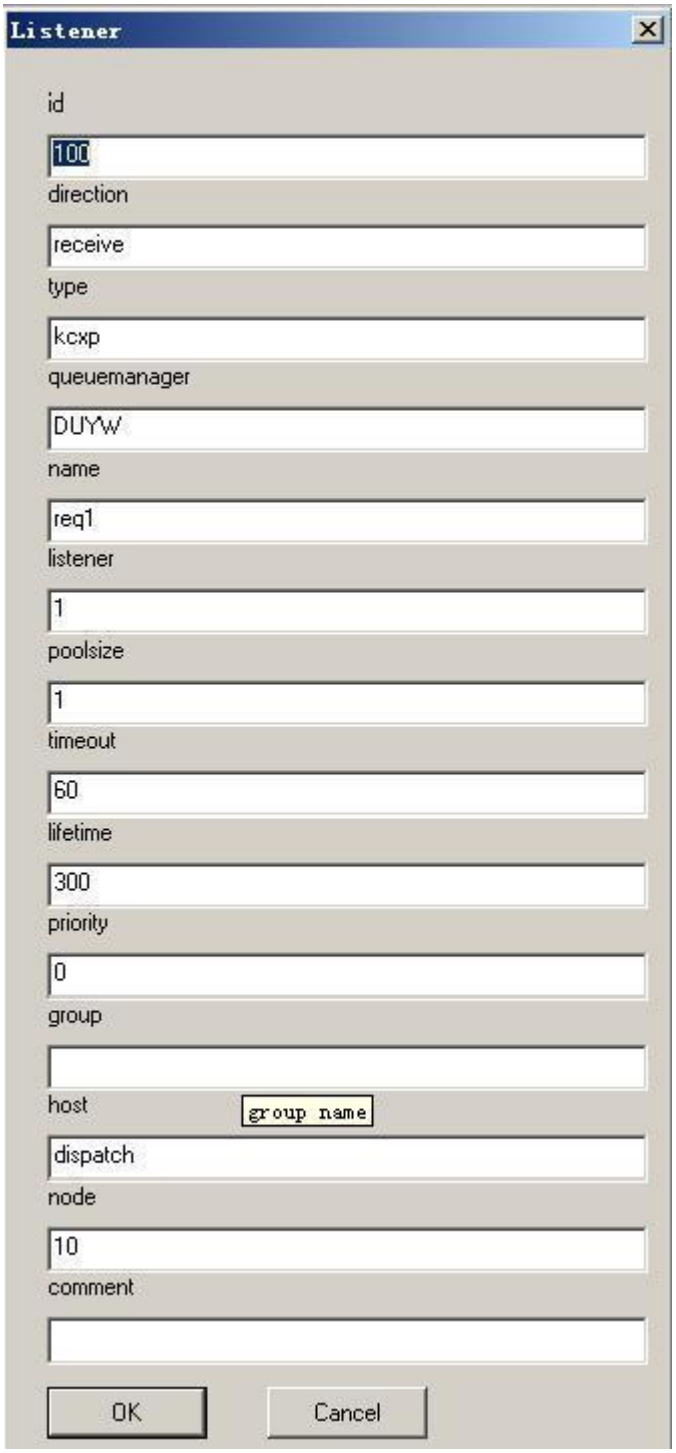
<xa entry="KCBP_KDMID_FACTORY" name="kdmid" switchloadfile="kdmid1pcfactory.dll" xaclose="" xaopen="127.0.0.1:28946" xaoption="userid=9999,password=3Mc+w9uU5lM=,srcnodeid=0000,destnodeid=0000,opcode=3" xaserial="all_operation"/>

6.2.3.3Listener 属性页

Listener 表用于存放通讯队列，通讯队列可以配置多组，一组队列包括一个请求接收队列和一个应答发送队列，两个队列具有相同的编号。Listener 表在 KCBPConf.xml 中定义，节点名称为 externalqueue。



队列的属性如图所示：



界面名称	配置说明
id	队列编号，同组队列编号相同。
Direction	队列方向，有两个有效值：receive 和 send。Receive 表示队列用于接收请求，send 表示队列用于发送应答。一组队列包括一个请求队列和一个应答队列。
type	队列类型，有效值：kcxp。
queuemanager	队列管理器名称，支持名称及 IP:PORT 两种格式
name	队列名称

listener	队列侦听线程数目，当 direction 为 receive 时有意义。
poolsize	队列连接池预建连接数目，缺省值是 1。在 KCBP/Windows 版和 KCBP/LINUX3.0 版本该数目填 1 即可。在 KCBP/LINUX2.0 版本上，该值可大于等于 1。
timeout	队列操作超时时间，单位秒，缺省 120。这个超时时间是 KCXP_Put 的超时时间，不是消息的生命周期。
lifetime	队列中消息的生命周期，单位秒，缺省 120。
priority	保留。
group	保留。
host	dispatch
node	保留
comment	备注，可以填入用户的辅助说明信息。

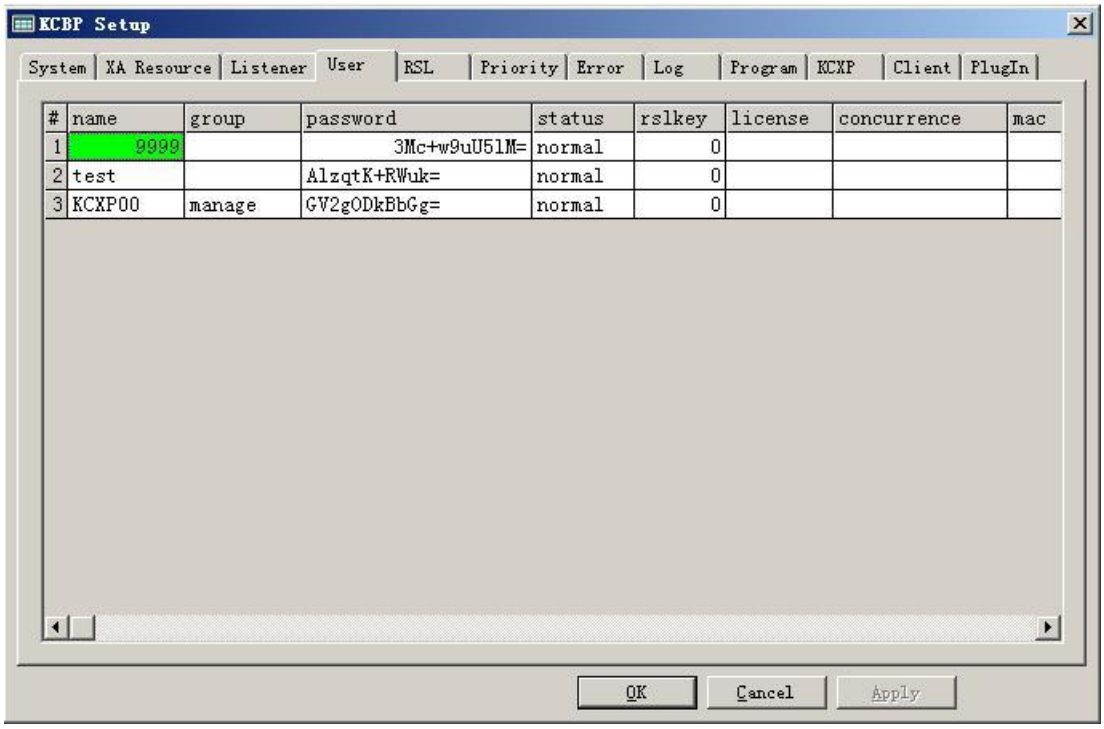
例子：

<externalqueue comment="" direction="receive" group="" host="dispatch" id="100" lifetime="300" listener="1" name="req1" node="10" poolsize="1" priority="0" queuemanager="DUYW" timeout="60" type="kcxp"/>

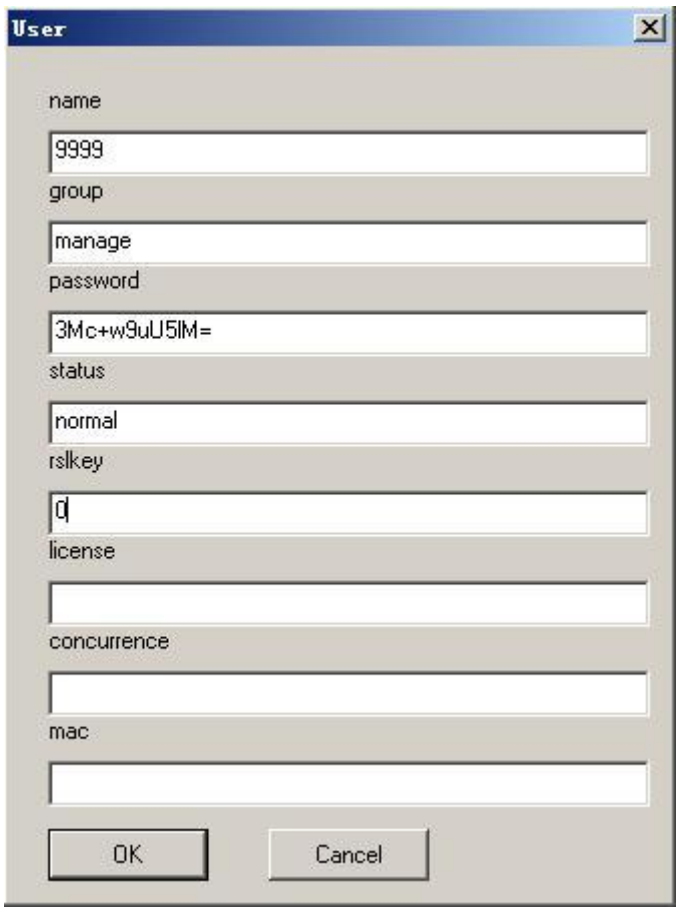
<externalqueue comment="" direction="send" group="" host="dispatch" id="100" lifetime="300" listener="1" name="ans1" node="10" poolsize="1" priority="normal" queuemanager="DUYW" timeout="60" type="kcxp"/>

6.2.3.4User 属性页

User 表用于存放 KCBP 的用户定义信息。Listener 表在 KCBPUsr.xml 中定义，节点名称为 user。



User 属性如下图所示：



界面名称	配置说明
Name	用户名称

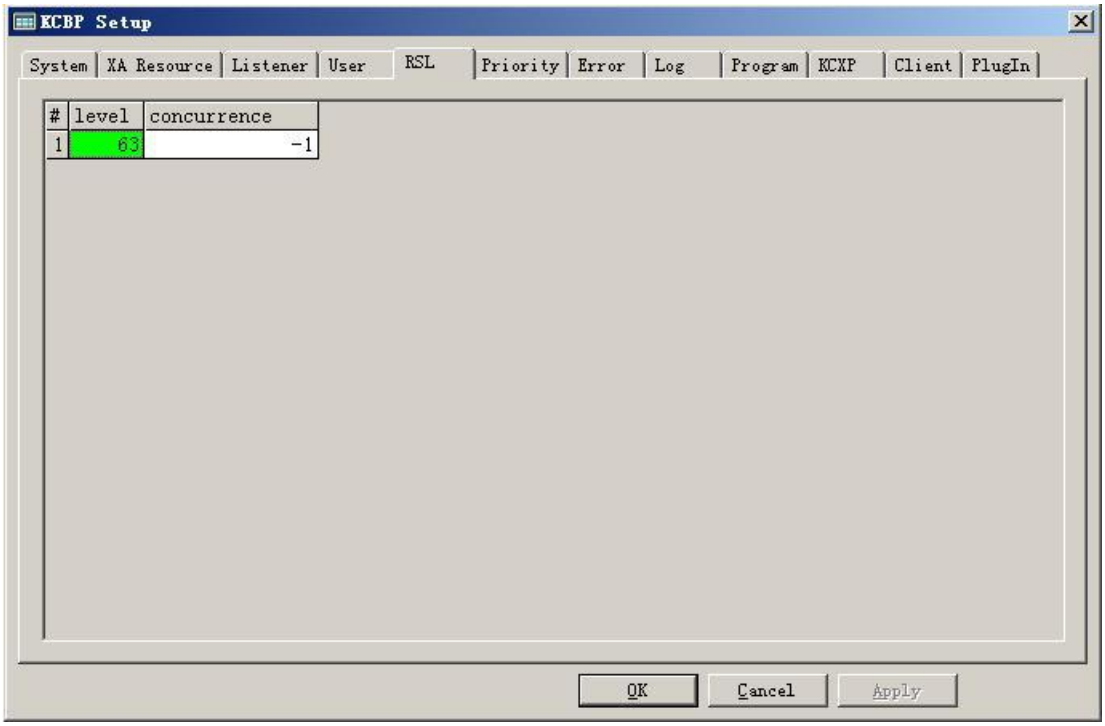
Group	组，可以是 manage 或 application。其中，属于 manage 组的用户有管理权限及服务程序访问权限，属于 application 组的用户只有服务程序访问权限。
password	密文口令。 生成密码的方法有两种： 1 在配置界面上密码字段输入 encrypt(User,PlainPassword)； 2 kcbpcp 命令行输入 encrypt 命令，key 输入用户名。
Status	用户状态，保留。
rslkey	用户的钥匙编号表，取值范围 0~63。编号之间用逗号分隔。 如：1,2,35
license	保留
concurrence	保留
mac	保留

例子：

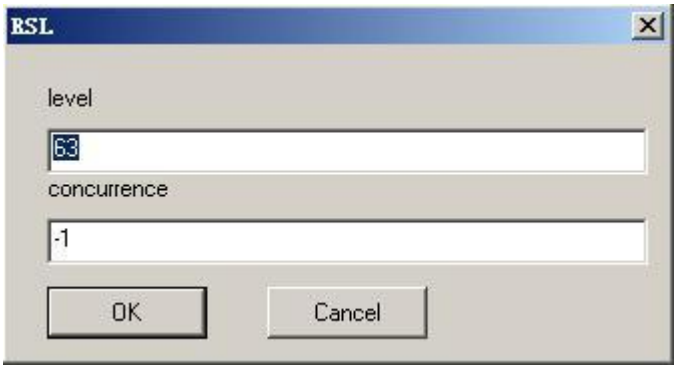
<user concurrence="" group="" license="" mac="" name="test" password="AlzqtK+RWuk=" rslkey="0" status="normal"/>

6.2.3.5RSL 属性页

RSL 表用来定义属于同一 rsl 程序的并发数，RSL 表在 KCBPRSL.xml 中定义，节点名称为 rsl。



RSL 属性如下图所示：



RSL 属性说明如下：

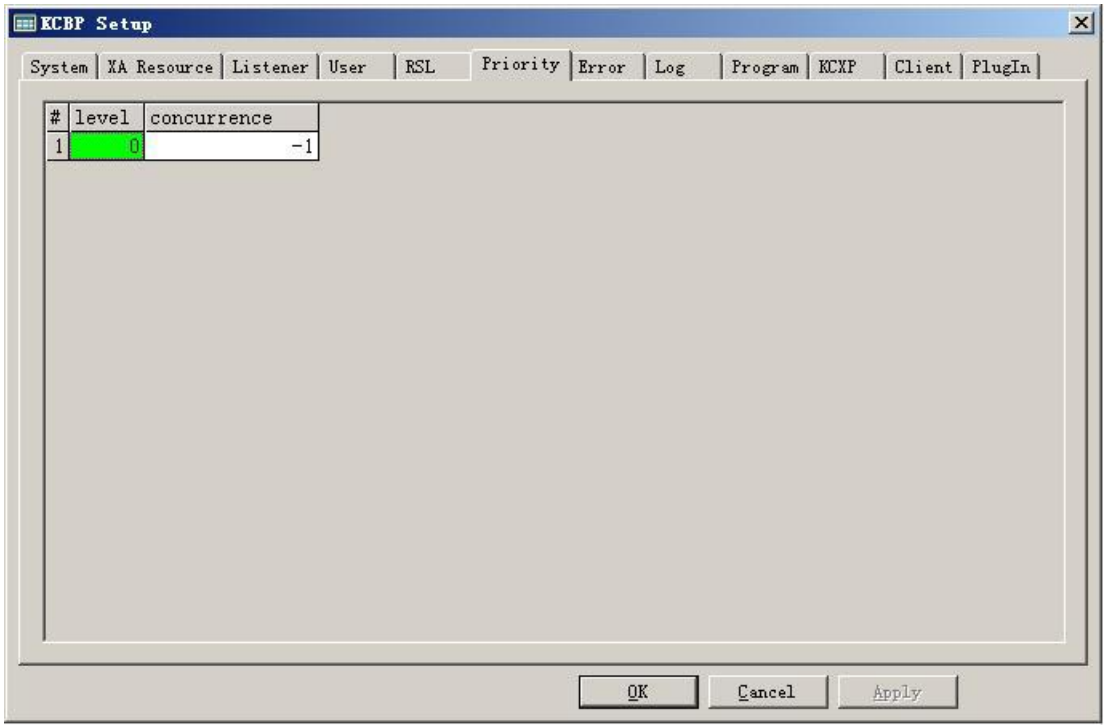
界面名称	配置说明
Level	级别，取值范围：0-63。
Concurrence	并发数，-1 表示不限制并发。

例子：

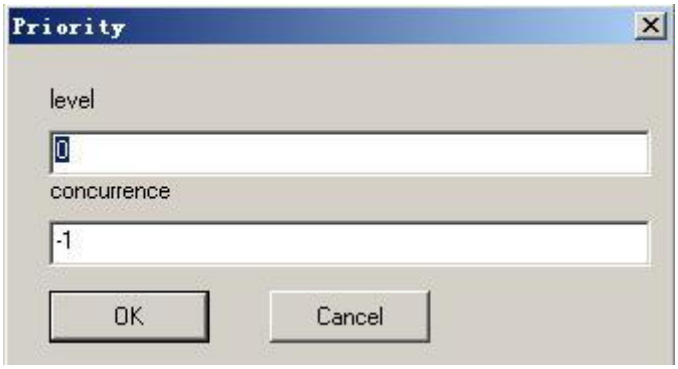
<rsl concurrence="5" level="63"/>

6.2.3.6Priority 属性页

Priority 表用来定义拥有相同 priority 的程序并发数，priority 表在 KCBPPrio.xml 中定义，节点名称为 priority。



Priority 属性如下图所示：



Priority 属性说明如下：

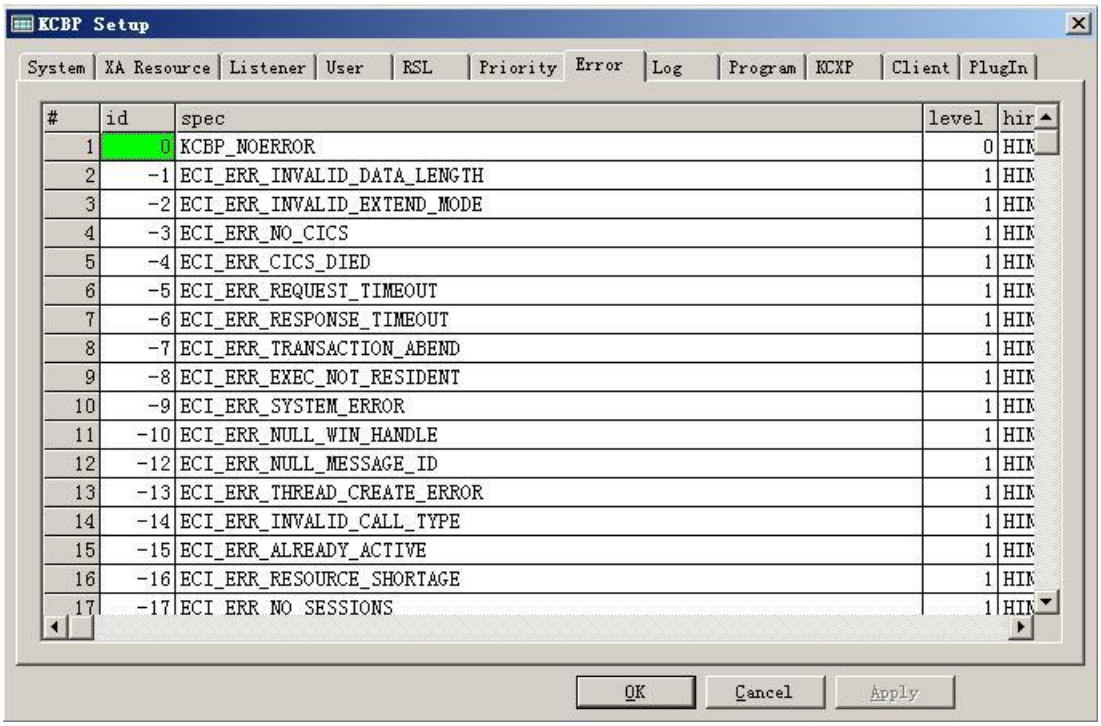
界面名称	配置说明
Level	级别，取值范围：0-63。
Concurrence	并发数，-1 表示不限制并发。

例子：

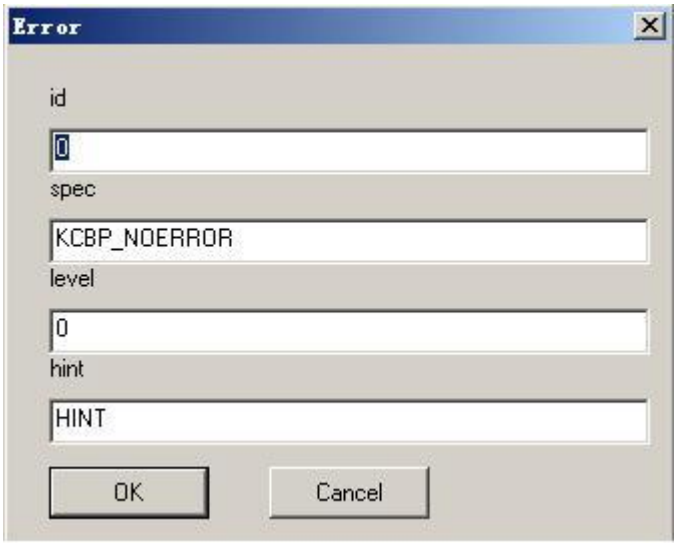
<priority concurrence="6" level="63"/>

6.2.3.7Error 属性页

Error 表中存放了 KCBP 的常用错误编码和错误信息，这个表位于 KCBPError.xml 中。一般来讲，用户不必修改这个表。



Error 属性如下图所示：



Error 属性说明如下：

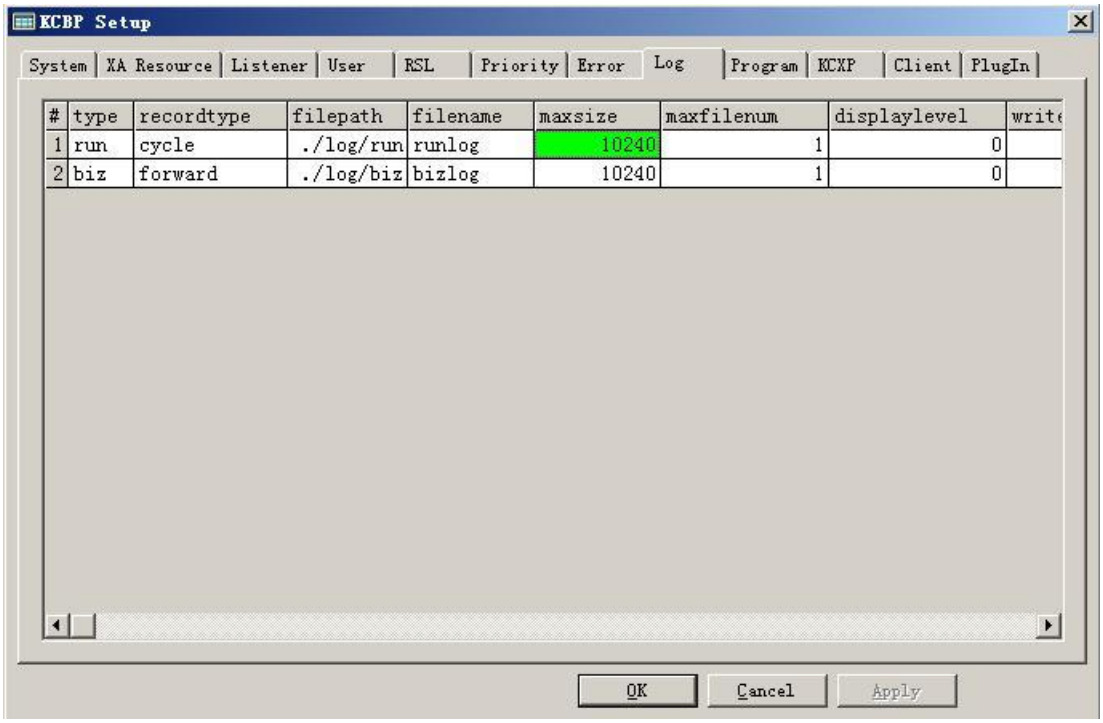
界面名称	配置说明
Id	错误编号
Spec	错误说明
Level	错误级别
hint	错误处理提示

例子：

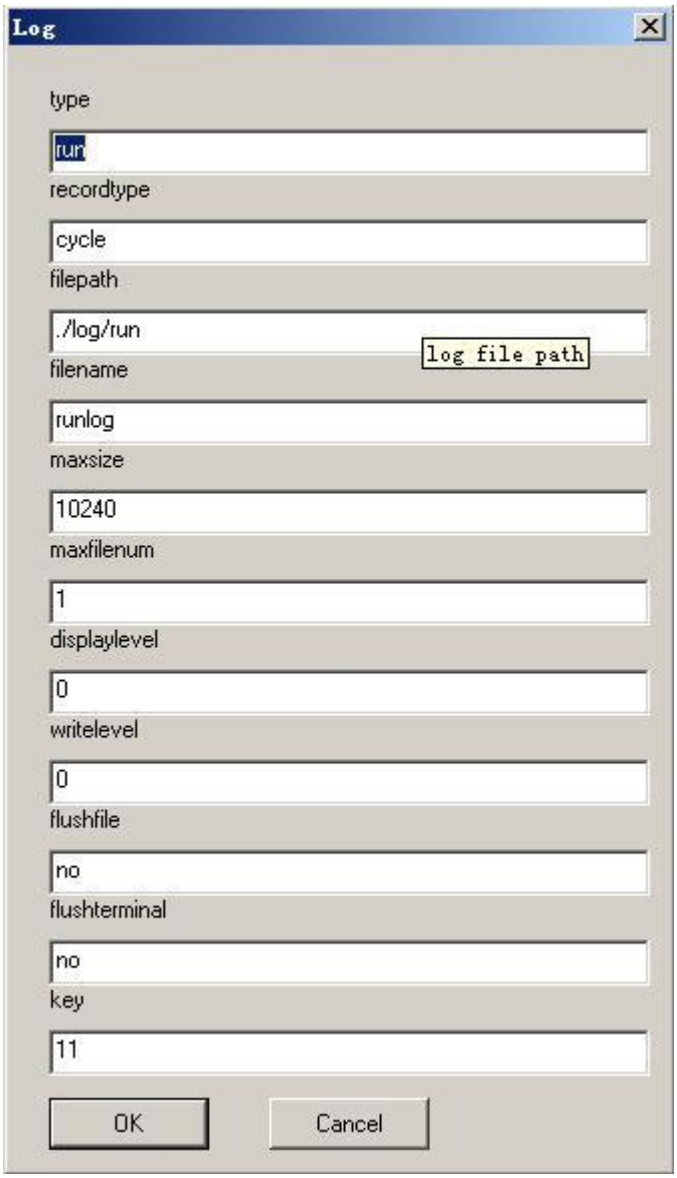
<error id="1002" spec="KCBP_ERROR_UNDEFINED_LBM" level="1"
hint="HINT" />

6.2.3.8Log 属性页

Log 表定义了 KCBP 系统日志相关信息。KCBP 的日志包括运行日志和事务日志两种，运行日志记录 KCBP 系统的运行信息，事务日志记录 KCBP 系统的分布式事务处理信息，目前版本的 KCBP 未启用事务日志。这节的内容是对运行日志的说明。


KCBP Setup
System
XA Resource
Listener
User
RSL
Priority
Error
Log
Program
KCXP
Client
PlugIn
type recordtype filepath filename maxsize maxfilenum displaylevel write
1 run cycle ./log/run runlog 10240 1 0
2 biz forward ./log/biz bizlog 10240 1 0
OK
Cancel
Apply

Log 属性如下图所示：



Log 属性说明如下：

界面名称	配置说明
type	类型，运行日志类型为 run，事务日志类型为 biz
recordtype	日志文件记录方式，可以是 cycle 或 forward。cycle 是循环记录，当日志文件长度达到 maxsize 后，从下个日志文件中开始写，当文件个数到达 maxfilenum，从第一个日志文件开始写，这时会覆盖以前记录的日志文件内容。forward 是线性记录，只在一个日志文件写，此时 maxsize 不起作用。
filepath	日志文件路径。KCBP 启动时自动在这个路径下创建以日期为名称的目录，并在日期目录下存放日志文件。
filename	日志文件名称，这是一个名称前缀。KCBP 的日志文件名称包括前缀和编号两部分。
maxsize	cycle 类型的日志文件大小，单位 KB
maxfilenum	cycle 类型的日志文件数目
displaylevel	显示级别，控制屏幕显示的级别。日志级别范围是：0-99 详细运行信息，

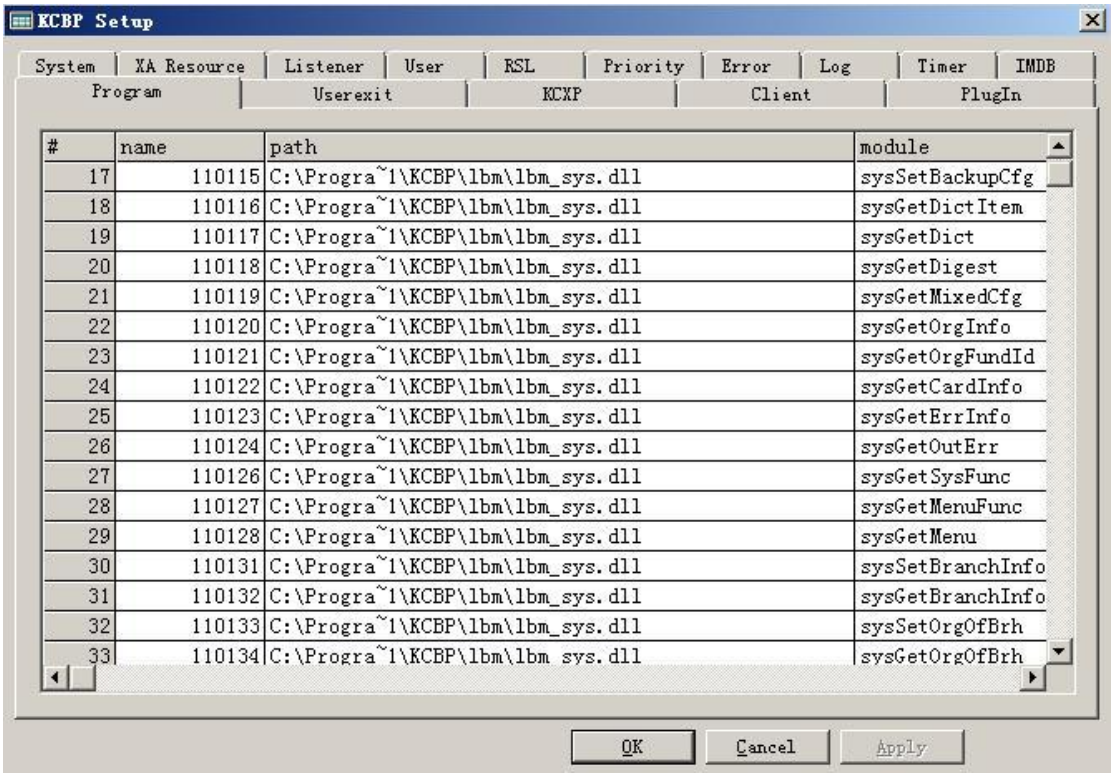
	100-999 警告信息，1000-9999 错误信息，10000 LBM 调试信息、系统启动、停止信息。 开发调试程序时，一般设置 displaylevel=0，可以显示详尽的运行信息。 生产环境上一般 displaylevel=1000，这时只显示错误信息。 性能测试时也要设置 displaylevel=1000，显示信息越少，KCBP 系统处理性能越高。 LBMAPI 的 KCBP_PrintStatus 显示级别为 10000。 在 KCBP/LINUX 信息不分颜色，KCBP/WIN 采用不同颜色区分信息： 绿色：LBM 调试信息和系统启动、停止信息 红色：错误信息 粉红色：警告信息 黑色：运行信息
writelevel	写日志级别，控制日志文件记录的级别。注意事项同 displaylevel。
flushfile	写文件模式，可以是 yes 或 no。yes 代表不使用文件写缓存，每条日志立即写文件，no 表示使用文件缓存，当缓冲区写满之后写文件。当日志内容较多时，建议使用 no，这样可以降低磁盘操作频率，提高效率。Flushfile 的控制对象是文件。
flushterminal	显示模式，KCBP/WIN 上无意义，KCBP/LINUX 上有意义。flushterminal 和 flushfile 类似，只不过控制对象是控制台。
key	KCBP/WIN 无意义，KCBP/LINUX 上为日志队列的 keyproj。

例子：

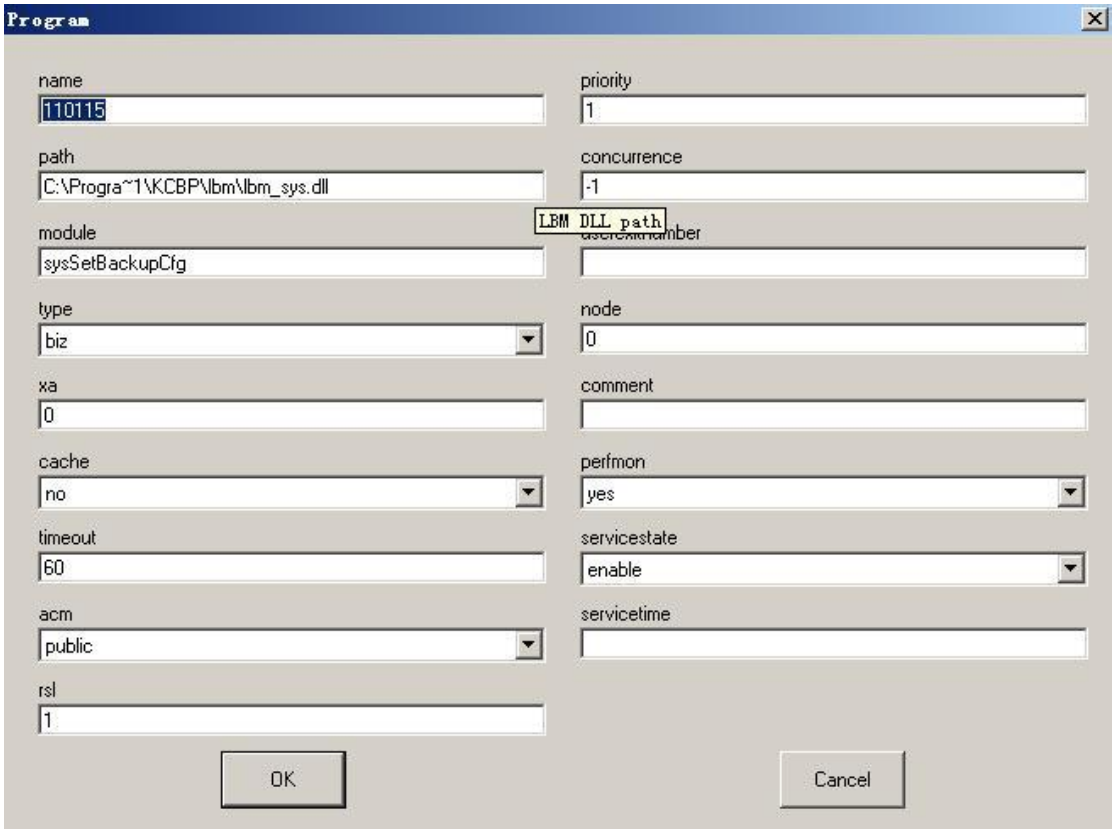
<log displaylevel="0" filename="runlog" filepath="./log/run" flushfile="no" flushterminal="no" key="11" maxfilenum="1" maxsize="10240" recordtype="cycle" type="run" writelevel="0"/>

6.2.3.9Program 属性页

Program 表定义服务程序相关信息。服务程序就是 LBM。这个表存放在 KCBPSPD.xml 中。



Program 属性如下图所示：



Program 属性说明如下：

界面名称	配置说明
Name	LBM 名称，长度不大于 8 个字节，系统内唯一，字母或数字均可，区分

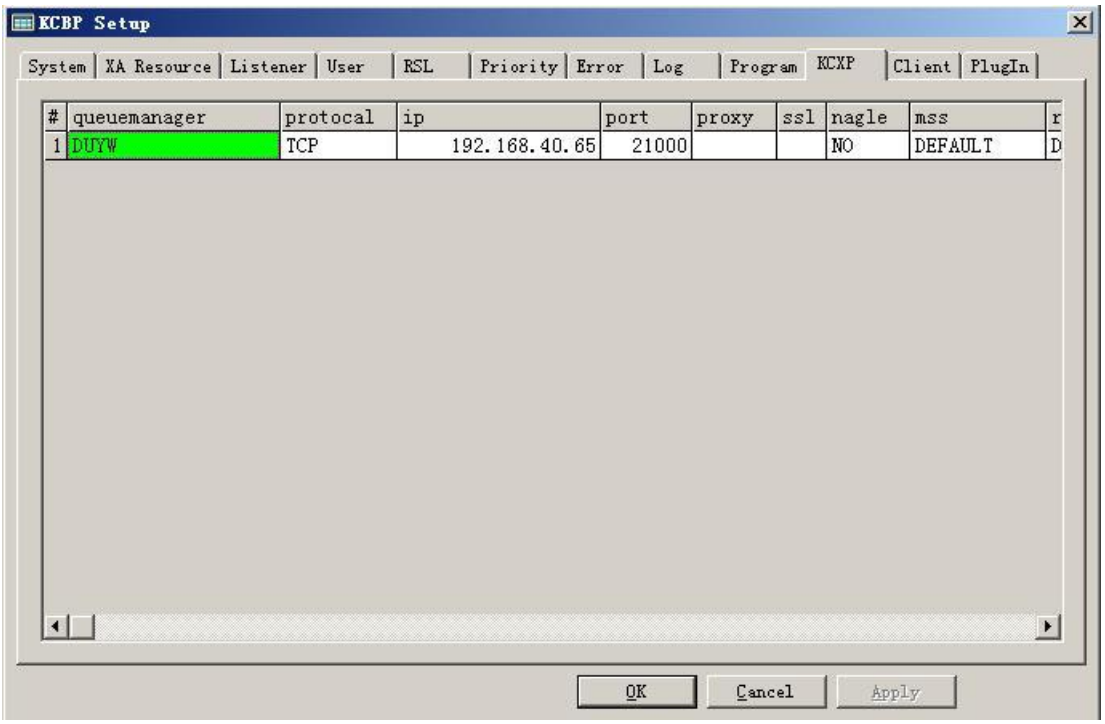
	大小写。
path	LBM 所在动态库路径。
module	LBM 函数入口。
type	LBM 类型,可以是 biz、deputy 或 bpel。biz 是本地服务，调用时会用到 Path 和 module 定义的内容，deputy 是代理其他节点上的服务，调用时不会用本地的 path 和 module 等内容，请求会被转发到 xa 定义的节点上处理。bpel 与 deputy 类似，bpel 转发类业务处理逻辑从 kcbp 内核移到 LBM 中，并且可以支持业务流程，在流程中可应调用本节点定义的 LBM 或其他节点的 LBM。bpel 支持同步和异步的 kcbpxa 跨系统调用协议，但不支持单线程多任务。bpel 类型业务的处理逻辑以 LBM 形式存在，在 kcbpsys.dll 的以 bpel 为入口的 LBM 中。
xa	LBM 转发目的节点名称，与 XA Resource 中的 XA 名称对应。只在 type 为 deputy 时才有效。
cache	LBM 缓存标志，可以是 yes 或 no。yes 表示 LBM 的动态库加载后不释放，常驻内存，no 表示 LBM 动态库调用完成后立即释放，不常驻内存。Cache 为 yes 时会提高系统处理效率，大约有 10-50%的提高。
timeout	请求超时时间，单位秒，缺省 60。这个时间控制的是请求在 KCBP 内部的排队时间，当请求被 KCBP Listener 接收到开始计时的，不包括请求在 KCXP 中的传输和等待时间。
acm	服务程序存取控制方式，是 Access controll method 的简写。可以是 public 和 private。Public 表示 LBM 调用时不检查用户全限，这个 LBM 可以被任何用户调用。Private 表示 LBM 调用时检查用户权限，这个 LBM 只能被有权限的用户调用，权限类别在 rsl 属性中定义，如果 LBM 的 rsl 在 user 表的该用户的 rslkey 列中，该用户就可以调用这个 LBM，否则不能调用。
rsl	资源安全类别，是 resource security level 的简写。取值范围 0-63。通过这个值可以控制 LBM 的访问，也可以对 LBM 分组并进一步控制分组并发数。 有关并发控制的说明显见并发控制使用方法。
priority	优先级，目前只作为一种 LBM 分组方式存在。用法类似 RSL
concurrence	并发数，-1 代表不控制并发数。如果为 0，这个 LBM 将不能被调用。
userexitnumber	用户出口号，详见用户出口使用说明。
node	节点号。
comment	备注信息，用户可以自己填写一些说明信息，如 LBM 的用途等。
perfmon	保留
servicestate	服务状态，值：enable 或 disable
servicetime	服务时间段，格式：[begindayofweek-enddayofweek][begintime-endtime] 例子： [1-5][09:00:00-11:29:59],[1-5][13:00:00-14:59:59] 最多可设置 3 个时间段

例子：

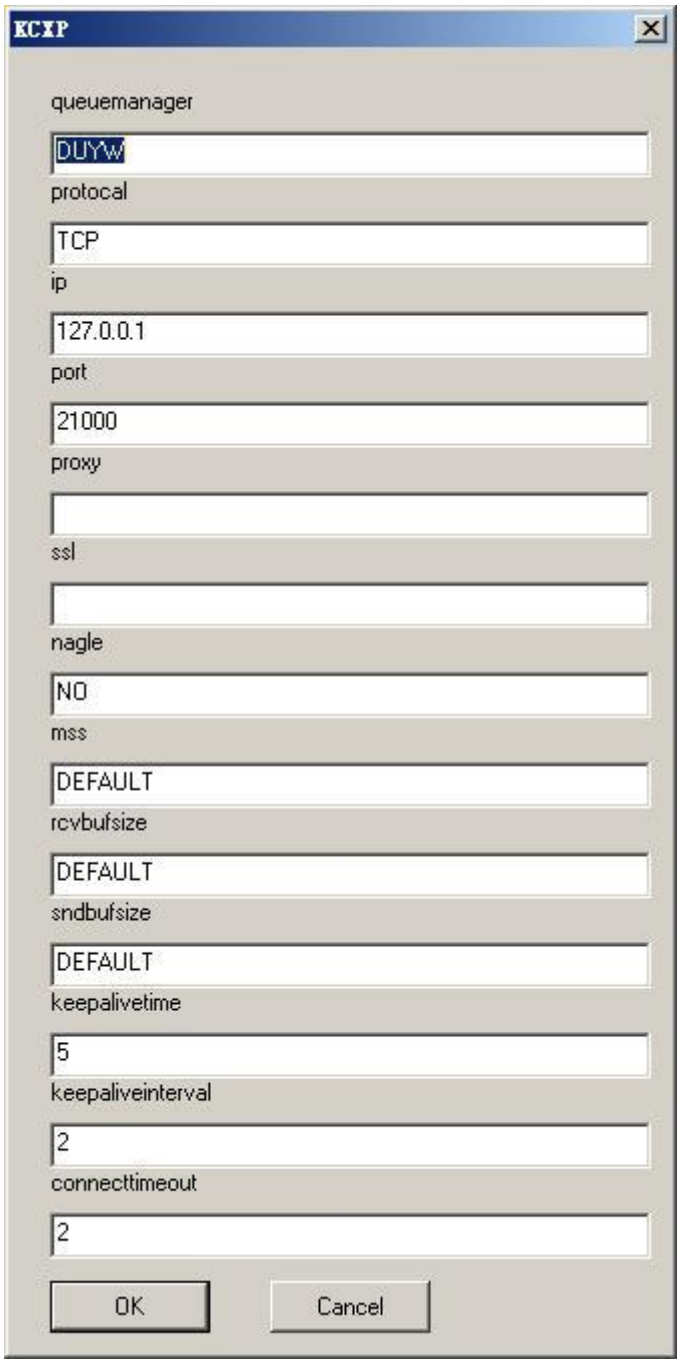
<program acm="public" cache="no" comment="" concurrence="-1" module="GenQryQueryMenu" name="900051" node="0" path="..\lbm\lbm_gen.dll" priority="1" rsl="1" timeout="60" type="biz" userexitnumber="" xa="0"/>

6.2.3.10 KCXP 属性页

KCXP 表用来配置 KCXP 队列管理器连接信息，这个表存放在 kcxpapi.ini 中。



KCXP 属性如下图所示：



KCXP 属性说明如下：

界面名称	配置说明
queuemanager	队列管理器名称，本地名称，和 KCXP 服务器无关。
protocal	协议，缺省为 TCP。一般不用修改。
ip	IP 地址
port	端口号
proxy	代理服务器地址，如果不通过代理服务器连接，空白即可。 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

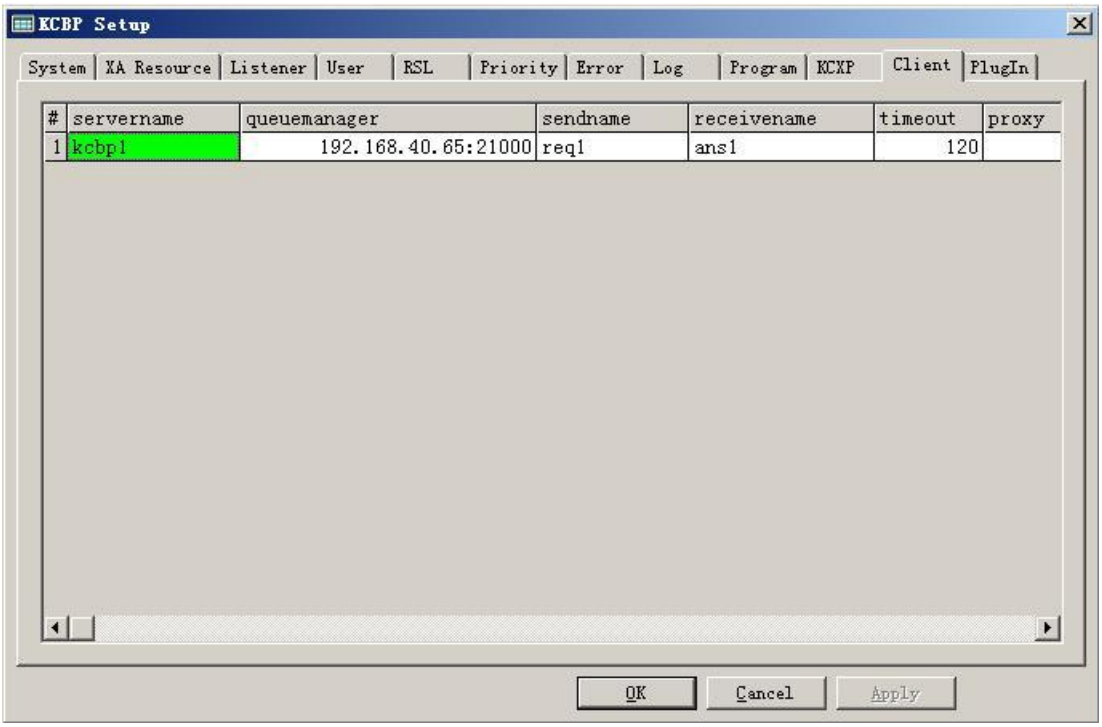
	socks5://guest:password@192.168.40.65:1080。
ssl	SSL 信息，要求和 KCXP 服务端对应。如果服务端不用 SSL，该项空白。 SSL 格式：状态,根证书,证书,密码,算法表,加密传输,主动发起握手，如： YES,ssccCA.pem, client.pem, client, ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH, YES, YES
nagle	TCP 优化算法开关，可以为 yes 或 no，yes 表示打开 nagle 算法，no 表示禁止。 缺省值为 yes。
mss	TCP 的一个参数，一般不用改变。
rcvbufsize	TCP 的一个参数，一般不用改变。
sndbufsize	TCP 的一个参数，一般不用改变。
keepalivetime	TCP 断线检测时间开始，单位秒。缺省 5。
keepaliveinterval	TCP 断线检测时间间隔，单位秒。缺省 2。
connecttimeout	连接超时时间，单位秒，缺省-1，表示采用同步方式建立 socket 连接；设置大于 0 的值表示异步方式建立 socket 连接，等待时间为设定的秒数。对于网络环境复杂并且需要建立很多连接的系统，有必要采用异步方式建立连接，并设置一个合适的超时时间，这样可以避免当有 KCXP 失效时重启动 KCBP 等待时间太长问题。

Kcxpapi.ini 例子：

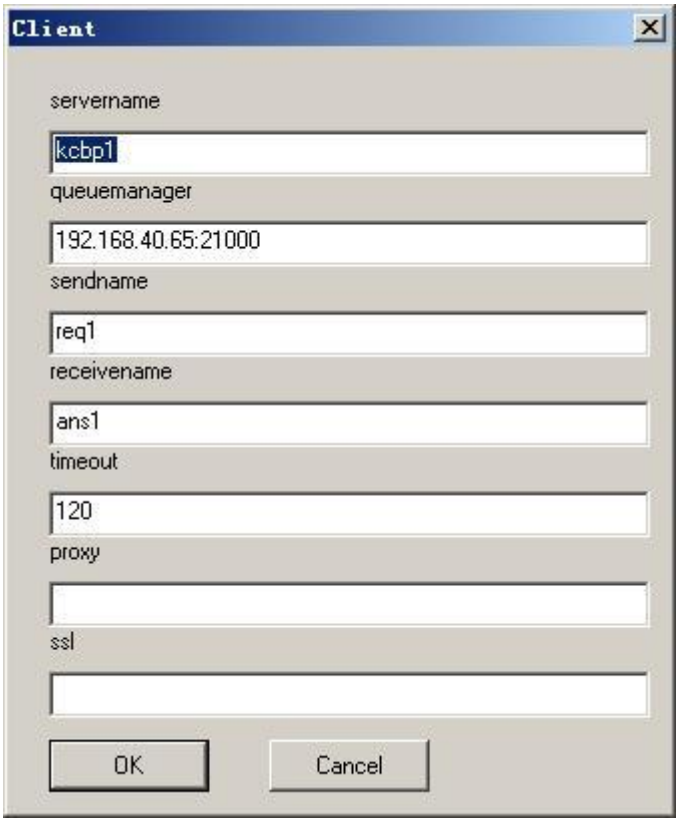
```
[QMgr]
QMgrName=DUYW
QMgrProtocol=TCP
QMgrIp=192.168.40.65
QMgrPort=21000
Proxy=
SSL=
[Option]
Nagle=NO
TcpMss=DEFAULT
TcpRcvBuf=DEFAULT
TcpSndBuf=DEFAULT
TcpKeepLive=5
TcpKeepLiveInv=2
TcpConnectTimeout=20
```

6.2.3.11 Client 属性页

Client 报表用来存放 kcbpcp 使用的配置信息，这个表存放在 KCBPCli.xml 中。



Client 属性如下图所示：



Client 属性说明如下：

界面名称	配置说明
servername	KCBP 连接名称，本地名称，有 KCBP 服务端无关。
queuemanager	KCXP 队列管理器名，格式 1：名称，格式 2：Ip:port 当使用格式 1 时，这个名称是指 KCXPAPI.ini 定义的队列管理器名称。

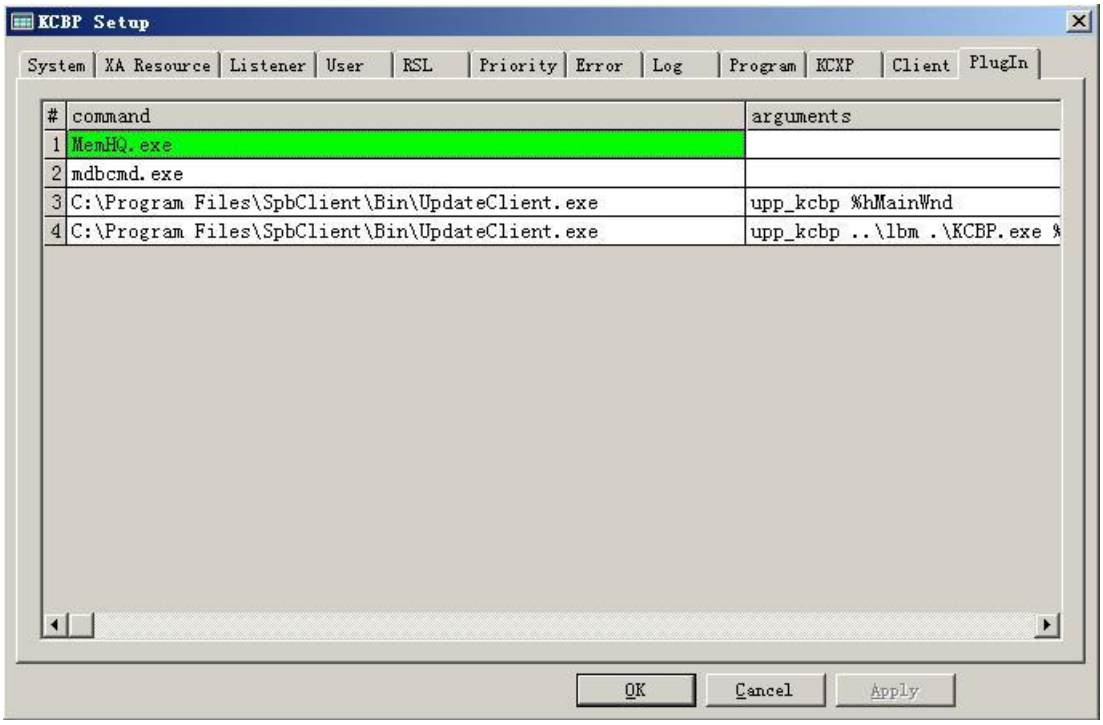
	当使用 ip:port 时，不会用到 KCXPAPI.ini。
sendname	请求队列名称
receivename	应答队列名称
timeout	请求超时时间，单位秒，缺省 120
proxy	代理服务器地址，如果不通过代理服务器连接，空白即可。当 queuemanager 使用 ip:port 格式时才会用到这个参数。如果使用 KCXPAPI.ini，该参数在 kcxpapi.ini 中定义。 PROXY 采用 URL 格式：协议://用户名:密码@服务器:端口，如 https://192.168.40.65:80 socks4://192.168.40.65:1080 socks5://guest:password@192.168.40.65:1080。
ssl	SSL 信息，要求和 KCXP 服务端对应。如果服务端不用 SSL，该项空白。当 queuemanager 使用 ip:port 格式时才会用到这个参数。如果使用 KCXPAPI.ini，该参数在 kcxpapi.ini 中定义。 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

例子：

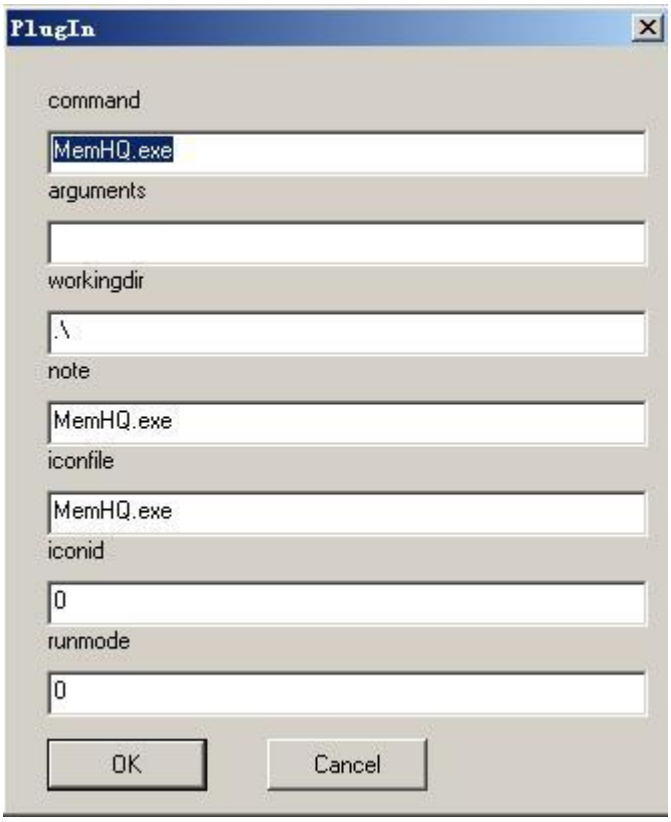
<externalqueue proxy="" queuemanager="192.168.40.65:21000" receivename="ans1" sendname="req1" servername="kcbp1" ssl="" timeout="120"/>

6.2.3.12 Plugin 属性页

KCBPPlugin 用来存放 KCBP 界面上 ToolBar 附加按钮信息，这个表存放在 KCBPPlugIn.XML 中。



KCBPPlugin 的属性如下图所示：



KCBPPlugin 的属性说明如下：

界面名称	配置说明
command	命令行。
arguments	命令参数。
workingdir	工作目录。
Note	tooltip 提示。
Iconfile	图标文件
Iconid	图标编号
Runmode	运行方式

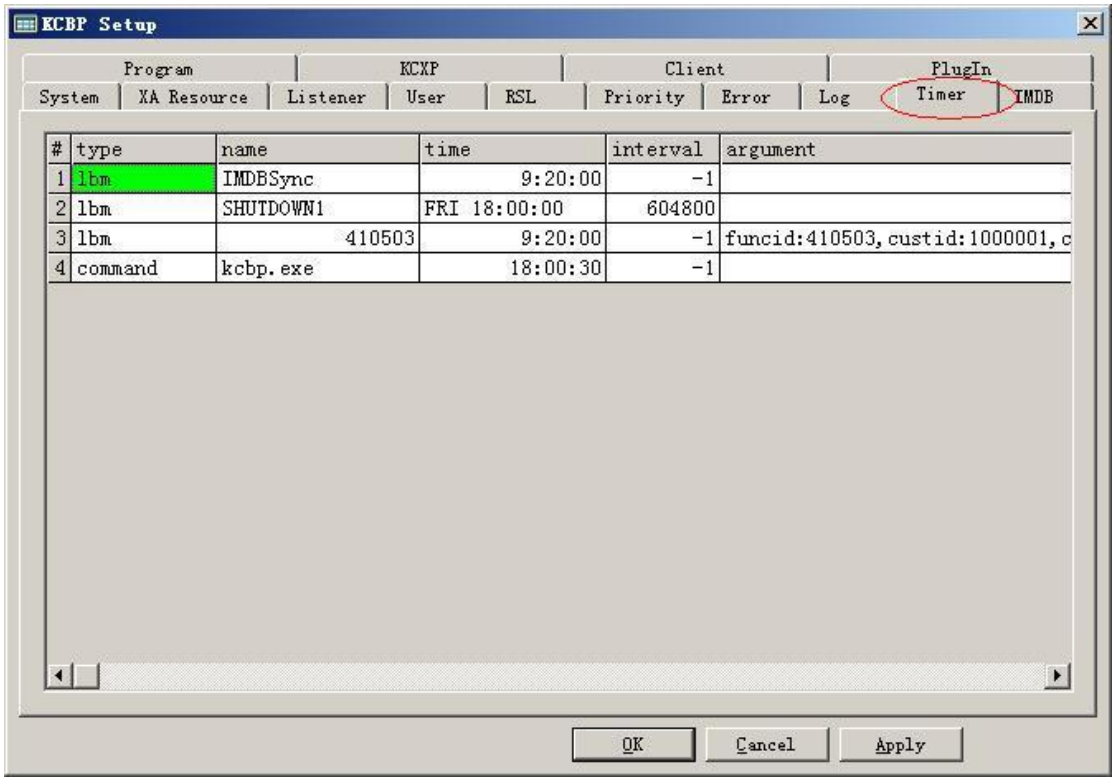
例子：

KCBP 检查到下面的配置时，会在界面的 Toolbar 行增加一个 MemHQ 的图标。

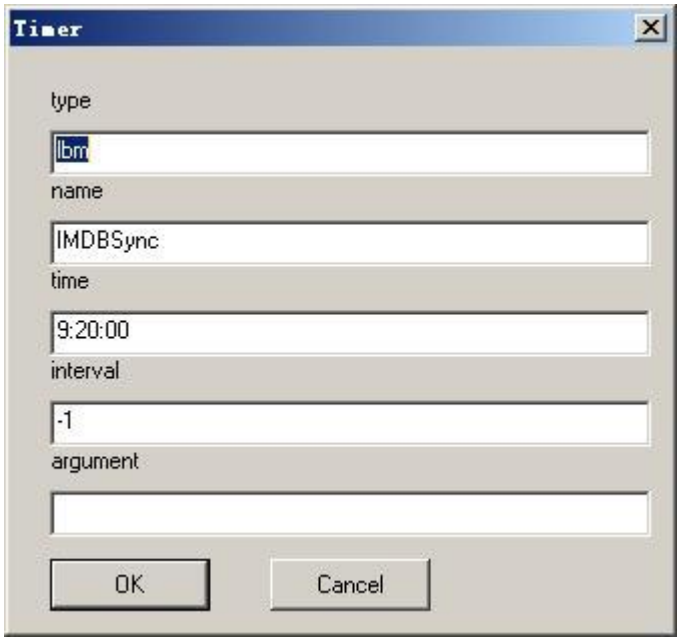
<tool arguments="" command="MemHQ.exe" iconfile="MemHQ.exe" iconid="0" note="MemHQ.exe" runmode="0" workingdir=".\"/>

6.2.3.13 Timer 属性页

TImer 页面用来设置定时调度任务，这个表的内容存放在 KCBPTimer.XML 中。



Timer 的属性如下图所示：



Timer 的属性说明如下：

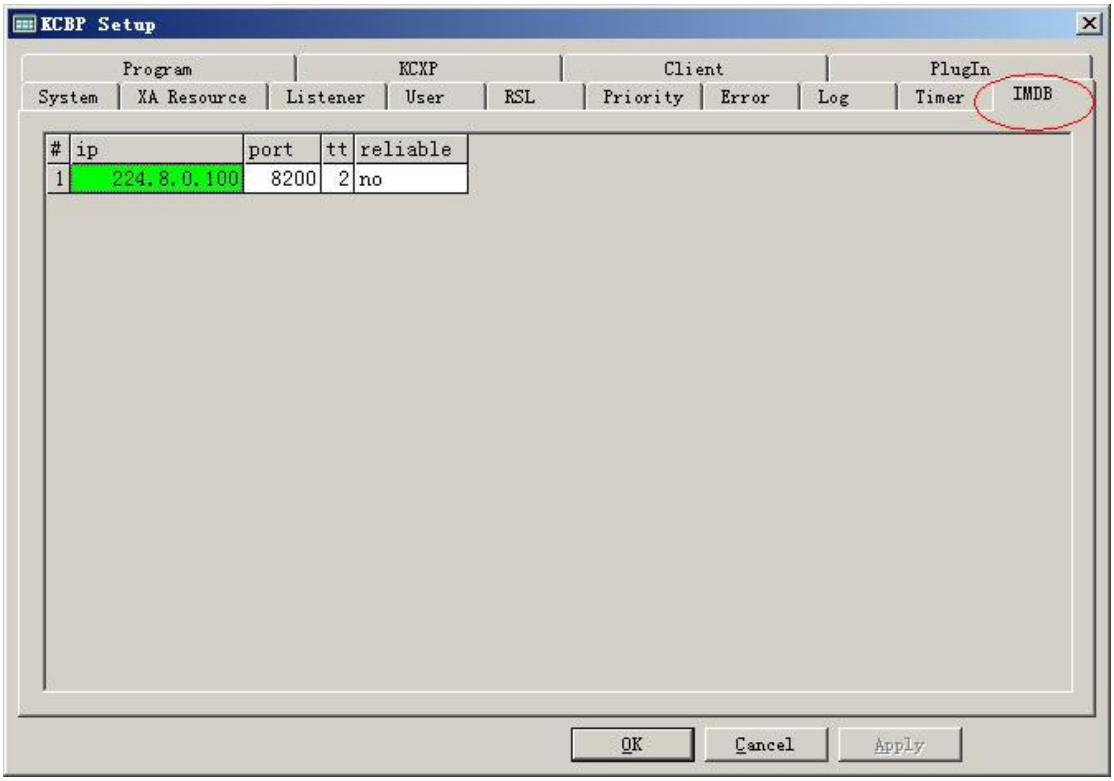
界面名称	配置说明
Type	任务类型，lbm 或 command
Name	LBM 名称或命令名称
Time	调用时间，格式: [YYYY-MM-DD] [SUN|MON|TUE|WED|THU|FRI|SAT] HH:MM:SS
Interval	每次调用时间间隔，单位秒
Argument	LBM 参数或命令

例子：

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<tasklist>
<!-9:20:00 调用一次，以后不再调用，interval 是-1，表示间隔时间无限大/-->
  <task argument="" interval="-1" name="IMDBSync" time="9:20:00" type="lbm"/>
<!--每天 9:20:00 调用一次/-->
  <task argument="" interval="86400" name="LBMStat" time="9:20:00" type="lbm"/>
<!--9:20:00 调用一次 410503 业务/-->
  <task
argument="funcid:410503,custid:1000001,custorgid:1,trdpwd:,netaddr:12345678,orgid:0001,operway:4,ext:0,market:0,fundid:1000001,secuid:,stkcode:,qryflag:0,count:,poststr:0" interval="-1" name="410503" time="9:20:00" type="lbm"/>
<!--18:00:30 关闭 KCBP /-->
  <task argument="-stop" interval="-1" name="kcbp.exe" time="FRI 18:00:30" type="command"/>
</tasklist>
```

6.2.3.14 IMDB 属性页

IMDB 页面用来设置多播服务器通讯参数，这个表只有一行内容，存放在 KCBPConf.XML 中。



IMDB 的属性如下图所示：

IMDB 的属性说明如下：

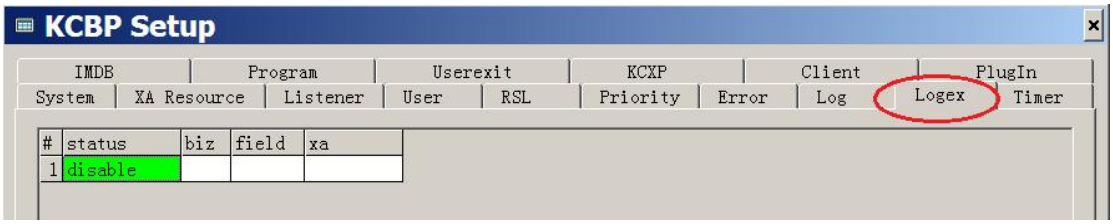
界面名称	配置说明
ip	多播地址，IPV4，取值范围 224.0.0.0 到 239.255.255.255
port	多播端口，取值范围 1-65535。
ttl	多播消息生命周期，值介于 1-16 之间。
reliable	是否使用可靠多播，值 yes 或 no,缺省为 no，推荐 no.

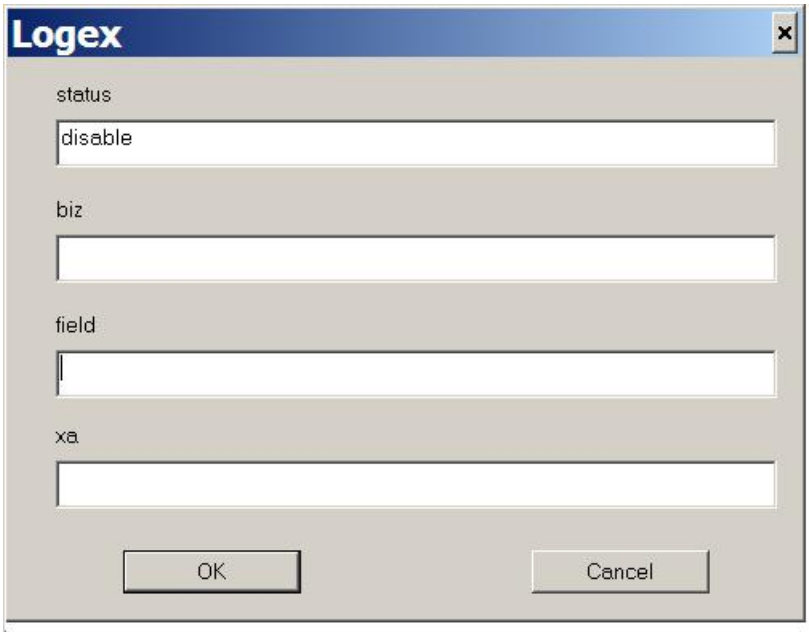
例子：

<imdbserver ip="224.8.0.100" port="8200" reliable="no" ttl="2"/>

说明：一个集群众功能对等的 KCBP 需要配置相同的多播 ip 和 port,节点号不能重复。不同集群的 KCBP 多播 ip 和 port 不能重复。

6.2.3.15 Logex 页面属性





LOGEX 页面用来设置扩展日志属性，用来对日志进行过滤。这个表只有一行，存放在 KCBPConf.xml 里。

LOGEX 的属性说明如下：

界面名称	配置说明
status	日志过滤状态，enable 允许，disable 不允许。
biz	要过滤的 LBM 名字，以逗号分隔。为空时不做按 LBM 名字过滤。
field	要过滤的字段名字，以逗号分隔。为空时不做按字段名过滤。

xa	要过滤的 XA 名字，以逗号分隔。为空时不做按 XA 名字过滤。

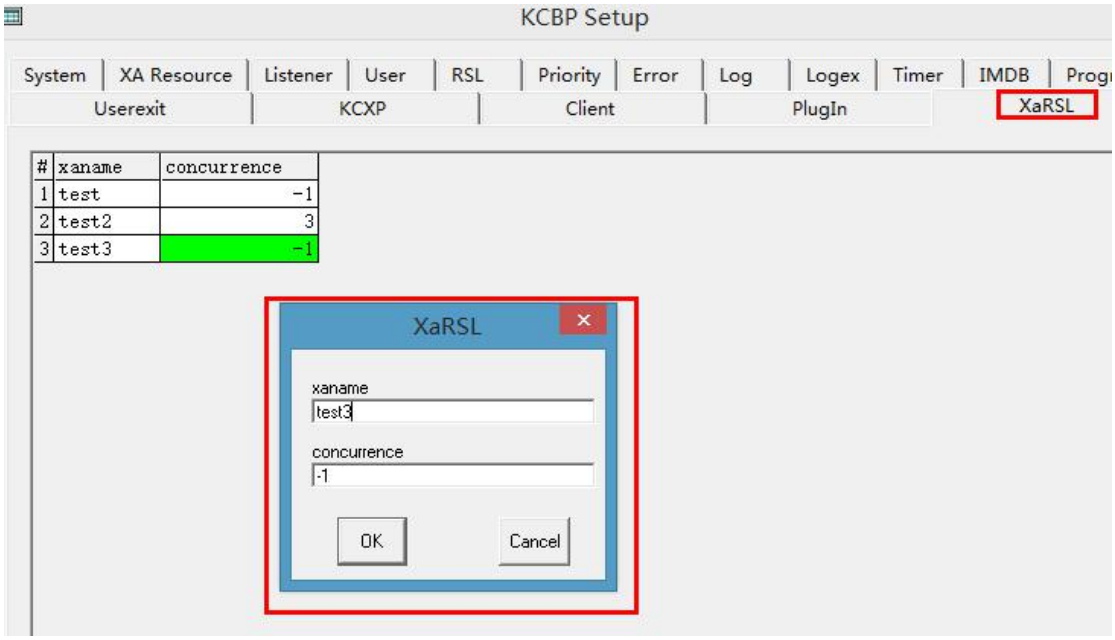
例子：

<logex status="enable" biz="L0101001,L0101002" field="ORGID=0201" xa="kcbp01,kcbp02" />

说明：

- 对于不属于 XA 的日志，不做 XA 过滤。对于 XA 写的日志，除了做 XA 名过滤外，还要做按 LBM 名字过滤，按字段名过滤。
- 过滤通过就写日志，过滤通不过就不写日志。
- logex 和 log 的关系：logex 的日志过滤允许时，log 中的 loglevel 不起作用。但错误日志除外，也就是，即是日志过滤允许，loglevel 大于 2000 的日志仍然是记录的，不管是否满足过滤条件。日志过滤不允许时，即 status=”disable”时，loglevel 起作用。

6.2.3.16 xarsl 页面属性



xarsl 页面用来配置 xa 并发控制信息。这个表的数据存放在 KCBPXARSL.xml 里。xarsl 的属性如下：

界面名称	配置说明
xaname	XA 的名称
concurrence	要控制的并发数，-1 表示不控制并发

例子　<xarsl concurrence="3" xaname="test2"/>

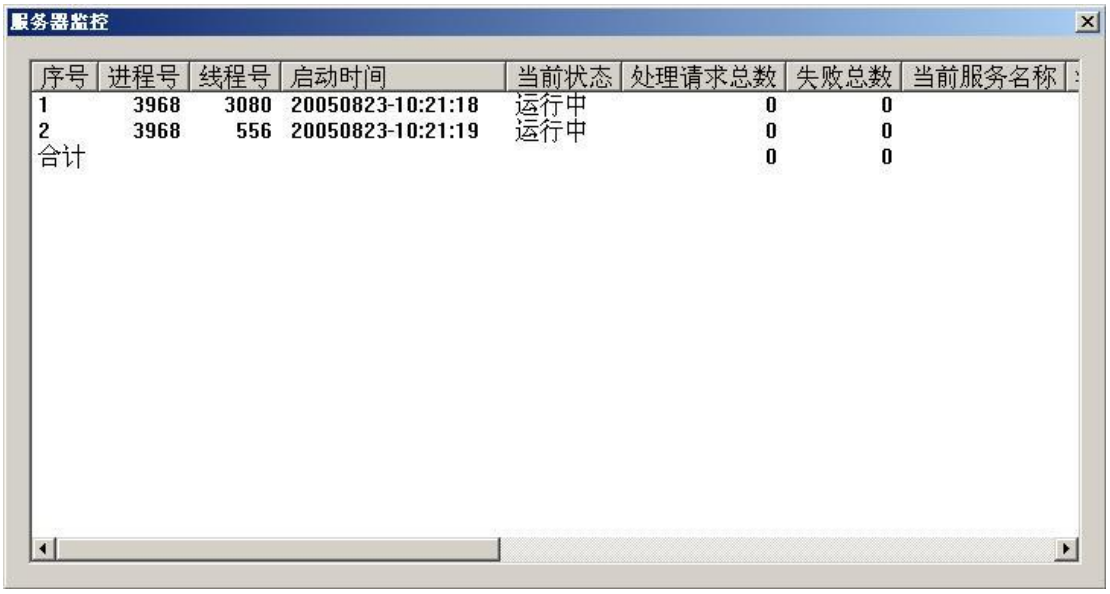
7. KCBP 工具用法

7.1 KCBP 运行状态监控

KCBP 提供了一个服务进程运行状态监控器，用来监控 AS 进程的运行状态。通过监控器，可以查看 KCBP 系统每个 AS 进程的处理效率以及所有 AS 进程的总处理效率，也可以查看正在处理的请求的相关信息。如发现 KCBP 系统响应很慢时，可以查看 AS 进程当前正在处理那些业务，它们是何时开始的，哪个业务处理耗时长等。

KCBP 系统启动后就可以启动运行状态监控器。如下图所示，运行状态监控器可以用 KCBP 的查看->进程状态菜单启动，或按 ToolBar 上的望远镜图标启动。

启动后的运行状态监控器界面如下：



上图中，每个 AS 进程的运行信息显示一行，最后一行显示所有 AS 进程的统计信息。

AS 进程信息说明如下：

列名	说明
序号	AS 进程在 KCBP 系统中的顺序号
进程号	进程号，对于 process 方式运行的 AS，每个 AS 的进程号都不同
线程号	线程号
启动时间	AS 进程启动时间
当前状态	AS 进程状态，可以是未启动、初始化、运行中、结束、未知
处理请求总数	AS 进程启动以来处理请求的总数
失败总数	AS 进程启动以来处理请求的失败总数
当前服务名称	AS 进程正在处理的服务名称（即 LBM 名称）
流水号	正在处理的服务的流水号，这个编号是 KCBP 内部赋予请求的编号
服务开始时间	正在处理的服务的开始时间
请求来源	当前请求来源，请求输入队列的 id，这个 id 在 Listener 中定义
消息号	当前请求的消息号，是 KCXP 赋予请求的消息编号
总处理时间(秒)	AS 进程启动以来处理请求的总时间
平均处理速度(笔/秒)	AS 进程的处理请求的平均速度

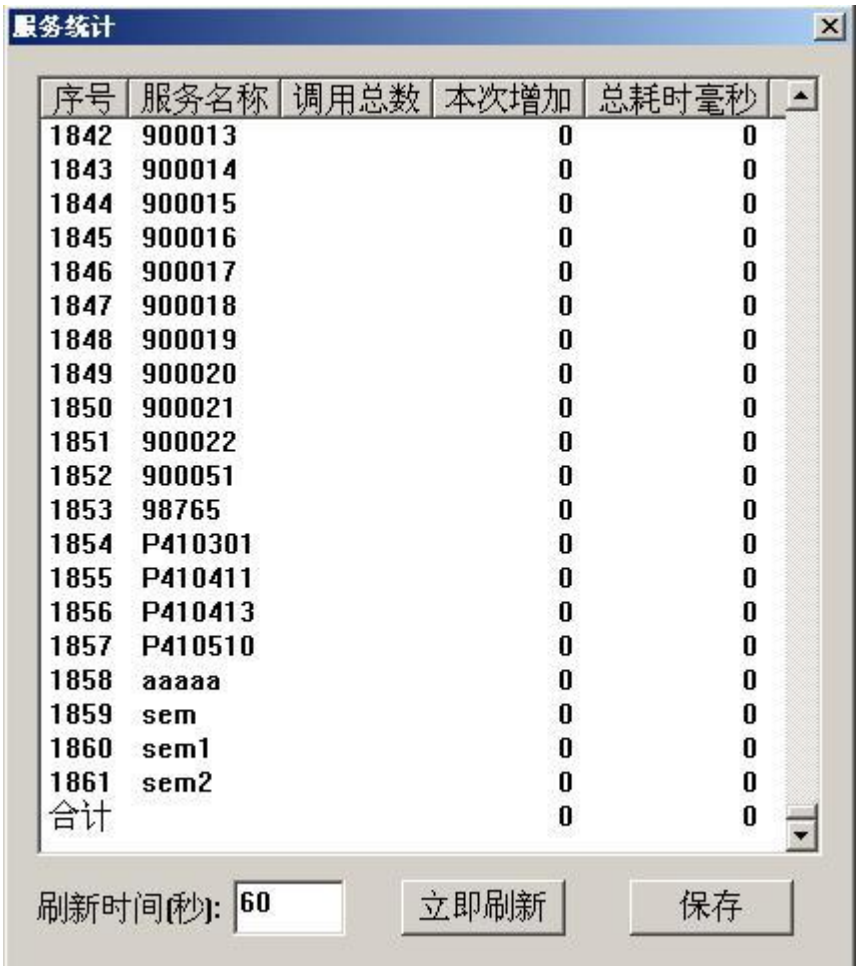
7.2 KCBP 服务统计

KCBP 提供了一个服务统计监控器，用来监控每个服务的统计信息，包括每个 LBM 被调用总次数，总运行时间，单位时间内的调用增加次数等，也包括上述信息的合计。

KCBP 可以设置是打开统计开关，通过图形管理器->system 属性页->collect service statistics 检查框设置。当打开通及开关时，KCBP 系统将统计每个 LBM 的调用信息，系统的处理性可能会有所下降，但一般不会超过 10%。通过图形管理器设置这个参数时，不会立即生效，需要重新启动系统后才生效，而通过 kcbpcp 命令更新这个参数立即生效。

KCBP 系统启动后就可以启动服务统计监控器。如下图所示，服务统计监控器可以用 KCBP 的查看->服务统计菜单启动，或按 ToolBar 上的照相机图标启动。

启动后的服务统计界面如下图：



上图中，刷新时间是 60 秒，如果刷新时间改为 1 秒(输入 1 然后回车)，就可以看到每个业务的处理速度以及系统总处理速度。单击每个列标题栏可重新排序。

7.3 命令行处理器 kcbpcp

本章介绍 KCBP 命令管理器 kcbpcp 的配置和用法。Kcbpcp 是 KCBP 命令行管理工具，是 KCBP command processor 的简写，顾名思义，这个命令处理器就是处理各种命令的，它能完成 KCBP 服务端配置管理、系统运行参数动态调整、LBM 调用等功能。

7.3.1 KCBPCP 的配置

KCBP 命令管理器的配置文件是 kcbpcli.xml。在 kcbpcli.xml 中配置所连接的 KCBP 名称和队列名，如下所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<clientcfg>
<externalqueue servername="kcbp1" queuemanager="192.168.40.75:21000" sendname="req1" receivename="ans1" timeout="120" proxy="" ssl="" />
</clientcfg>
```

说明：如果要连接其他的 KCBP，可在 kcbpcli.xml 文件中增加相应的配置即可。

7.3.2 kcbpcp 用法

可以通过 KCBP 的界面菜单运行命令管理器，或在 kcbp\bin 目录下直接运行 kcbpcp.exe 程序,会出现如下的提示行：
kcbp =>
然后就可以在提示后面输入命令。

7.3.2.1 察看命令帮助

kcbp => ?
屏幕输出：

```
Help message :
    connect              : Connect to the server
    disconnect           : Disconnect to the server
    delete               : Delete config item at server
    encrypt              : encrypt text
    execute              : execute LBM at server
    help                 : Display help infomation of all
    insert               : Insert config item at server
    list tables          : list tables for handling.
    quit                 : Quit this program
    run                  : execute system command
    runsc                : execute command script file
    select               : Retrieve config at server
    set connection       : select connection
    shell                : invoke sh
    start                : start a KCBP node
    stop                 : stop    a KCBP Node
    update               : Update config item at server
    verbose [on|off]     : switch verbose mode
    ? [command]          : Display the help about the command
  For further help: ? kcbp-command      - help for specified command
```

7.3.2.2 连接到 KCBP

这个命令完成和 KCBP 建立连接的动作，也可以验证 KCBP 是否已经正常启动。
命令格式为：kcbp => connect to kcbp1 user KCXP00 using 888888
此处 kcbp1 是在 kcbpcli.xml 文件中定义的所连接的 KCBP 的名称，KCXP00 和 88888 是在 KCBPUsr.xml 文件中定义的用户名和密码。
如果连接成功，则 KCBP 已经正常启动，否则 KCBP 存在异常，客户端无法登陆。

kcbpcp 连接时进行了权限检查，权限错误会提示 Login error : Authentication error。Connect 使用的用户需要在 KCBP 服务端的 user 中配置 group 为 manage。KCBP 的 User 支持 manage 和 application 两个组，manage 有管理权限，application 有 lbm 调用权限。

在 KCBP V3.0 版中，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,connecttimeout=5,userid=9999,password=3Mc+w9uU5lM=,timeout=30;option=

设置了连接参数之后，就可以使用本节前面的命令与非目标系统建立连接。

7.3.2.3调用 LBM

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。

7.3.2.3.1 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

   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。
5. 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。

6. kcbpcp 提示输入另一个 LBM 名称，这时可以输入 quit 返回。

7.3.2.3.2 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。

7.3.2.3.3 KCBP 系统变量输入方法

如果需要输入 KCBP 的系统变量，格式是 kcbpsystemparam:系统变量编号：值，如：kcbpsystemparam:7:0001, 这相当于调用 KCBPCLI_SetSystemParam 设置 KCBP 系统变量，系统变量功能编号定义如下：

```
#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_SERVICENAME       6
#define KCBP_PARAM_RESERVED          7
```

#define KCBP_PARAM_DESTNODE 8

7.3.2.4循环调用 LBM

语法：
for Var = nInitialValue to nFinalValue [step nIncrement] Commands
例子：
kcbp => for @marketvalue = 0 to 3 step 1 execute 20102900 with market:@marketvalue
成功后输出：
每个 LBM 命令的执行结果.

7.3.2.5密码加密

KCBP 配置中需要的密文可使用下面的命令来生成。
命令格式为：
kcbp => encrypt
input key:KCXP00
input plaintext:888888
cipher text is: GV2gODkBbGg=
其中 GV2gODkBbGg=为 888888 的密文。

7.3.2.6察看表定义

kcbp => list tables
输出：
Retrieved 1 node.
Node 1:
Table : system
Table : deputy
Table : biz
Table : transfer
Table : thread
Table : log
Table : internalqueue
Table : externalqueue
Table : xa
Table : program
Table : error
Table : user
Table : publish

Table : subscribe
Table : rsl
Table : priority

7.3.2.7在表中插入和删除行

```
kcbp => select * from rsl
   Retrieved 1 node.
   Node 1:
      Table : rsl
      concurrence=-1 level=0

kcbp => insert into rsl (level,concurrence) values (10,-1)
   Insert 1 node.
kcbp => select * from rsl
   Retrieved 2 node.
   Node 1:
      Table : rsl
      concurrence=-1 level=0
   Node 2:
      Table : rsl
      concurrence=-1 level=10

kcbp => delete * from rsl where level=10
   Deleted 1 node.
kcbp => select * from rsl
   Retrieved 1 node.
   Node 1:
      Table : rsl
      concurrence=-1 level=0
```

7.3.2.8察看表内容

```
kcbp => select * from log
输出：
   Retrieved 2 node.
   Node 1:
      Table : log
      displaylevel=99 filename=runlog filepath=./log/run flushfile=no flushtermina
l=no key=11 maxfilenum=1 maxsize=10240 recordtype=cycle type=run writelevel=1000
```

Node 2:
Table : log
displaylevel=10000 filename=bizlog filepath=./log/biz flushfile=yes flushter
minal=no key=12 maxfilenum=1 maxsize=10240 recordtype=forward type=biz writeleve
l=10000

7.3.2.9更新表内容

kcbp => update log set (displaylevel=100) where type=run
成功后输出：
Update 1 node.
这个参数更改后马上起作用。
Update 命令需要注意使用 where 条件，当不输入 Where 条件时会更新所有的行。

7.3.2.10 执行命令脚本

命令脚本中可以存放多行 kcbpcp 命令，例如名称为 test.sc 的脚本：
connect to kcbp1 user KCXP00 using 888888
select * from log
disconnect

执行命令脚本方法如下：
kcbp => runsc test.sc
command line 2: connect to kcbp1 user KCXP00 using 888888

Connect to Server.
Login success.
command line 3: select * from log

Retrieved 2 node.
Node 1:
Table : log
displaylevel=100 filename=runlog filepath=./log/run flushfile=no flushtermin
al=no key=11 maxfilenum=1 maxsize=10240 recordtype=cycle type=run writelevel=100
0
Node 2:
Table : log
displaylevel=10000 filename=bizlog filepath=./log/biz flushfile=yes flushter
minal=no key=12 maxfilenum=1 maxsize=10240 recordtype=forward type=biz writeleve
l=10000

command line 4: disconnect
Disconnect succuss .

提示：用户可以使用命令脚本批量测试 LBM 的功能。

7.3.2.11 调用操作系统命令

显示当前目录：
kcbp => run cd
输出：
D:\kcbp\work\kcbp.win\debug

7.3.2.12 启动 KCBP

kcbp => start

7.3.2.13 关闭 KCBP

kcbp => stop

7.3.2.14 启动 shell

kcbp => shell
在 shell 中操作完毕后，输入 exit 返回 kcbpcp 环境。

7.3.2.15 切换连接

kcbp => connect to kcbp1 user KCXP00 using 888888
Connect to Server.
Login success.
kcbp => connect to kcbp2 user KCXP00 using 888888
Login success.
kcbp => set connection kcbp1
命令执行后，kcbp1 为当前连接。

7.3.2.16 显示详细信息

察看当前设置：
kcbp => verbose
  Verbose mode off
显示详细信息：
kcbp => verbose on
这个命令作用范围只在 kcbpcp 本身。

7.3.2.17 断开连接

kcbp => disconnect
  Disconnect succuss .

7.3.2.18 退出 kcbpcp

kcbp => quit

7.4 LBM 调试工具 debuglbm

在 KCBPV3.0 版，DebugLBM 功能已经转移到 KCBPCP 中。

本节内容仅供适用于 KCBPV3.0 以下版本。

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 LBMAPI 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。

7.4.1 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. Debuglbm 执行 lbm，将结果输出到屏幕上。
6. 输入 quit 退出

这时，Debuglbm 和 kcbpcp 比较如下：

1. kcbpcp 调用 LBM 时输入 LBM 名称，debuglbm 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 debuglbm 则不通过 KCBP Server，它自己直接调用 LBM。
4. debuglbm 直接显示 lbm 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 debuglbm 相同，也是裸格式。

7.4.2 Windows 上使用 VC 和 debuglbm 调试 LBM

Debuglbm 和 VC、LBM 配合进行单步调试，步骤如下：

1. 首先在需要调试的 LBM 中适当的位置插入一行 KCBP_DebugBreak，设置断点，编译出 Debug 版的 DLL。
2. 其次在 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。
3. 再次调试 debuglbm，用 msdev –p debuglbm 进程号。
4. debuglbm 提示”input lbm file full path =>”，这时输入 lbm 所在动态库的全路径。
5. debuglbm 提示”input lbm export function =>”，这时输入 lbm 的入口名称。
6. 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。
7. Debuglbm 执行 lbm，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

7.4.3 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 命令起动 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 srkcode 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

7.4.4 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，如果是以多进程方式运行 KCBP，为了便于调试，注意设置 kcbpas 初始化进程数目为 1 个。

[注：KCBP V3.0 中，kcbpcp 不存在本节提到的任何限制]

7.5 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。

7.5.1 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 -concurrence 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

-concurrence, LBM concurrence, -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

7.5.2 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 的数据库连接命令。

```
#makefile for kcbplbm, Mr. Yuwei. Du , 20020719
PLATFORM = LINUX
CC = gcc
LBMTYPE = LBM
LBMAPI = lbmapi
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 -l$(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.*
```

7.6 KCBP 系统压力工具 KCBPTEST

KCBPTest 是一个 Windows 系统上的图形界面压力测试工具，通过它可以调用 KCBP Server 上的 LBM，它支持多线程，可逐笔配置 LBM 调用参数，能记录调用结果，可统计性能数据。

KCBPTest 是一个可扩展的压力测试工具，它采用适配器的方法连接不同的目标系统。目前我们可提供中国证券行业内主流柜台供应商的各种柜台的适配器。另外，只要遵循适配器的开发规范，非常易于实现对其他目标系统的压力测试。

7.6.1 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.dl libzlib.dll libzlib1.dll SSLeay32.dll

7.6.2 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,ssccCA.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: https://192.168.40.65:80 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，

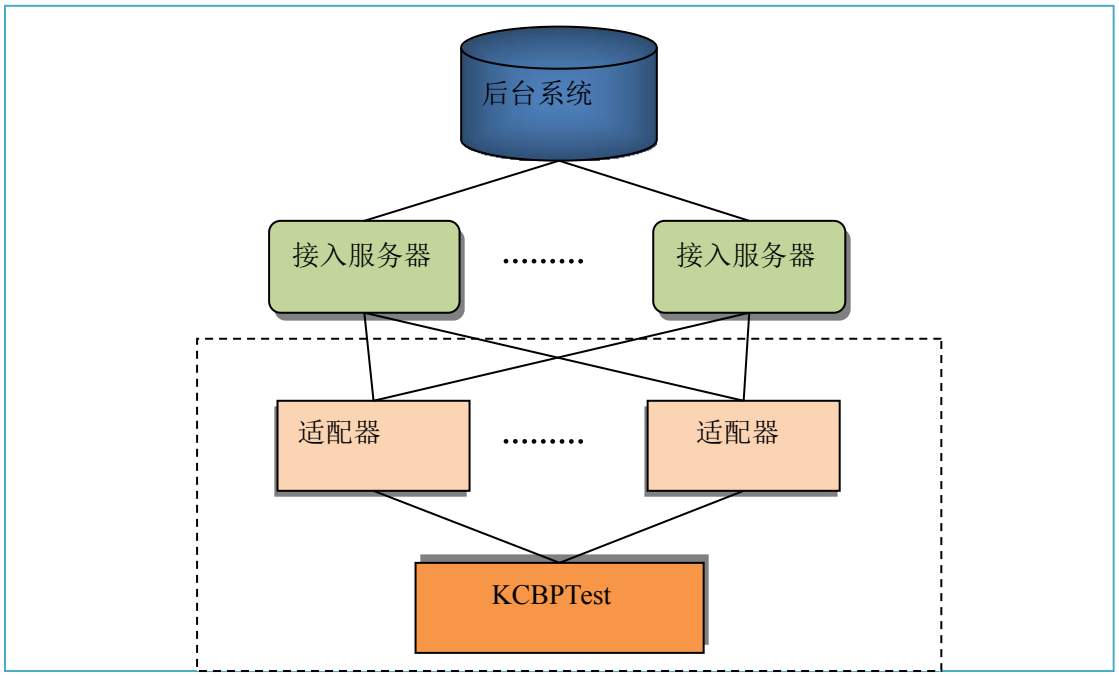
只需将抬头修改为[TASK2]。

7.6.3 指令文件格式

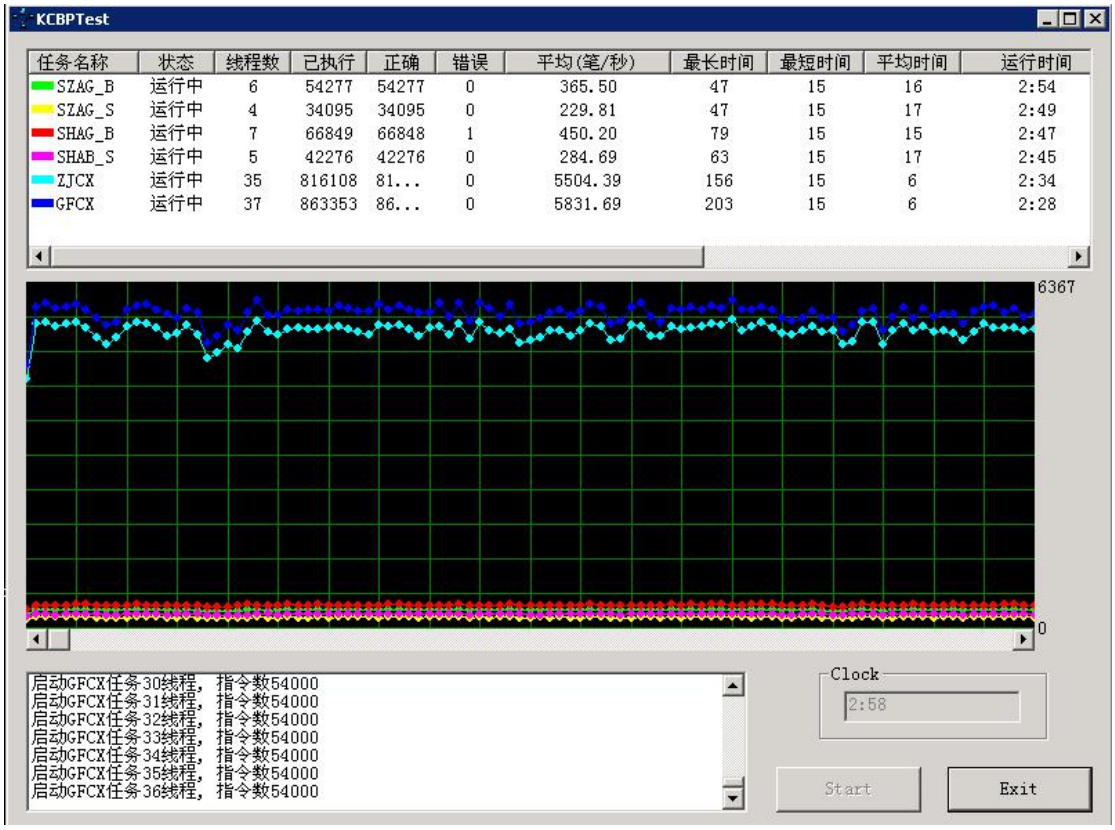
指令文件由一行或多行指令构成，每行存放一条指令，指令包括要调用的 LBM 名称及参数，LBM 名称何参数之间使用感叹号分隔，参数之间使用逗号分隔，参数名称和值之间使用冒号分隔，例如：

150101!USERTYPE:3,USERID:1,CUSTOMER:1001,MARKET:0,TRADE_ACCT:C920001001,STK_INTL:900921,TRADE:0B,VOLUME:100,PRICE:0.41,BRANCH:0001,CHANNEL:0

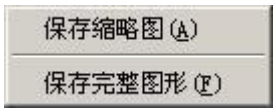
7.6.4 测试系统架构



7.6.5 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

7.6.6 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

7.6.7 适配器接口函数列表

```
#ifndef _KDMIDAPI_H
#define _KDMIDAPI_H
#ifdef KDMIDAPI_EXPORTS
#define KDMIDAPI_API __declspec(dllexport)
#else
#define KDMIDAPI_API __declspec(dllimport)
#endif
#define KDMIDAPISTDCALL __stdcall          /* ensure stcall calling convention on NT */
typedef void *KDMIDAPIHANDLE;
#ifdef __cplusplus
extern "C" {
#endif
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
```

7.6.8 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!khbslx: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。

7.7 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.

本节内容仅供适用于 KCBPV3.0 以下版本。

LBMTest 是一个 Windows 系统上的图形界面压力测试工具，通过它可以直接调用 LBM 进行压力测试，它支持多线程，可逐笔配置 LBM 调用参数，能记录调用结果，可统计性能数据。它与 KCBPTest 差别在于 LBMTest 直接调用 LBM，而 KCBPTest 通过 KCBP Server 调用 LBM，这也是 debuglbm 和 kcbpcp 调用 LBM 时的差别。

7.7.1 配置文件说明

LBMTest 除了使用 KCBPServer 的配置文件之外，还有一个自己的配置文件 LBMTest.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,TASKn 的参数配置与 KCPBTest 的 TASK 配置项一致，各项参数含义具体说明如下：

配置节	配置项	说明
COMMON	STEP	线程每次读取指令数
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

7.7.2 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 路径、入口、及参数，他们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：

kcbplbm.dll!stkOrder!USERTYPE:3,USERID:1,CUSTOMER:1001,MARKET:0,TRADE_ACCT:C920001001,STK_INTL:900921,TRADE:0B,VOLUME:100,PRICE:0.41,BRANCH:0001,CHANNEL:0

注意 LBMTest 的指令文件与 KCBPTest 指令文件之间存在差别：KCBPTest 指令使用 LBM 名称，而 LBMTest 指令使用 LBM 路径及入口。

7.7.3 界面显示结果说明

与 KCBPTest 的界面显示结果一致，参见 7.3.3。

7.8 其他工具

7.8.1 kcbpcmd 用法

kcbpcmd 是 kcbpcp 命令和 kcbpadd 命令的运行环境，当需要在批处理文件中使用 kcbpcp 命令时，需要先启动 kcbpcmd，然后再调用批处理文件。比如，使用 makefile 编译并向 KCBP 系统提交 LBM 时，需要先运行 kcbpcmd，然后在 kcbpdmd 的命令窗口下再调用 nmake 或 gmake。

7.8.2 kcbpkill 用法

kcbpkill 是一个小工具，可以按名称杀进程，一般不会用到。
用法如下：

```
kcbpkill : Display all processes.
kcbpkill pname : Kill all processes which name is pname.
```

7.8.3 imdbcmd 用法

imdbcmd 是 KCBP 提供的一个内存数据库命令行工具：

```
IMDB command processor Version: 1.0
Copyright Shenzhen Kingdom Co. Ltd. 2008
Author: Mr. Yuwei.Du
```

```
IMDB: help
Help message:
help | ?
use [kcbpdb | imdb | dbname] : switch database, imdb mean in memory database.
list tables : select * from QSYS_TABLE_INFO
list column tablename: display columns information defined in QSYS_COLUMN_INFO
time : display current time
mi : display memory information
import table from file [delimiter] [endline]: import table
export table to file [delimiter] [endline]: export market to market.txt 9 13
unlock table: release all lock on the shared table
otherwise, please use SQL syntax.
```

其中，命令 list tables 是查看系统中所有表，list column 查看指定表的列定义信息，mi 显示 IMDB 内存使用情况，import 从外部文件导入表，export 导出表内容到外部文件，unlock 对死锁表进行解锁。
除上述命令外，用户还可使用 SQL 语句对数据库及表进行操作。

8. 应用

8.1 KCBP 高可用性介绍

KCBP 提供各种高可用性解决方案，最大限度减少关键业务系统计划内及计划外停机次数和恢复时间。这些解决方案为业务系统进行 7*24 小时服务提供技术保障。

8.1.1 导言

KCBP 是面向企业关键业务系统而开发，而关键业务系统对高可用性要求期望非常高。为了更好满足这类系统的高可靠性需求，KCBP 提供了多种高有针对性可用性设计方案，力求在高可用性方面做得更好。我们在后面章节简要介绍 KCBP 的高可用性，希望能加强大家对高可用性的理解，优化系统部署，提高系统的运行服务质量。本文内容适合 KCBP/Windows 系统，除单线程多任务外的内容也适合 KCBP/UNIX 系统。

8.1.2 模糊负载均衡及容错机制

模糊负载均衡及容错机制是目前各类中间件所能提供的负载均衡及容错机制中

最强大、最有效的一种实现机制。KCBP 在 2002 年便实现了这种机制，历经多年的生产检验，已经被实践证明是一种优异的方案。这种机制，建立在 KCXP 的队列技术基础之上，通过队列实现了作为消费者的客户端应用程序和作为生产者 KCBP 的服务端的解耦。目前，每个 KCXP 可以为 8192 个 KCBP 提供队列服务，目前最大可以建设由 8192 个 KCBP 和 8192 个 KCXP 组成的庞大集群，对外提供统一的服务。集群中每个 KCBP 和 KCXP 节点，其处理能力，由本身的硬件及软件配置决定，集群中每个 KCBP 节点都是一个独立的对等处理节点，KCBP 节点之间不需要传递信息，业务处理不是由消费者来决定调用哪个生产者，而是将请求写入到队列中，由生产者在线程能够处理请求时选取请求进行处理，实现了模糊负载均衡机制。整个集群对外看来是一个整体，通过增加监听同一队列的 KCBP 节点的数量可以透明地提升整体吞吐量，通过控制监听队列的每个 KCBP 节点线程数量可简单地调节各节点的负载。每个 KCBP、KCXP 节点均可热插拔，这个过程对客户程序是透明的，不需要客户端程序的参与。这项技术实现了透明的故障转移功能，如果监听同一队列的一些 KCBP 终止了，剩下的仍然会继续选取并处理消息。KCBP 的负载均衡和容错机制，实际上也是一种网格计算机制，KCXP 和 KCBP 完整封装了网格功能，构成了网格计算的基础设施，由于 KCBP、KCXP 支持多种操作系统，因此这个网格支持异构结构，并且，网格中的每个节点可以根据自身应用负载情况自动调整计算规模。KCBP 的这项技术，理论上可以提供无限的处理能力，是现代软件技术的巅峰之作，如果大家没感觉到这点，恐怕是“不识庐山真面目，只缘身在此山中”。

8.1.3 进程式服务器

KCBP 的业务处理器-KCBPAS（KCBP application server）设计成既可运行在线程方式运行，又可以运行在进程方式。由于 KCBP 面向的业务领域是关键业务处理系统，而这个领域对业务程序处理性能要求很高，因此，业务程序（LBM）都是使用高效的 C/C++语言编写，由于 C/C++语言的固有特性，逻辑不严谨的业务程序可能出现导致业务处理进程崩溃的各种 BUG，为了避免业务程序的相互干扰，KCBPAS 采用进程式运行方式来预防这种情况的出现。根据我们在主流 PC 服务器上的测试数据和客户多年运行效果的分析，我们确认 Windows2003 以上版本的操作系统对进程的调度效率与对线程的调度效率看起来非常接近，几乎看不出差别。实际上，进程式服务器的方法不只是我们采用，Google 浏览器也采用类似的思想避免页面之间的干扰。我们 2003 年就开始采用，而 Google 直到 2008 年才开始采用。考虑到关键业务系统对高可靠性的要求，我们推荐客户采用多进程方式运行 KCBP。

8.1.4 应用服务进程规模自动调节

KCBP 建议一组 KCBPAS 用来处理业务逻辑，每个 KCBPAS 可以处理任何业务逻辑，与业务程序之间没有亲和性，KCBPAS 之间是对等的关系。编写不严谨的业务程序运行时可能出现挂起、死循环等错误，这种错误一经出现，就会占用一

个 KCBPAS 进程。有的业务程序，比如大数据量的历史数据统计，可能需要运行较长时间，也会长时间占用 KCBPAS 进程，而这类业务并发较高的情况，会占用大量的 KCBPAS 进程，导致可用的 KCBPAS 进程数目减少，系统处理能力不足。为了应对前面两种情况，KCBP 采用服务进程规模自动调节技术，尽量保证服务质量与业务逻辑无关。KCBP 中有几个参数与规模自动调节相关：

Min AS 和 Max AS：Min AS 表示系统 KCBPAS 的最低水位，Max AS 表示 KCBPAS 的最高水位。KCBP 系统启动时，KCBPAS 处于最低水位。KCBP 运行过程中，根据业务处理情况在最低水位和最高水位之间调节 KCBPAS。当 Min AS 与 Max AS 相等时，自动调节功能处于关闭状态。

Overload factor 和 Overload time：Overload factor 表示处于忙状态的 KCBPAS 个数占总 KCBPAS 个数的百分比，当忙于工作的 KCBPAS 比例超过该百分比时，并且持续时间超过 Overload factor 所设定的时间（以秒为单位），KCBP 系统将根据 CPU 和内存负载情况调整 KCBPAS 规模。

Max CPU Load：控制允许自动调节 KCBPAS 的最大 CPU 负载。当 CPU 负载超过该值，不自动调节 KCBPAS 规模。

Max Memory Load：控制运行允许自动调节 KCBPAS 的内存最大利用率。当内存利用率超过该值，不自动调节 KCBPAS 规模。

Recycle time：KCBPAS 回收时间，单位为秒。当系统中存在多于 Min AS 个数的 KCBPAS 进程，并且在该时间段内正在使用的 KCBPAS 数目一直小于 Min AS 数目，则表示部分 KCBPAS 处于空闲状态，KCBP 将回收处于空闲状态的 KCBPAS 进程，直到 KCBPAS 数目达到 Min AS 所设定的个数。

客户在使用自动调节功能时，要正确理解上述参数，并结合自己的系统特点合理搭配，防止 KCBPAS 进程个数变化频繁，剧烈抖动，保证系统平稳运行。

8.1.5 业务处理超时自动监测及回收

当业务逻辑编制不合理，出现业务挂起、死循环等状态，或由于后台系统出现故障，导致系统处理缓慢，发生 KCBPAS 进程被占用，KCBPAS 可用数目不足，系统处理能力下降的问题。我们可以通过 KCBPAS 规模自动调节功能增加 KCBPAS 个数来缓解这个问题，但由于 KCBPAS 个数总归有个上限，最后总会碰到 KCBPAS 全部挂起的情况，因此需要更有针对性的方案来彻底解决业务挂起、超时问题。

KCBP 提供业务超时自动监测和回收功能来解决上述问题。KCBP 自动监控每个 KCBPAS 进程正在处理的业务是否超过该 program 定义项的 timeout 设定预订时间，如果超时，KCBP 在控制台上每隔 1 秒打出 1 行提示，并且，当设置了 Auto kill timeout AS 选项时，KCBP 将终止该超时的 KCBPAS 进程。被终止的 KCBPAS 进程，会被重新创建。注意，使用该项功能，需要合理设置每个 LBM 的超时时间，否则可能导致需要长时间运行的业务，由于超时时间设置太短，始终被 KCBP 当作异常业务自动终止，无法正常运行。

现在的证券集中交易系统上有超过 3000 个业务，要对每个业务设置相对合理的超时时间，工作量太大，难以实现，该怎么办？对于这个问题，可以逆向思考。考虑到大多数业务逻辑编制的都是正确的，运行不会出现死循环、挂起的问题，

因此，可以将其超时时间设置为-1，表示不限定超时时间；当在系统运行过程中，发现某个业务出现挂起、死循环问题，这时，如果修改业务程序解决这个问题需要一定的时间，为了保证生产系统的继续运行，可以临时对这个问题业务设置一个合理的超时时间，并控制其最大并发数，控制其影响范围，保证正常的业务平稳运行。
注意：这项功能相关因素较多，可能对系统产生较大影响，要慎重使用。

8.1.6 业务处理崩溃自动修复

当业务逻辑编制不合理，引起 KCBPAS 进程崩溃，导致 KCBPAS 可用数目不足，系统处理能力会出现下降的问题。当业务处理超时自动监测及回收功能终止了异常的 KCBPAS 进程时，也会产生同样问题。为了解决这个问题，KCBP 提供崩溃进程自动修复功能，让系统能持续提供必要的处理能力。这项功能始终处于使用状态。
另外，当 KCBP 的主控进程出现异常，发生崩溃时，KCBP 也可以自动修复主控进程。这项功能由 KCBPDaemon 进程提供，当 KCBP 启动时，KCBPDaemon 会被自动加载，自动监控 KCBP 主控进程的状态。注意，我们建议在生产系统上打开 auto repair crash 启用这项功能。

8.1.7 业务及分组最大并发度控制

为了控制异常业务及需要长时间运行的业务对 KCBPAS 进程的占用，KCBP 提供业务及分组最大并发度限制。
首先，KCBP 提供按业务控制最大并发度，这是最小力度的并发度控制方式。
其次，KCBP 提供 2 种分组方式提供业务并发度控制：一是按 RSL 分组，二是按 priority 分组，每个维度各提供 64 组。
对于长时间运行的业务，如与第三方系统同步式通讯的业务，由于运行时间不是完全由自己控制的，因此有必要采用合适的方式控制并发数，防止出现异常时消耗过多 KCBPAS 资源，影响系统的处理能力。

8.1.8 资源调控器

结合 KCBP 提供的统计功能，系统运行维护人员可以准确识别系统中需要长时间运行的业务，并可根据需要在每个 KCBP 上对这些业务进行分组，并限制业务组的并发数，控制其对资源的使用。但是，当系统中存在多个业务分组的情况下，其总的并发数可能超过 KCBPAS 总数，我们称这种情况为并发溢出，这时，按分组控制并发数策略失效，采用什么办法才能应对？
设想一下：如果业务已经被 KCBP 获取，在业务处理过程中发现并溢出问题并去想办法解决，实际上是一种被动的问题处理方式，如果能在业务被 KCBP 获

取之前主动进行资源调配，那么，我们就可以先发制人，防范上述的并发溢出问题。俗话说的“先发制人，后发之于人”，还是颇有道理，KCBP 何不采取主动技术预防问题？
好了，现在我们对 KCXP 部署资源调控器，结合 KCBP 的主、从节点部署策略，主动防范 KCBP 主处理节点并发溢出问题。具体来讲，我们可以分配从 KCBP 节点，专门用来处理需要长时间运行的业务，而主 KCBP 节点只用来处理短业务，这样，长业务根本没有机会造成主 KCBP 并发溢出。主、从 KCBP 分别守候不同的请求队列，相互间不需协调。业务分配在 KCXP 上实现。KCXP 支持按请求服务名称分配队列功能，在其配置文件 exit.ini 中进行简单配置即可。
前面所述，实际是资源调控器的最简单功能，我们完全可以利用资源调控器提供更多的功能，比如，按前端应用名称进行资源调控、按请求报文内容进行资源调控、赋予不同的服务不同的优先级别等功能，以及按照组合条件进行资源调控。由于篇幅限制，具体内容此处不详述。

8.1.9 单线程多任务处理

KCBP 提供独创的单线程多任务处理功能，可彻底解决长时间运行的业务对 KCBPAS 进程的占用问题。目前的交易中间件产品，包括商业中间件 CICS、TUXEDO、也包括各种 J2EE 中间件，以及各种国产中间件，事实上都遵循线程单并发原则，这个原则是这样的：中间件有 M 个业务处理线程（进程），在某一时刻最大并发处理能力是 M 笔业务，这时每个线程处理一笔业务。每个线程，在任何时间段内，对业务程序的处理都是串行的，一个业务程序一旦开始，就必须运行到结束。系统以业务程序为是最小执行单位。
现在，在 KCBP 上，交易中间件产品的思维定式已经不复存在，我们已经在 KCBP/WIN 上做到：在一定的条件下，单线程多并发处理，即一个线程在一个时间段内并发处理 N（N >=1）笔业务。这时，业务程序不再串行处理，系统不再以整个业务程序为最小执行单位，KCBP 自动将业务程序分割成多个程序单元，以业务单元为最小执行单位。一个线程，在一定时间段内，可以并发运行多个业务程序，并在不同的业务程序的业务单元间自动切换处理。
KCBP 在处理存在系统间通讯的 LBM 时，会自动将业务程序拆分为业务单元，然后按业务单元进行处理。具体地讲，在第一个 KCBP 处理的业务程序 A 将请求（比如转账）发送给负责银证通讯处理的第二个 KCBP 后，第一个 KCBP 自动保存业务执行环境，并将业务程序 A 从线程中卸掉，然后处理其他的业务程序，当第二个 KCBP 向第一个 KCBP 返回结果时，第一个 KCBP 在当前正在处理的业务程序完成一个单元后，恢复前面保存过的业务程序 A 的执行环境，继续业务程序 A 的处理。系统间通讯进行等待时，不占用中间的处理线程，可以并行处理其他业务。
对业务程序来讲，KCBP 单线程多任务是透明的，任务转移、调度、唤醒完全由 KCBP 自动处理，KCBP 的业务处理线程也不会为等待系统间通讯结果而阻塞。KCBP 系统单线程多任务功能由 KCBP1PCAsync.dll 接口提供。

8.1.10 资源管理器断线自动恢复

这是一项实用的功能，可以解决由于网络瞬断、数据库服务器集群切换等原因导致的中间件与资源管理器失去连接的问题。KCBP 可自动处理断线恢复问题，整个过程不需要人工干预，有了这项功能，就可以进一步保证关键业务系统的服务质量。这项功能是由 KCBP 的各 XA 通讯模块提供，业务程序 LBM 参与断线恢复。当 LBM 调用 KCBP_Commit 失败时，KCBP 会自动检查是否发生断线情况，如果发生，将自动尝试恢复连接。
读者不要以为这项功能只是小把戏，没什么好说的，告诉大家，CICS、TUXEDO 等产品均不提供这项功能，外国的月亮并不总是圆的，本地的和尚念经可以念的更好。

8.1.11 通讯服务器集群

当存在一组提供通讯服务区的服务器 KCXP 时，为了进一步提高提供运维质量，可以为将 KCXP 建立集群。这样做的好处就是，当集群中某个 KCXP 节点出现故障时，不会影响客户端应用，故常处理与恢复过程对前端应用来讲是透明的。目前，每个 KCXP 集群最多支持 16 个 KCXP 节点。整个系统中，理论上可以存在无数的这样集群。这项功能既可以提供容错功能，也可以提供负载均衡功能。这里的负载均衡，采用的不是模糊算法，而是基于 KCXP 客户端参与的紧耦合算法：KCXP 集群之间交换负载信息的，当 KCXP 客户端建立连接的时候，首先取得集群中各节点的负载情况，如果存在一个最小负载的节点，那么 KCXP 客户端就与该节点建立连接。
KCXP 支持多级集群，当 KCXP 部署多层时，每层都可以部署成集群方式，集群可实现故障自动接管，对外表现是透明的，这种部署方案对类似券商的总部-营业部多层网络架构环境非常方便有效。
与 KCXP 集群功能配合，KCBP 客户端每个实例可配置 16 个单独的接入点，并采用随机算法自动分配接入点。通过这个方法，我们就可以在实现整个 KCXP 集群的各接入点之间均衡负载。
实际上，当我们采用模糊负载均衡及容错机制时，完全可以不用通讯服务器的集群功能，如果 KCBP 客户端设置了多个接入点，并不会降低整个系统的容错和均衡性能。

8.1.12 动态配置

KCBP 上的多项参数，包括加密方式、压缩方式、压缩阀值、日志级别、LBM 程序的增删改、用户定义信息、错误定义信息、错误重试时间、各种并发控制方式、业务分组并发数目、各种通讯参数、服务规模自动调整各相关参数、统计功能、关闭提示功能、ＫＣＭＭ监控功能等都支持动态配置，参数修改后可即时生效。这些设计使 KCBP 在不停机状态就可调整参数，保证系统的服务质量。

注意，动态配置目前只能通过命令行处理器 kcbpcp 来修改，图形管理界面修改的参数尚不支持动态配置。
由于 KCBP 实现了模糊负载均衡及容错机制，当系统中存在多个对等的 KCBP 节点时，并且处理能力有余，我们调整参数往往不是采用命令行动态调整参数的方式，而是采用图形界面调整参数后重新启动节点的方式，因为这样做会更简单，并且不会影响系统的服务质量。

8.1.13 集成监控

这项方案包括几部分：
第一，KCBP 集群中各节点可以向 KCMM 发送运行情况，使用 KCMM 可在单个控制台上集中管理全部 KCBP 节点，发现每个节点的故障，启动、停止每个节点，为提高系统的运维服务水平提供基础支持。KCBP 通过调用 mmsuite.dll 与 KCMMAgent 通讯，KCMM Agent 与 KCMM Server 通讯，监控参数存放到数据库中。KCMM Agent 安装在 KCBP 所在的机器上，mmsuite.dll 是 KCMM Agent 中的一个文件，安装在 windows 系统的 system32 目录。
KCBP 提供以下四个 bool 型监控参数，true 表示正常，false 表示异常：
KCXP Listener
KCBP Log
KCBP Daemon
KCBP AS
第二，KCBP 提供按每个业务和所有业务统计调用总次数、调用总时间等统计功能，这些统计信息可以显示在 KCBP 的界面上，同时，KCBP 还提供一个 LBMStat 业务供外部程序调用以获取上述信息。KCMM 能够调用 LBMStat 获取当前所有业务处理的统计信息。KCMM 可以定期调用这个接口，并显示出系统各业务的调用时序图。
第三，KCBP 可向 Windows 性能监控器提供单个业务的统计数据及全部业务的统计数据，方便用户通过性能监控器监控 KCBP 系统的运行情况。这些统计信息包括：处理业务的总数(executed #)，处理业务的总时间（executed total time(ms)），每秒处理业务的总数（executed # per second），每笔业务平均处理时间（average time(ms) per execution），以及当前正在运行的 KCBPAS 进程数目（# AS in progress）。每个业务缺省不提供性能计数器，如果设置该 LBM 的 Perfmon 属性为 yes，则开始对性能计数器每秒更新一次数据。
第四，在性能监控器的基础上，采用 snmptools 和 cacti，可方便实现远程监控多 KCBP。Snmptools 提供性能计数器到 SNMP 的转换，cacti 实现监控展示。有关这部分的详细内容，见 http://it.dianping.com/using_cacti_performance_counter_to_implement_customized_remote_monitoring.htm。
第五，借助 KCBP 计数器提供的统计功能，用户可以分析系统中各业务的调用频率和特点，指导系统的业务优化和部署。

8.1.14 一键式主备切换

大家对 KCBP 系统的理解，不能只局限于 KCBP 本身，我们所说的 KCBP，广义上讲是一个三位一体的系统，KCXP 是 KCBP 的通讯基础架构，KCMM 是 KCBP 的运维监控工具。借助于 KCMM 和 KCXP 提供的功能，业务系统可以实现一键式的主备切换，可以解决人工逐个切换时间长、忙乱易错的问题。一键式主备切换可以达到毫秒级别完成，对需要进行主备切换的场景非常方便实用。客户需要做的，就是在部署外层 KCXP 的时候，设置好主用节点和备用节点，并安装好 KCMM，设置好切换密码，做好演练，一切就是这么简单。

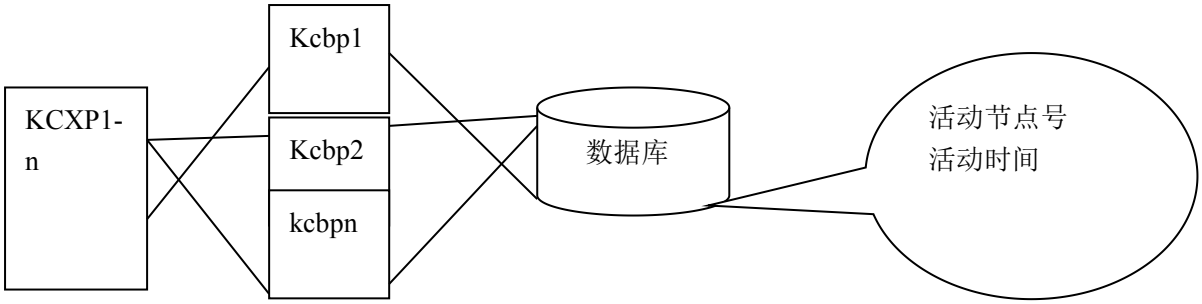
8.1.15 KCBP 串行化处理功能说明

一 目的

每个 KCBPAS 只处理一个 KCXP 请求队列送过来的请求，利用队列的先进先出特性，实现串行化处理业务。

高可用性通过 kcbp 主备集群实现，集群中只有一个 KCBP 处于工作状态，其它 KCBP 都处于后备状态。

二 原理



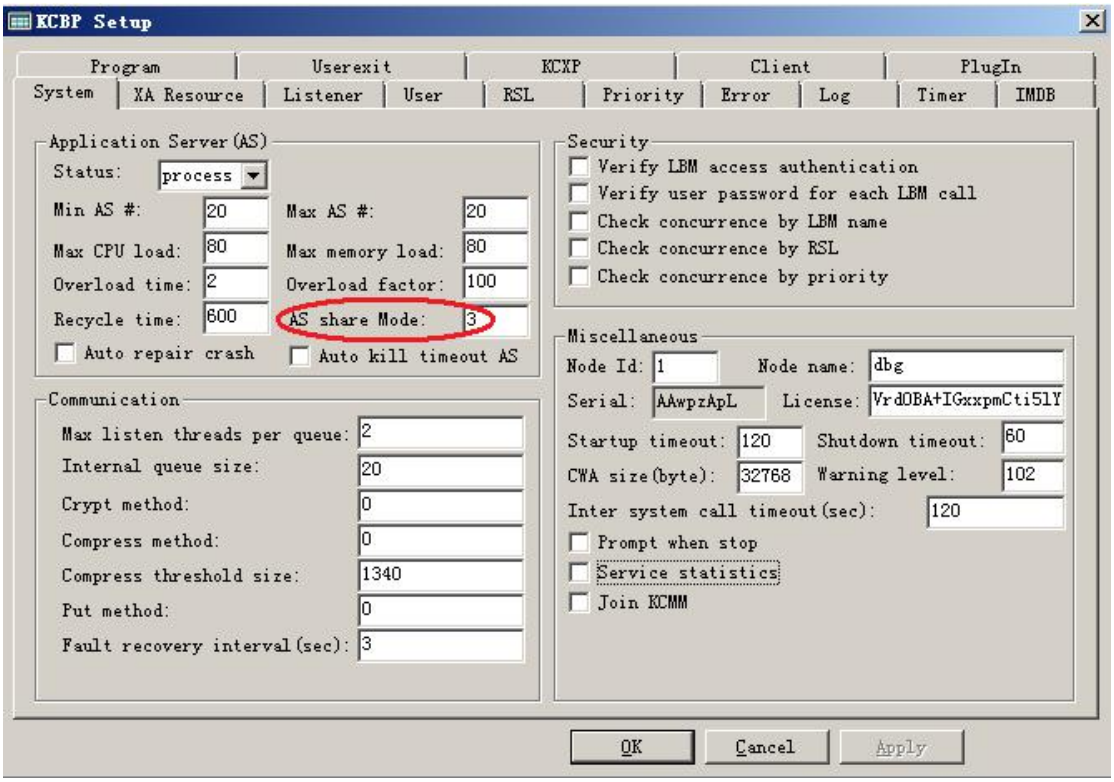
多个 KCBP 组成一个集群，集群中只有 1 个 KCBP 处于活动状态，其它 KCBP 处于后备状态，状态信息（活动 KCBP 节点号＋活动时间）通过数据库交换，该信息保存在数据库内存中，由 CLR 维护。每个 KCBP 上通过定时任务每秒调用一次专用 LBM： KCBPXA 更新状态信息。

集群中所有 KCBP 节点配置只有节点编号不同, 其余全部相同。

集群中 KCXP 可以配置多个。每个 KCBP 最多配置 1024 个 KCBPAS 进程。

三 生产版本在 KCBP System 配置页面, 配置 AS Share Mode 为 3 ,这时, 每个 KCBPAS 专用于 1 组请求和应答队列，并且出于后备状态。

关于 KCBPAS 运行状态: 1 专用+活动,3 专用+后备, 0 共享，是普通 KCBP 的运行状态，2 部分共享+活动，4 部分共享+后备。



关于 AS Share Mode 的说明：

0：这是 KCBP 普通用法，所有 KCBPAS 处于共享模式，请求由 KCBP 的 Listener 线程从 KCXP 队列取出，放到 KCBP 内部队列（只有一个），所有 KCBPAS 从内部队列取请求，处理业务。 这个场景，KCBPAS 与 KCXP 请求队列没有直接关联。

1： 专用+活动，当前 KCBP 处于活动状态，KCBPAS 可以处理外部请求；指定的 KCBPAS 直接从 KCXP 请求队列获取请求消息， 每个 KCXP 请求队列，可以配置 1 个以上 KCBPAS。一个 KCBP 节点中 KCBPAS 总数量，应该大于服务于外部队列的 KCBPAS 数量，多出来的 KCBPAS 服务于内部队列，处理系统类请求（如定时任务）。本文应用场景中，每个 KCXP 队列配置 1 个 KCBPAS。

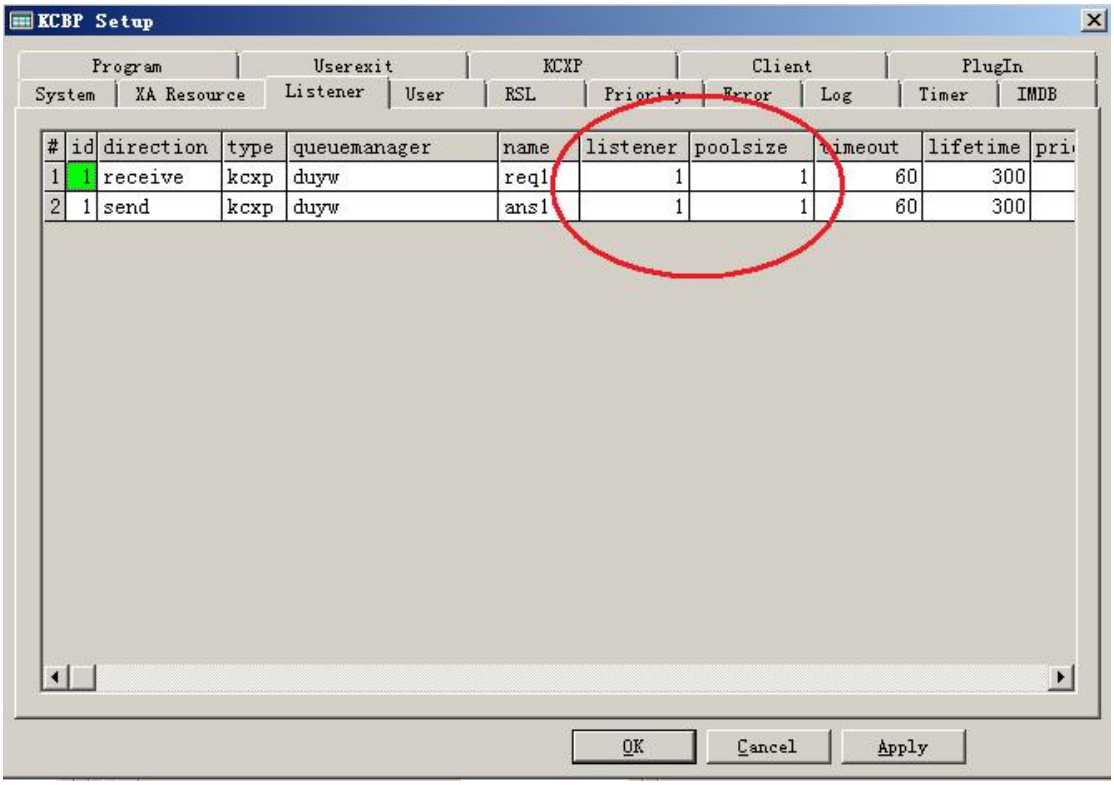
3：专用+后备，除了运行在后备状态，其余与 1 相同。后备状态 KCBPAS 不处理外部请求。后备状态与活动状态可以通过程序控制状态切换。KCBP 提供系统 LBM（KCBPHA）提供状态自动切换功能。

2：部分共享+活动，一个 KCBP 上所有 KCBPAS 依次平均分配给内部队列和外部队列使用。比如配置 3 组外部队列，20 个 KCBPAS 进程，那么，每个队列将有 20/（1+3）=5 个 KCBPAS 处理请求。

4：部分共享+后备，除了运在在后备状态不处理外部请求外，其余与 2 相同。

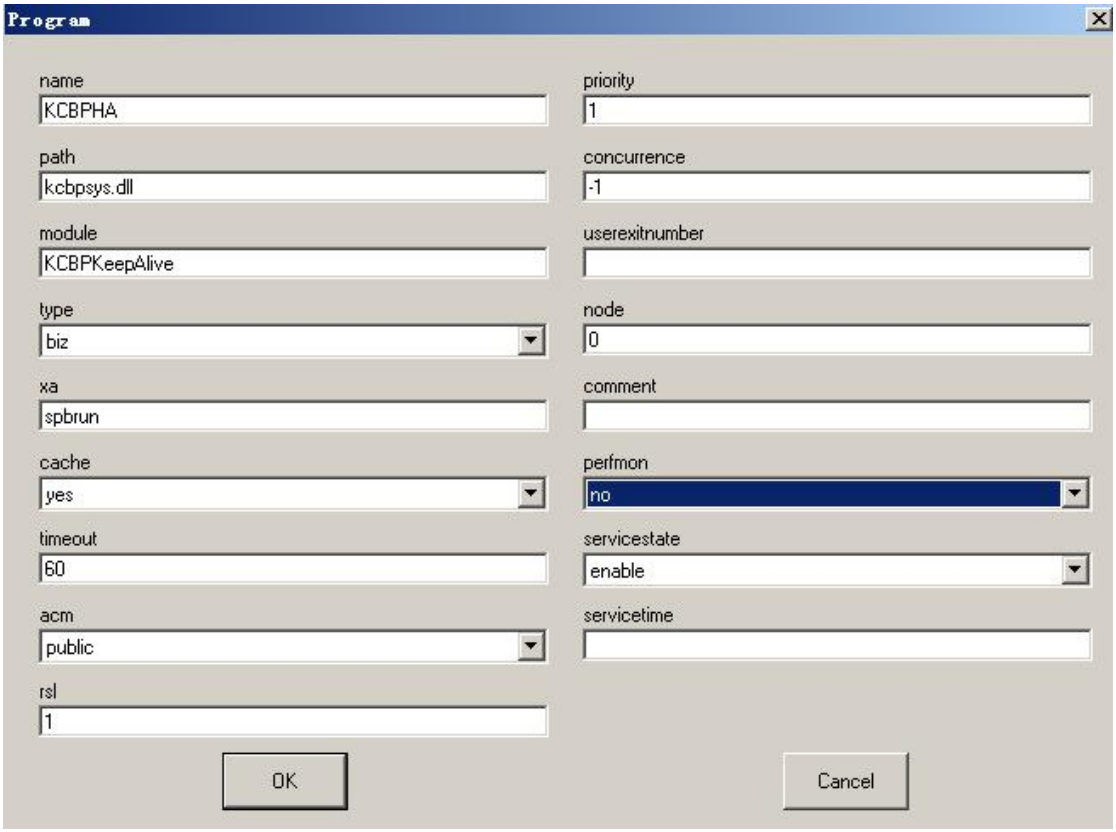
关于 KCBPAS 自动伸缩功能， 为了避免系统复杂，不建议使用在 AS Share Mode 0 之外的场景。

四 在 KCBP Listener 配置界面, 这是 Listener 为 1,Pool Size 为 1.

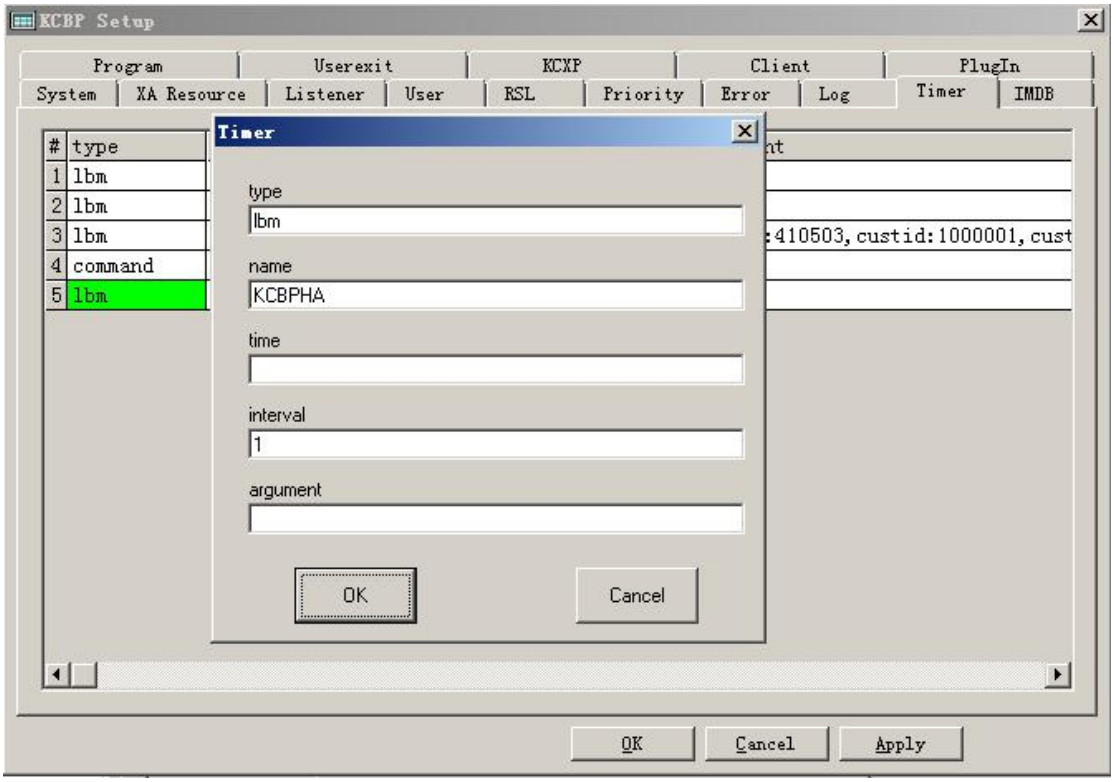


注意：kcbpas 进程数 〉 队列总组数。比如，有 20 对外部队列，那么 as 进程要大于 20， 超过 20 的进程数用于处理定时任务等内部请求。

五 配置 KCBPHA LBM，这个函数在 kcbpsys.dll 中。KCBPHA 与数据库系统交互， 通过调用存储过程 up_kcbp_keepalive 更新节点活动信息。 数据库中保存了 KCBP 状态切换时间，缺省是 5 秒，要改变该时间，需要执行存储过程 SetKCBPHeartBeat ，这个存储过程定义在 KDSerialGenerator CLR 中，有一个时间参数 （SetKCBPHeartBeat @Seconds int）。



六 配置定时任务。每秒调用一次 KCBPHA，更新 KCBPAS 运行状态。如果集群中没有其它 KCBP 处于活动状态(在约定时间没有更新活动状态)，那么当前 KCBP 将切换到活动状态，开始处理 KCXP 队列中的请求。



8.1.16 结束语

当企业中的关键业务系统，特别是证券集中交易系统这类实时性和可用性要求非常高的系统，出现故障无法正常工作时，都会产生严重的影响，造成巨大的损失。可用性问题和需求千差万别，一种解决方案无法解决所有的问题。针对高可用性的解决方案没有最好，只有更好。KCBP 试图提供多种解决方案以提高可用性，包括上述的模糊负载均衡及容错机制、进程式服务器、应用服务进程规模自动调节、业务处理超时自动监测及回收、业务处理崩溃自动修复、业务及分组最大并发度控制、资源调控器、单线程多任务处理、资源管理器断线自动恢复、通讯服务器集群、动态配置、集成监控、一键式主备切换等方案，这种可用性至少在性能和多样性方面能够媲美国际上主流中间件厂商所提供的任何类似产品，并且 KCBP 具有自己鲜明的特色。

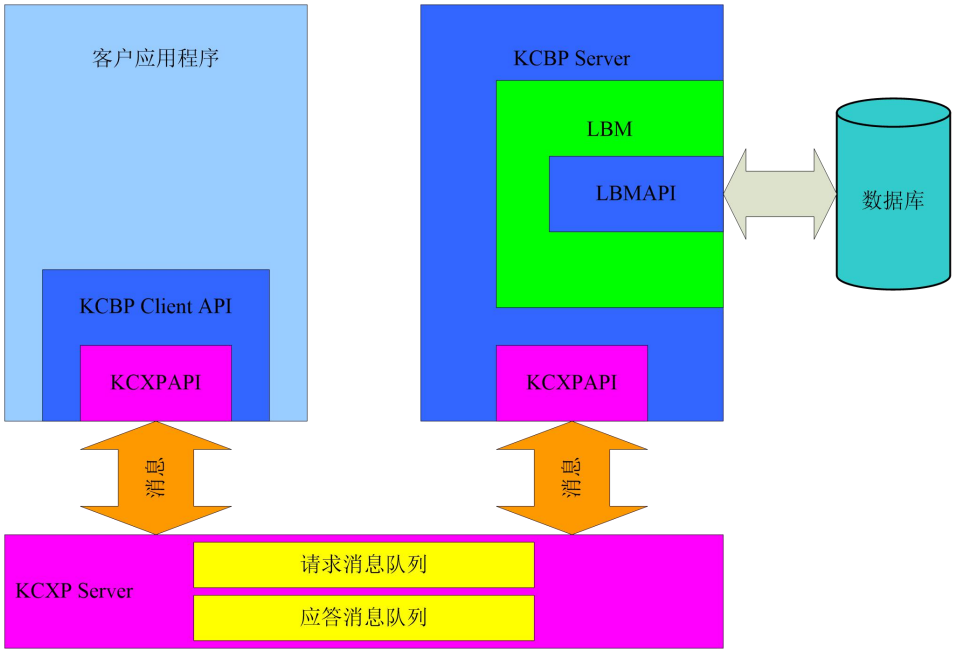
8.2 部署方案

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。

8.2.1 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；

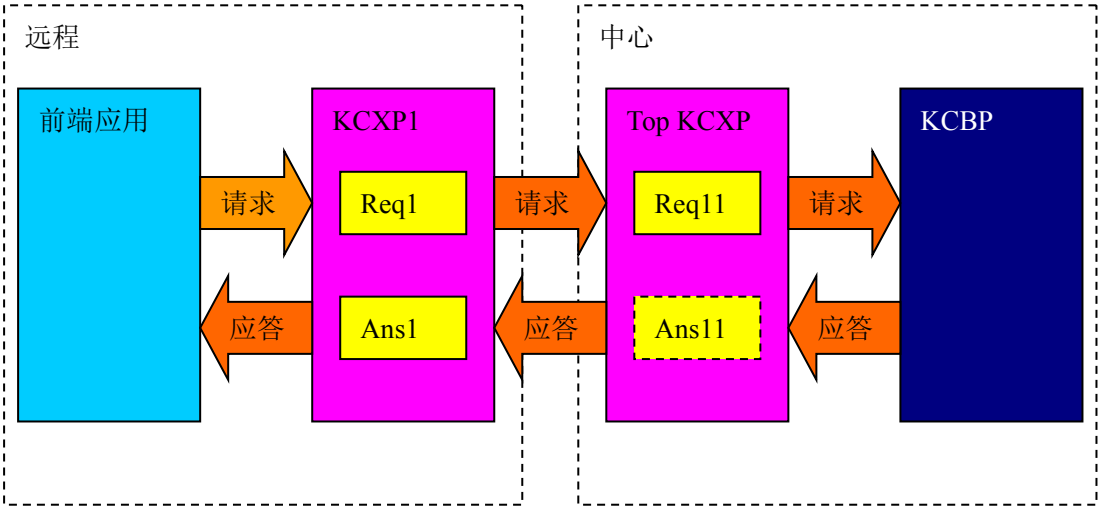
7. KCXP API 从 KCXP Server 上的应答消息队列中取得应答消息并将其返回给 KCBP Client；

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。

8.2.2 KCXP 多级部署方式

8.2.2.1 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。

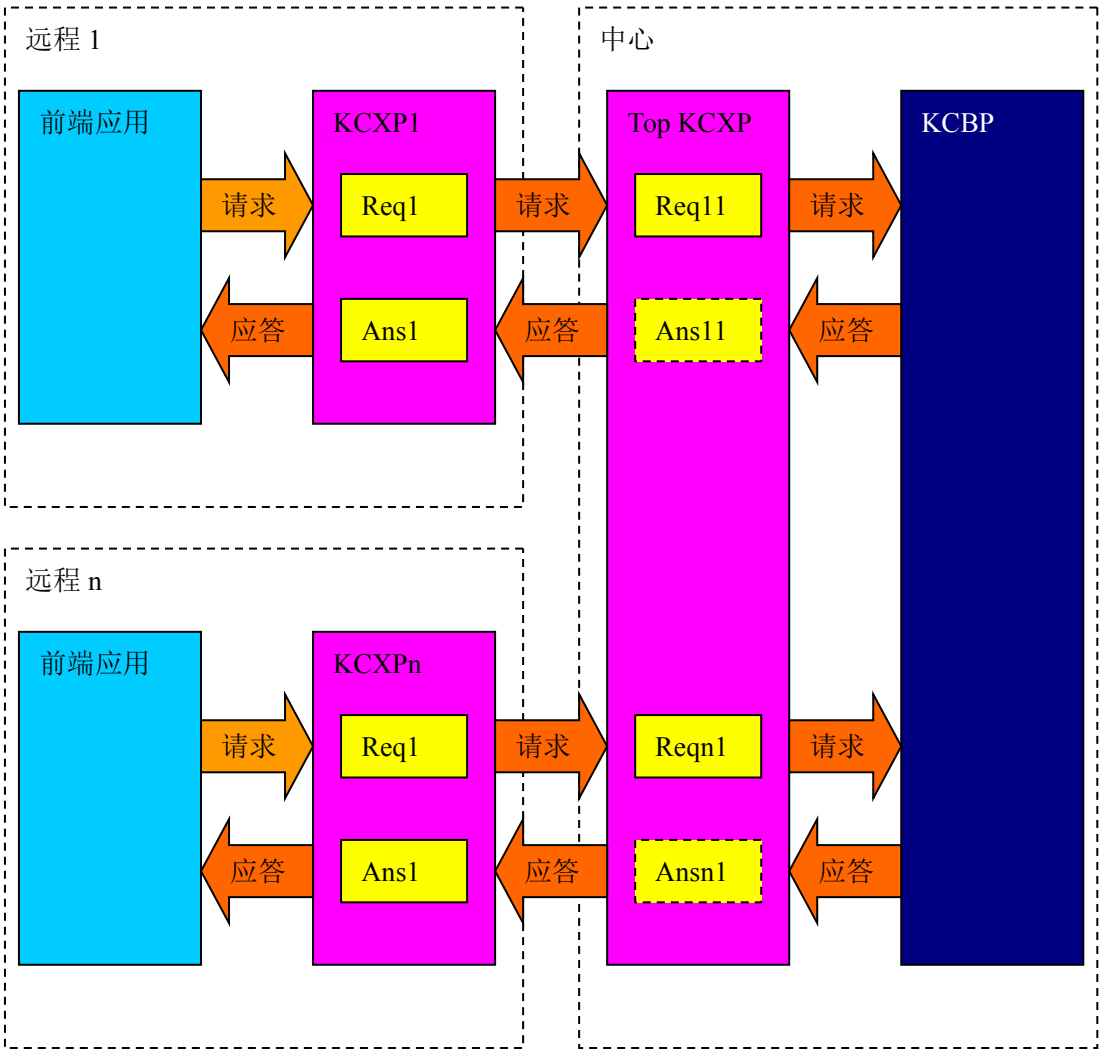
KCXP1 上面的请求队列 Req1 是一个远程队列，它对应 Top KCXP 上的本地队列 Req11，请求通过 Req1->Req11 传递给 KCBP。

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。

8.2.2.2 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>=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。

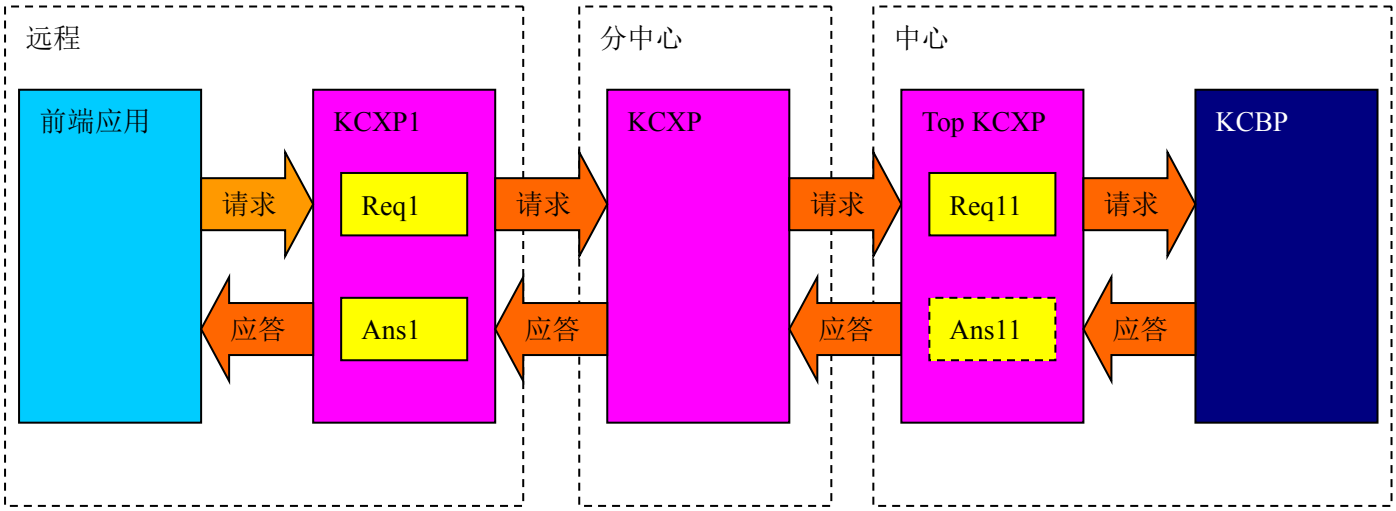
同理，KCXPn 上面的请求队列 Req1 是一个远程队列，它对应 Top

KCXP 上的本地队列 Reqn1，请求通过 Req1->Reqn1 传递给 KCBP。Top KCXP 上的应答队列 Ansn1 是一个远程队列，它对应 KCXPn 上的本地队列 Ans1，应答通过 Ansn1->Ans1 传递给前端应用。

Ansn1 的注意事项同上节。

8.2.2.3 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 Req1 是一个远程队列，它对应 Top KCXP 上的本地队列 Req11，请求通过远程 Req1->分中心->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->分中心->Ans1 传递给前端应用。

Ans11 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。

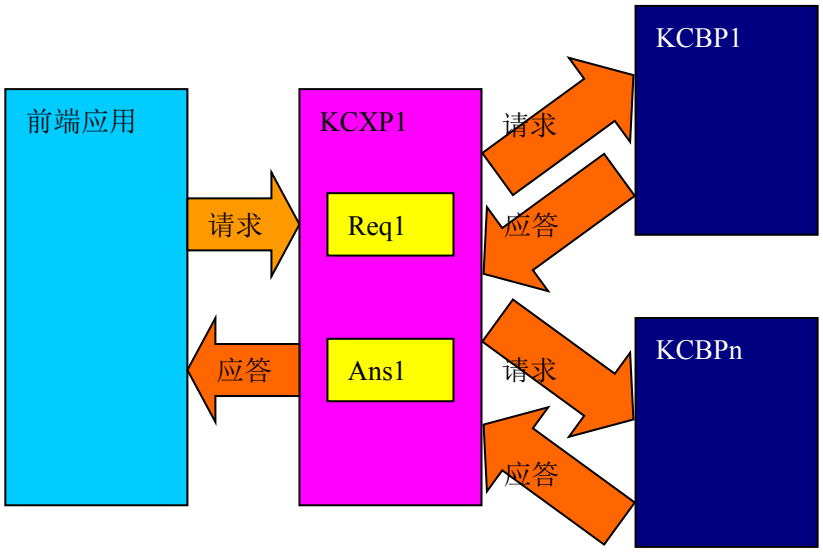
8.2.3 KCBP 动态负载均衡部署

这节中描述的 KCXP 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。

8.2.3.1 单级单 KCXP+多 KCBP 动态负载均衡部署

示意图如下：



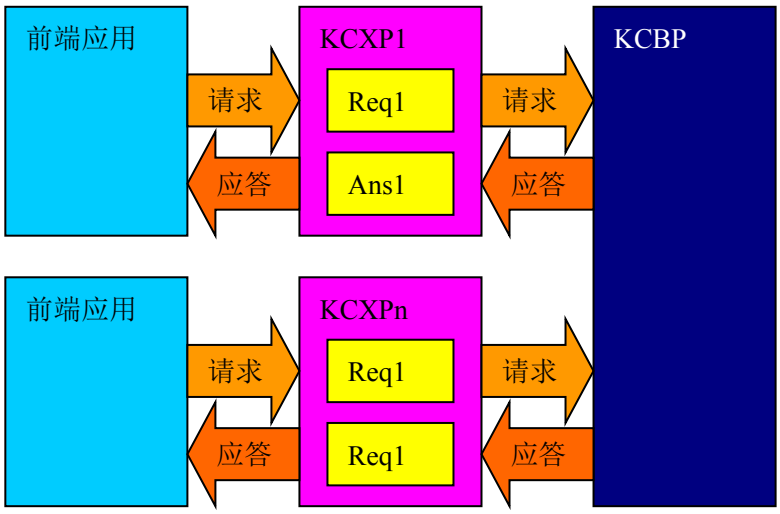
系统中存在多个功能相同的 KCBP 节点，这些节点连接同一个 KCXP 上，从同一个请求队列 Req1 中取请求，应答发送到 Ans1 上。

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。

KCXP 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。

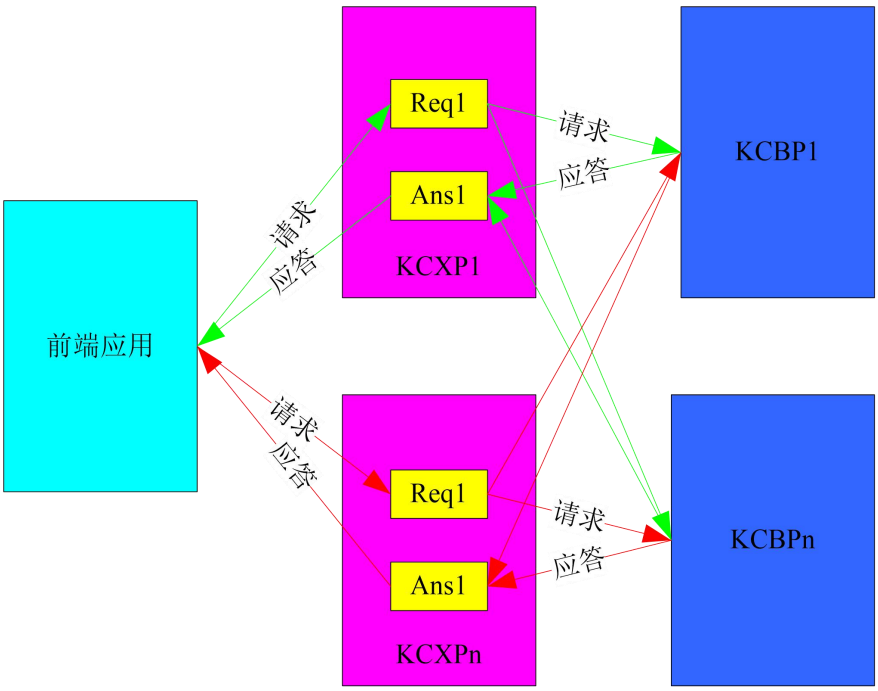
8.2.3.2 单级多 KCXP+单 KCBP 动态负载均衡部署



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。

8.2.3.3 单级多 KCXP+多 KCBP 动态负载均衡容错部署



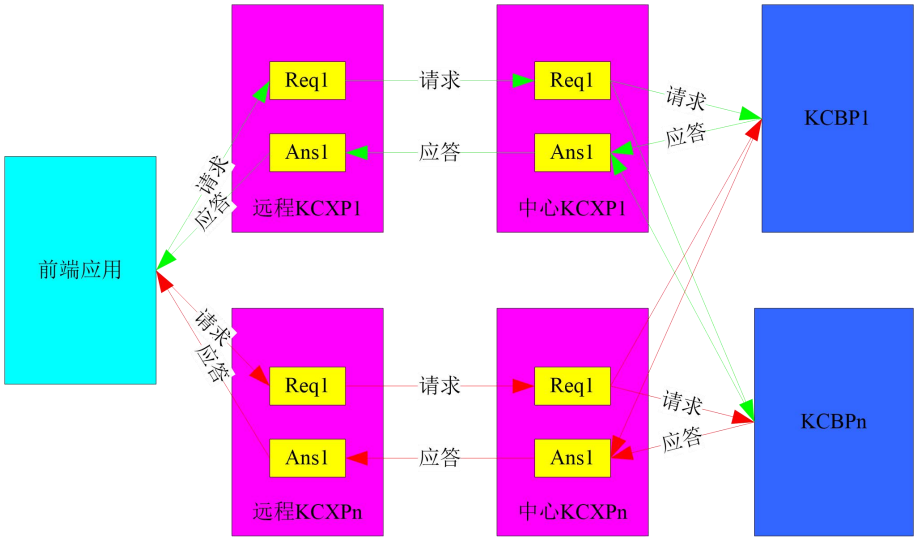
这种部署方法既可以容错，也可以动态负载均衡，总的来讲，这是一种理想的部署方法。

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。

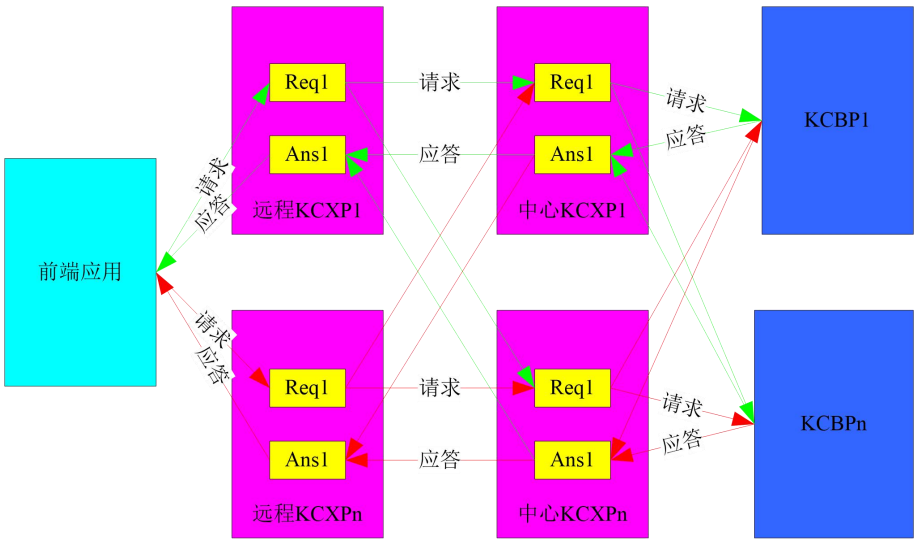
各 KCBP 节点是功能对等的，他们对外提供相同的服务。每个 KCBP 可以从多个 KCXP 并行获取请求，并行处理。

8.2.3.4 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。

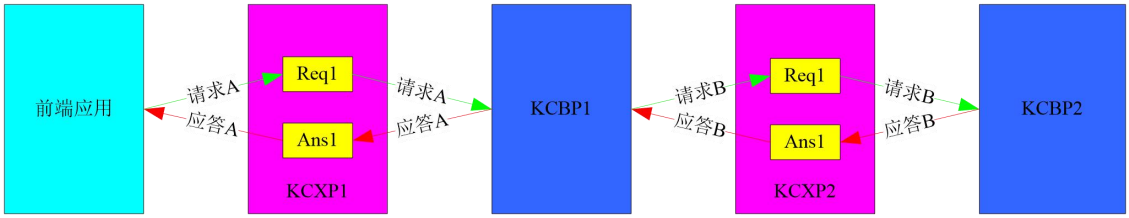
8.2.3.5 多级 KCXP+多 KCBP 网状动态负载均衡容错部署



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。

8.2.4 KCBP 转发部署

8.2.4.1 一对一转发部署



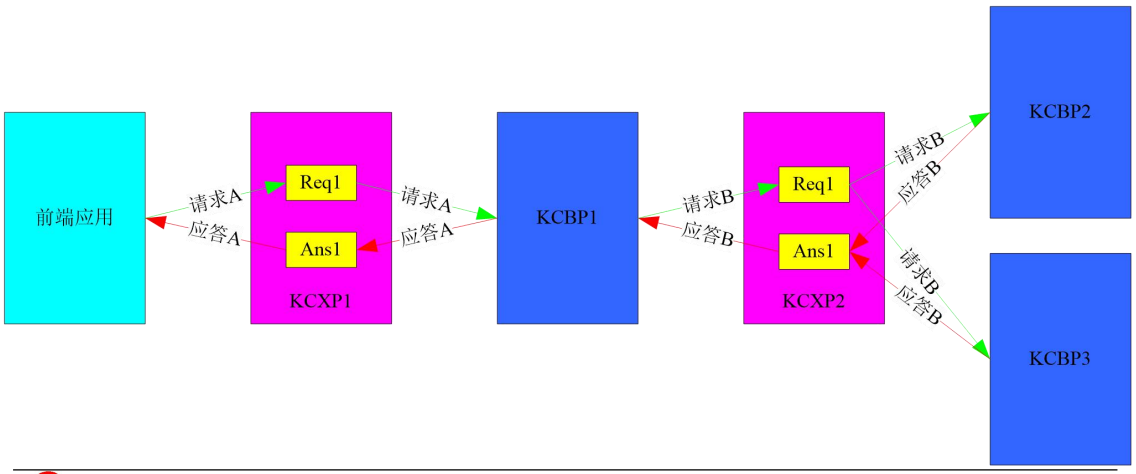
上图中，KCBP 节点 1 和 KCBP 节点 2 完成不同的功能。

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。

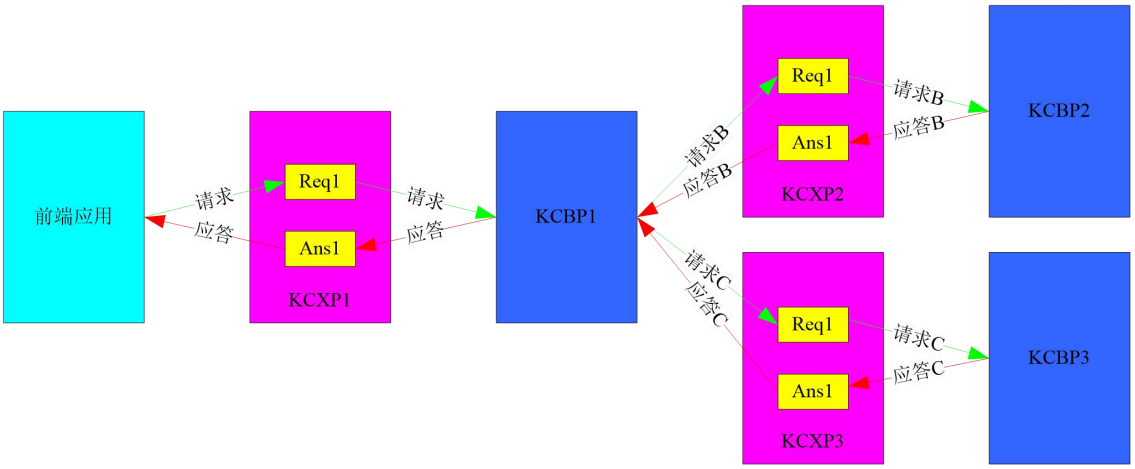
如果 KCXP1 和 KCXP2 是同一个 KCXP，KCBP1 和 KCBP2 需要使用不同的请求/应答队列。

8.2.4.2 一对多转发部署 1



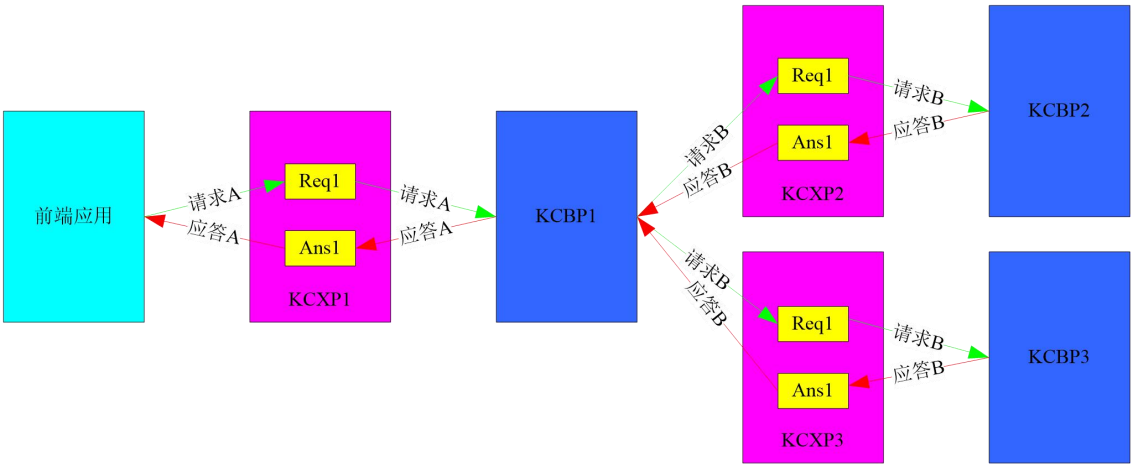
上图中 KCBP2 与 KCBP3 完成相同的功能，组成一个集群，对外统一提供服务。对于 KCBP1 来讲，它代理的是后面这个集群的业务。

8.2.4.3 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。

8.2.4.4 一对多转发部署 3

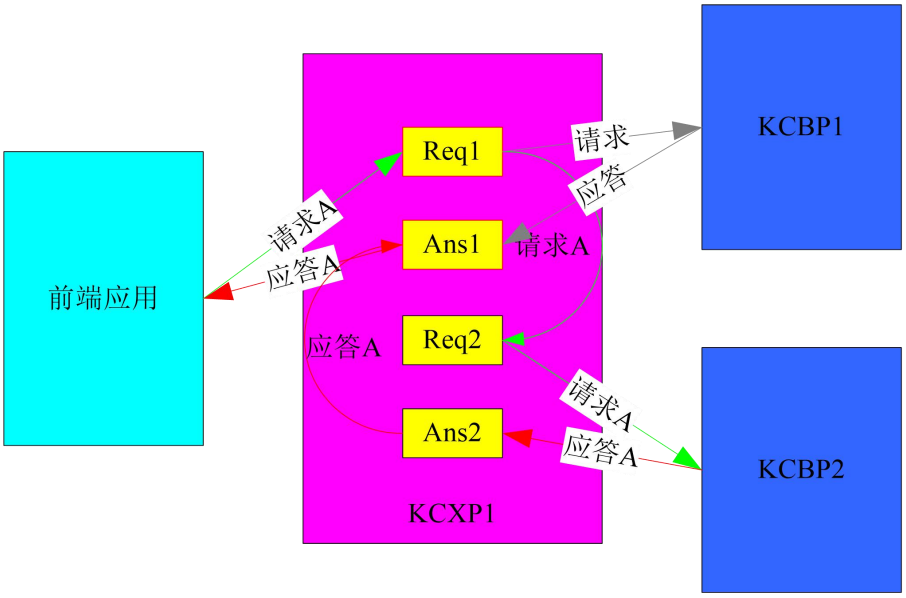


KCBP2 上的业务和 KCBP3 上的业务内容实现不同（比如访问不同的数据库），名称皆为 B，KCBP1 代理了 KCBP2 和 KCBP3 上的业务 B，

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。

8.2.5 KCXP 的消息重定向

KCXP 插件也可以控制消息的转发。



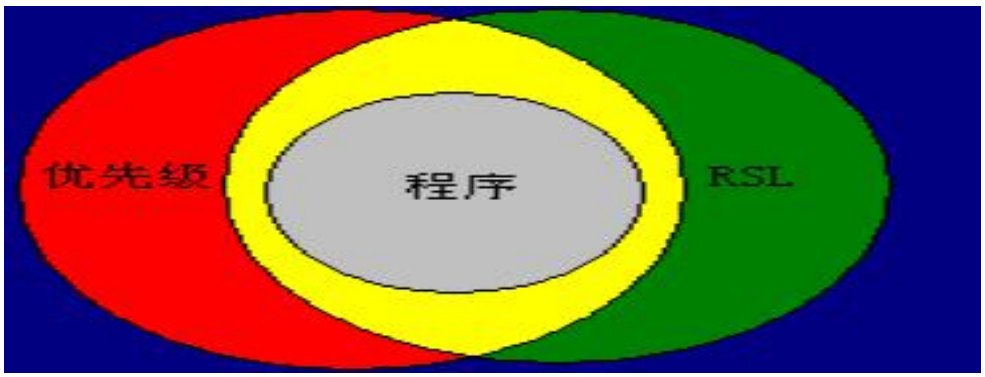
KCXP 的 Exit.ini 中可配置转发条件，能实现按各种条件设置。上图中，前端应用发起的请求 A 送到 Req1 时，被插件重定向到 Req2 交给 KCBP2 处理，应答被插件由 Ans2 重定向到 Ans1 再返回给前端应用。

8.3 LBM 并发控制

为了防止个别执行时间长的 LBM 耗尽 AS 进程，需要限制 LBM 的并发数，达到提高系统的整体可用性的目的，KCBP 提供以下几种并发控制方法：

- 按服务名称限制并发数
- 按 RSL 级别限制并发数
- 按 Priority 限制并发数

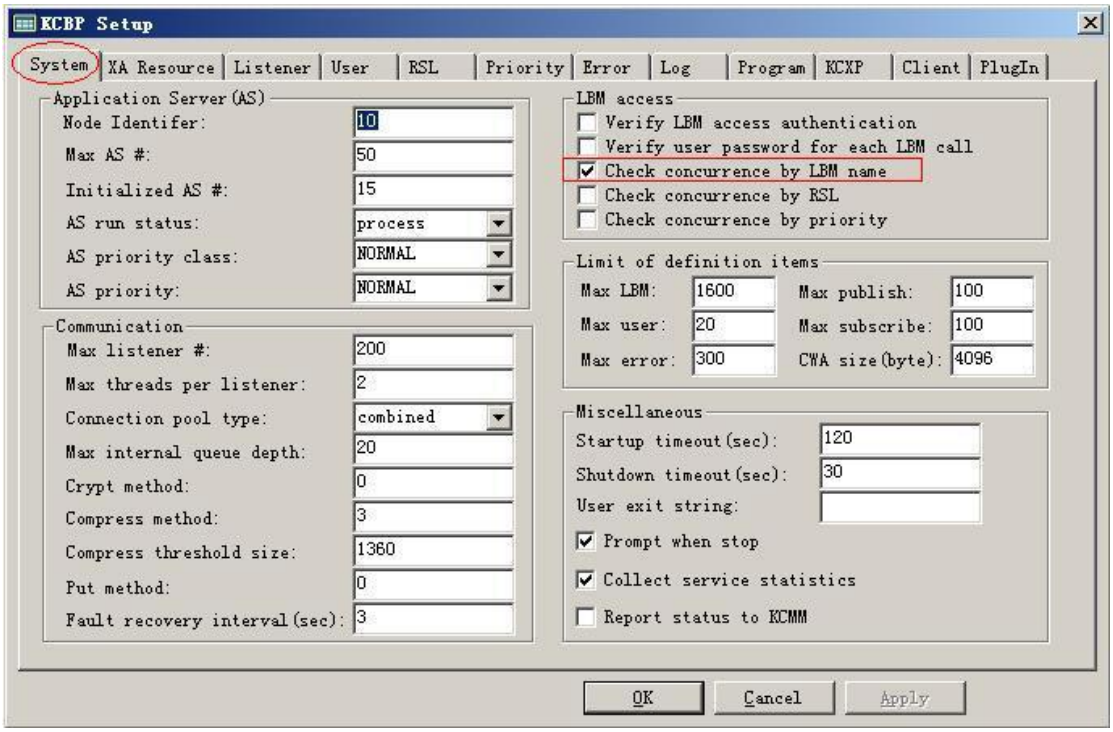
各并发控制方式的的控制范围关系图如下：

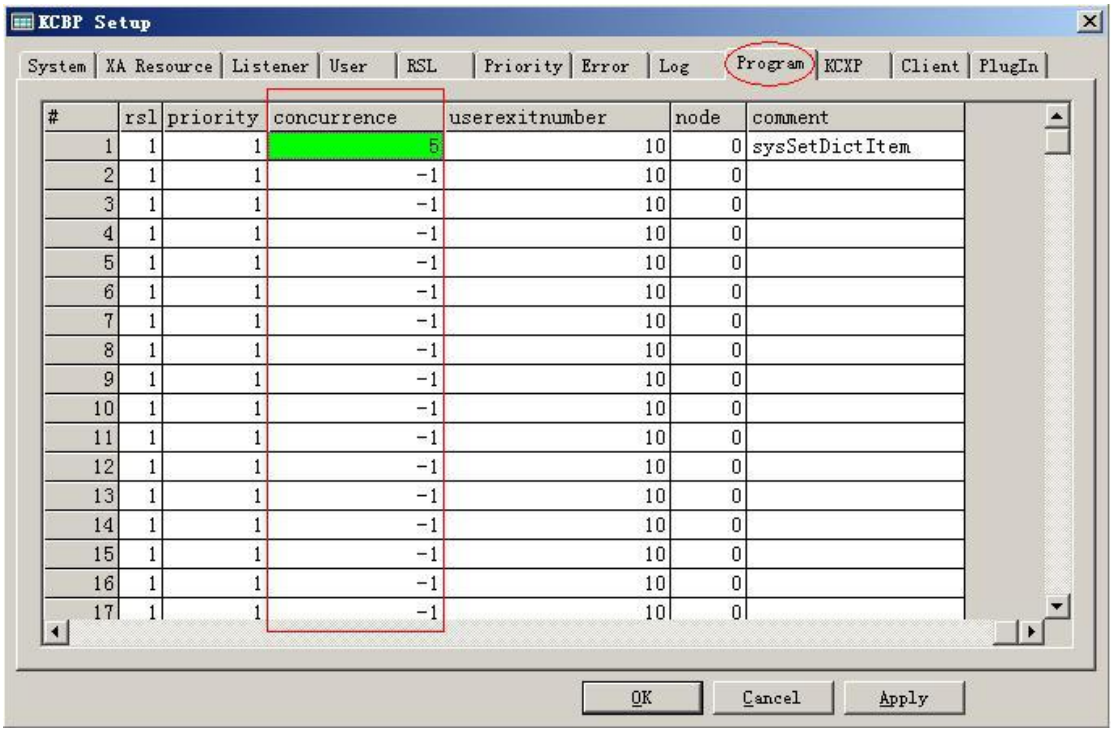


由上图可见，按服务名称控制的作用范围是按优先级控制和按 RSL 控制作用范围的交集。

8.3.1 按服务名称限制并发数

这种方式实现最小粒度的并发限制方法。需要用 KCBPSetup 做如下配置(注意红框部分)：



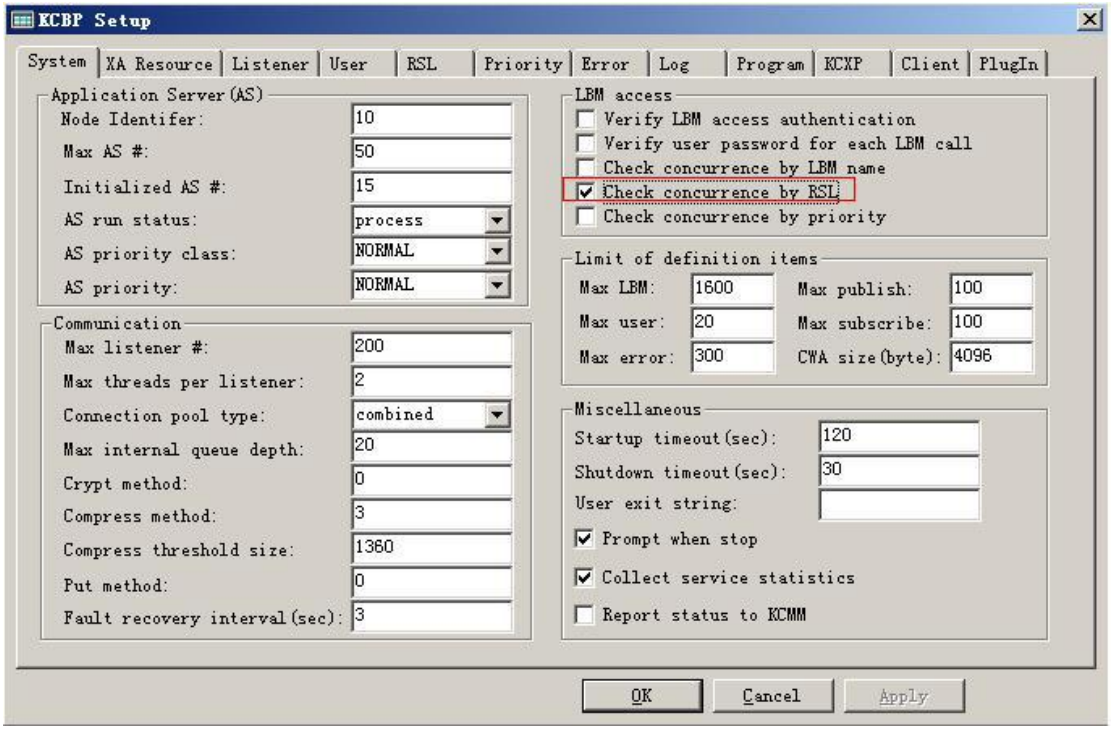


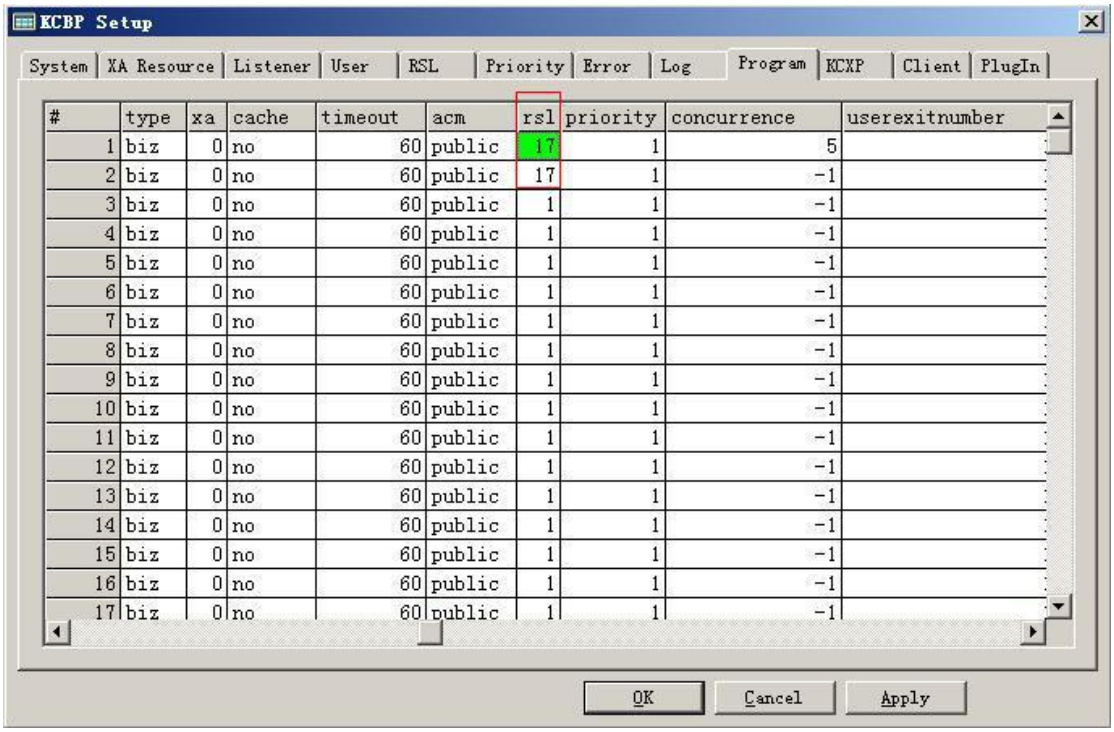
#	rsl	priority	concurrence	userexitnumber	node	comment
1	1	1	5	10	0	sysSetDictItem
2	1	1	-1	10	0	
3	1	1	-1	10	0	
4	1	1	-1	10	0	
5	1	1	-1	10	0	
6	1	1	-1	10	0	
7	1	1	-1	10	0	
8	1	1	-1	10	0	
9	1	1	-1	10	0	
10	1	1	-1	10	0	
11	1	1	-1	10	0	
12	1	1	-1	10	0	
13	1	1	-1	10	0	
14	1	1	-1	10	0	
15	1	1	-1	10	0	
16	1	1	-1	10	0	
17	1	1	-1	10	0	

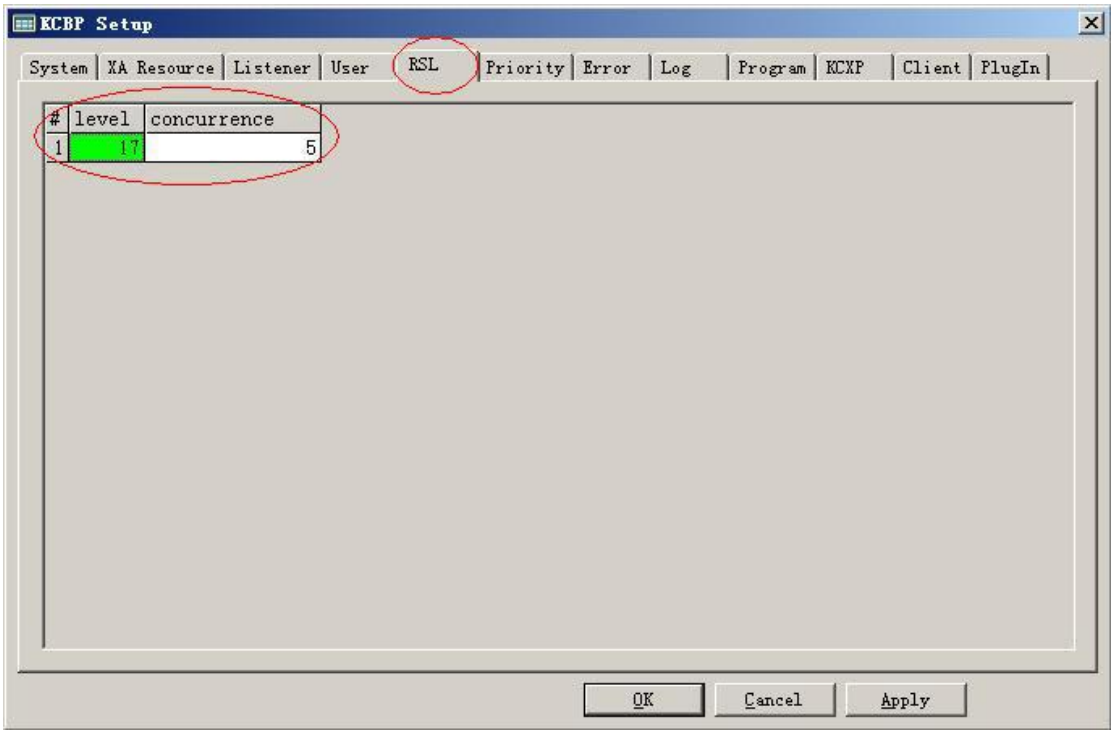
注意：concurrence 为-1 时不限制并发数,为 5 时限制并发数为 5 个。

8.3.2 按 RSL 级别限制并发数

按 RSL 控制并发数可以实现分组并发控制，拥护可以几个 LBM 设置成相同的 RSL,然后根据 RSL 控制并发数。下面举例说明如何将 RSL 为 17 的 LBM 并发数限制为 5 个。使用 KCBPSetup 配置如下(注意红框部分)：

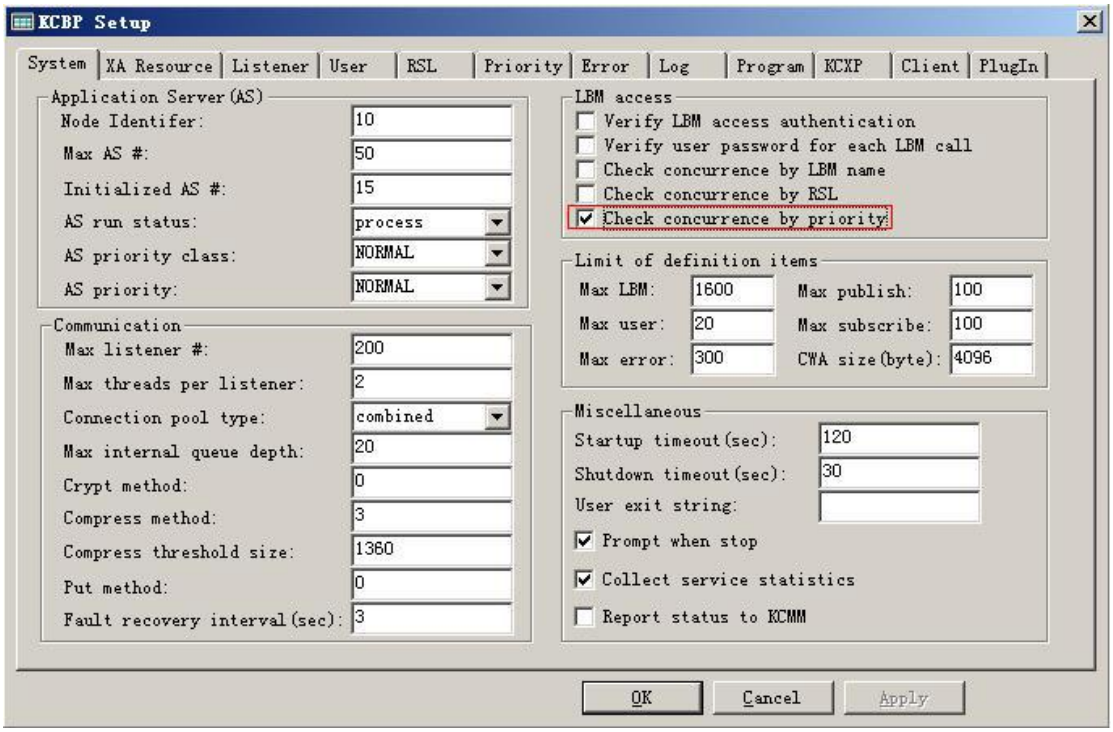


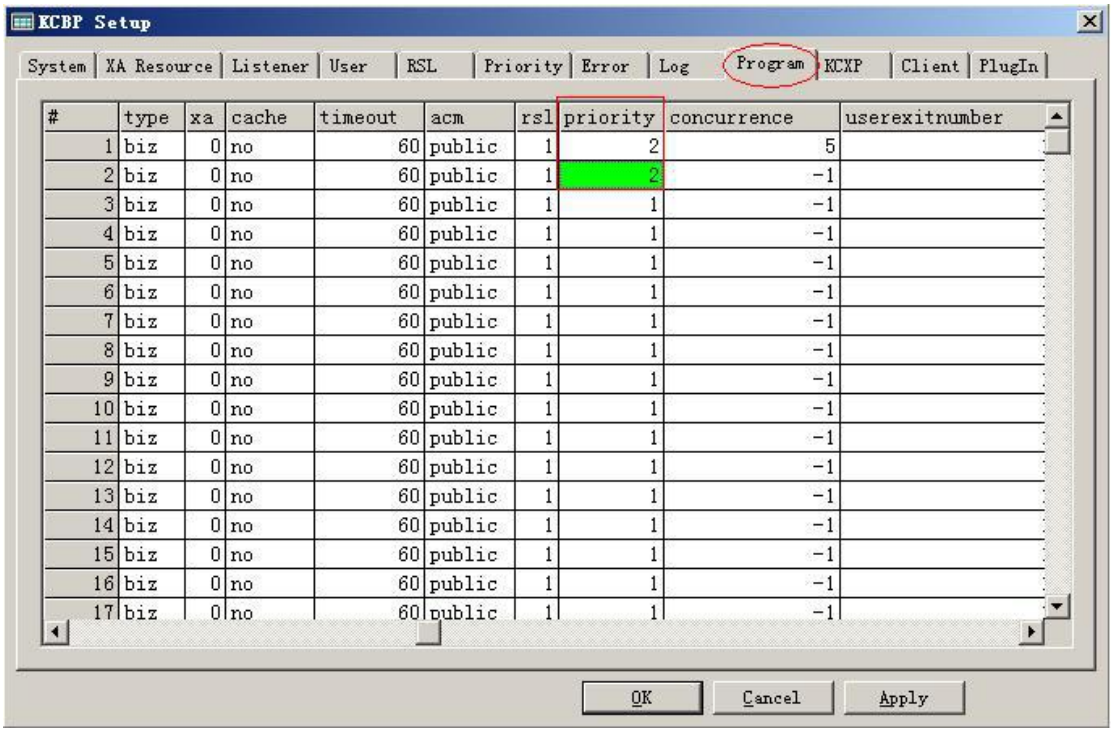


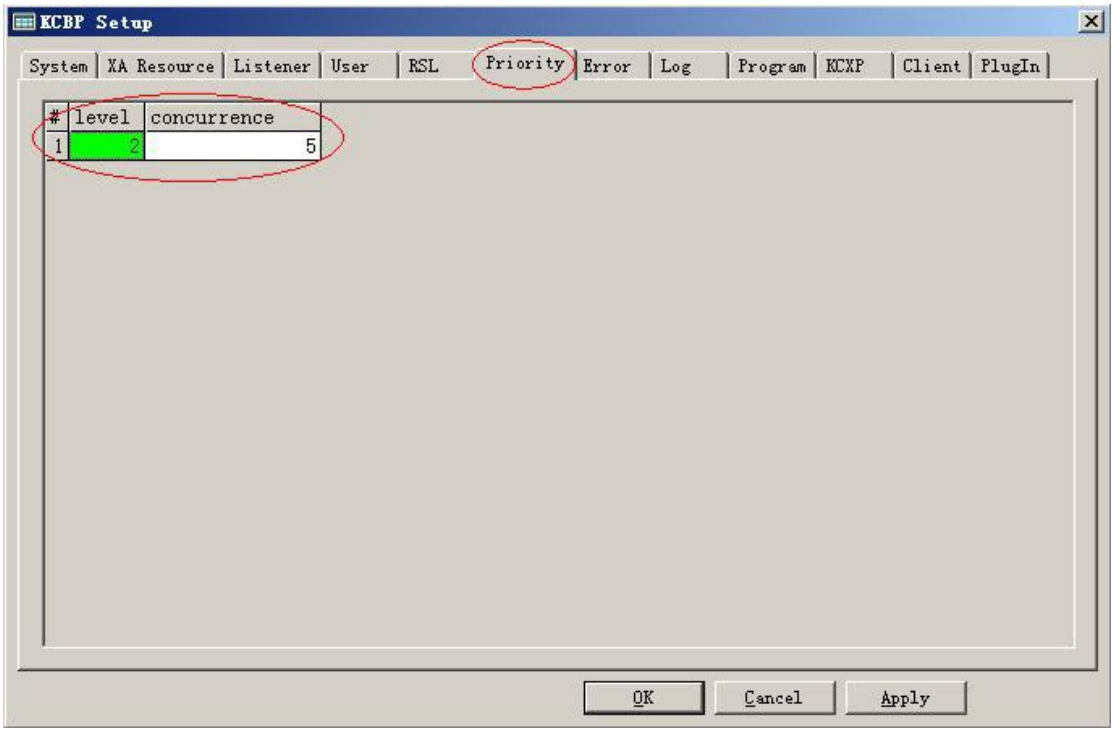


8.3.3 按 Priority 限制并发数

按 Priority 控制并发数提供了按 RSL 控制以外的一种分组并发控制方法。下面举例说明如何将 Priority 为 2 的 LBM 并发数限制为 5 个。使用 KCBPSetup 配置如下(注意红框部分)：


KCBP Setup
System
XA Resource
Listener
User
RSL
Priority
Error
Log
Program
KCXP
Client
PlugIn
Application Server(AS)
Node Identifer: 10
Max AS #: 50
Initialized AS #: 15
AS run status: process
AS priority class: NORMAL
AS priority: NORMAL
Communication
Max listener #: 200
Max threads per listener: 2
Connection pool type: combined
Max internal queue depth: 20
Crypt method: 0
Compress method: 3
Compress threshold size: 1360
Put method: 0
Fault recovery interval(sec): 3
LBM access
Verify LBM access authentication
Verify user password for each LBM call
Check concurrence by LBM name
Check concurrence by RSL
Check concurrence by priority
Limit of definition items
Max LBM: 1600
Max publish: 100
Max user: 20
Max subscribe: 100
Max error: 300
CWA size(byte): 4096
Miscellaneous
Startup timeout(sec): 120
Shutdown timeout(sec): 30
User exit string:
Prompt when stop
Collect service statistics
Report status to KCMM
OK
Cancel
Apply


KCBP Setup
System
XA Resource
Listener
User
RSL
Priority
Error
Log
Program
KCXP
Client
PlugIn
type xa cache timeout acm rsl priority concurrence userexitnumber
1 biz 0 no 60 public 1 2 5
2 biz 0 no 60 public 1 2 -1
3 biz 0 no 60 public 1 1 -1
4 biz 0 no 60 public 1 1 -1
5 biz 0 no 60 public 1 1 -1
6 biz 0 no 60 public 1 1 -1
7 biz 0 no 60 public 1 1 -1
8 biz 0 no 60 public 1 1 -1
9 biz 0 no 60 public 1 1 -1
10 biz 0 no 60 public 1 1 -1
11 biz 0 no 60 public 1 1 -1
12 biz 0 no 60 public 1 1 -1
13 biz 0 no 60 public 1 1 -1
14 biz 0 no 60 public 1 1 -1
15 biz 0 no 60 public 1 1 -1
16 biz 0 no 60 public 1 1 -1
17 biz 0 no 60 public 1 1 -1
OK
Cancel
Apply

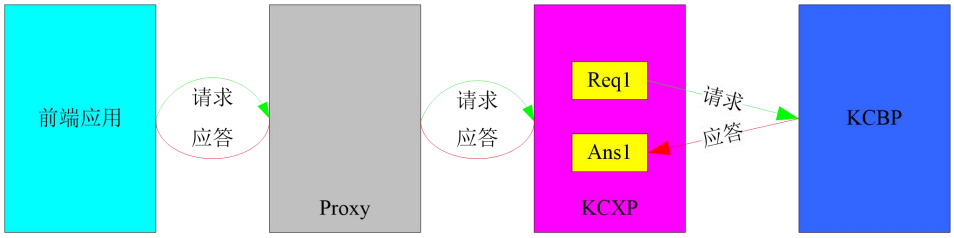


8.4 代理服务器和 SSL

8.4.1 PROXY

8.4.1.1 使用 PROXY 的 KCBP 应用系统关系图

下图的前端应用通过代理服务器连接 KCXP。

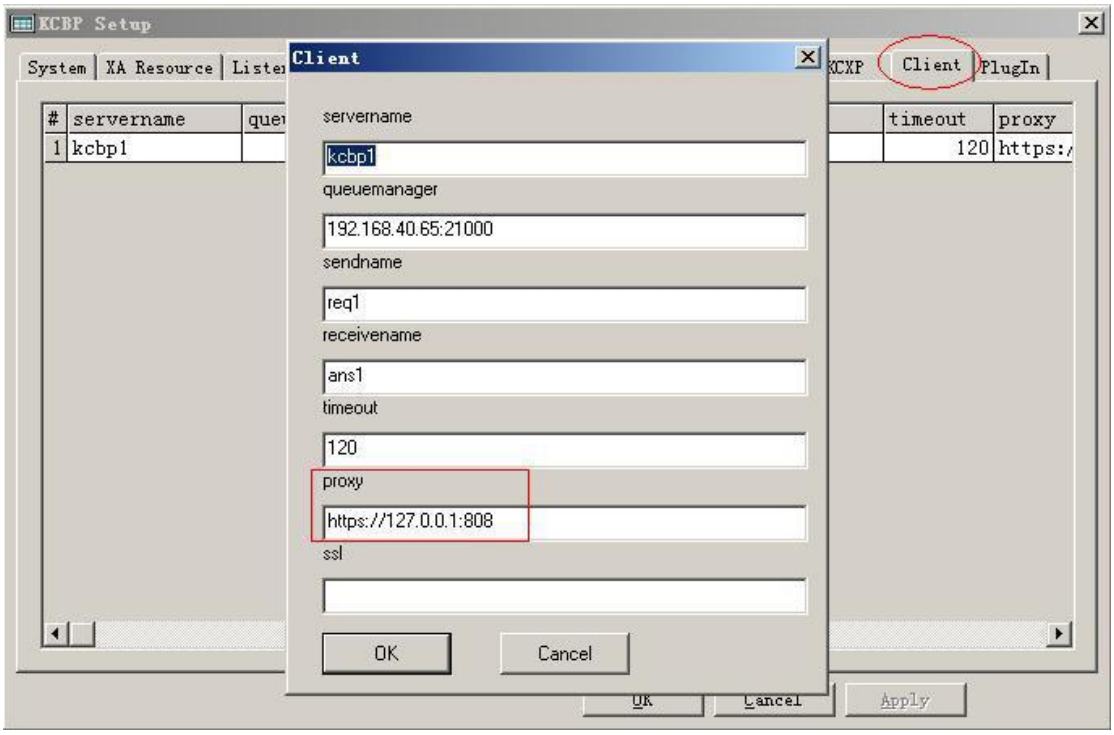


前端应用连接代理服务器可以使用 socks4、socks5、http、https 等协议，代理服务器采用格式 URL。下面我们介绍 kcbpcp 如何通过代理服务器和 KCXP 连接。

8.4.1.2kcbpcp 使用代理服务器

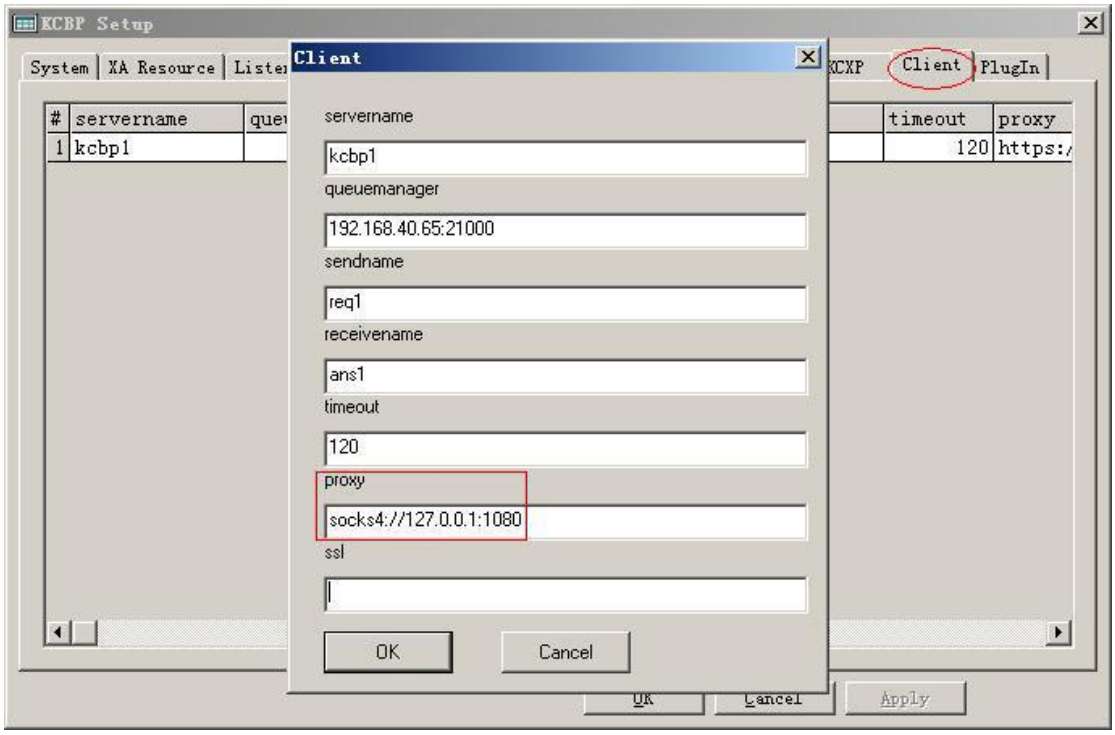
8.4.1.2.1 使用 HTTPS 协议连接代理服务器

KCBPSetup 设置界面如下（注意红框里的设置）：



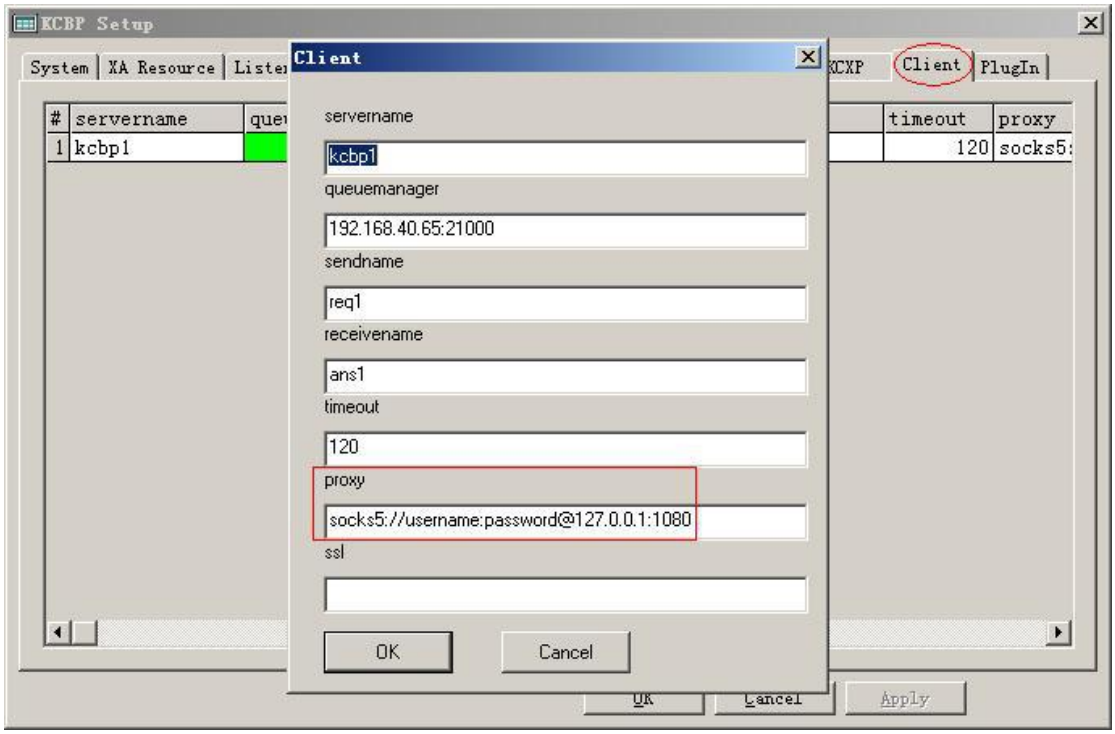
其中 https://127.0.0.1:808 是 https 代理服务器地址和端口。

8.4.1.2.2 使用 SOCKS4 协议连接代理服务器



其中 socks4://127.0.0.1:1080 是 socks4 代理服务器地址和端口。

8.4.1.2.3 使用 SOCKS5 协议连接代理服务器



其中 socks5://username:password@127.0.0.1:1080 是 socks5 代理服务器地址和端口。

8.4.2 SSL

8.4.2.1 设置 KCXP 支持 SSL

KCXP 可以对外提供 SSL 协议进行通讯。选择 KCXP 配置向导->手工配置，然后选择 Next，在 SSL 配置页面进行如下所示的设置。

其中 NetAddr1 设置的值内容为：

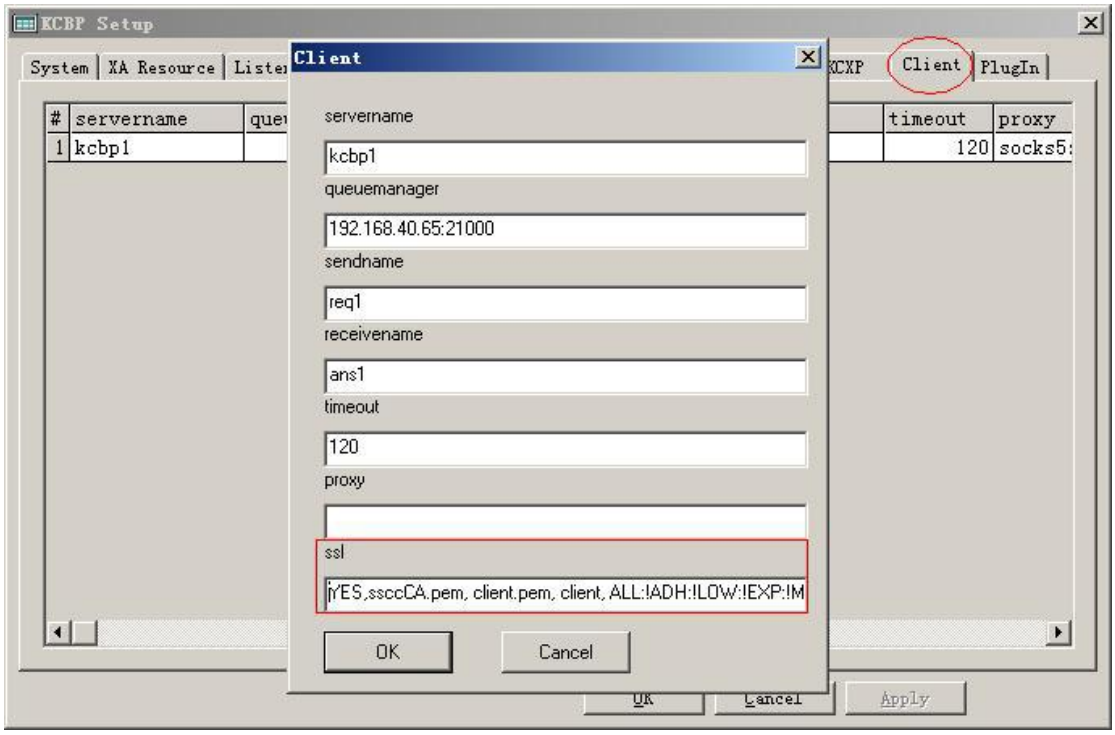
YES,ssccCA.pem,server.pem,123456,ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH,YES,NO

各项含义如下:状态,根证书,证书,密码,算法表,加密传输,主动发起握手

ssccCA.pem 是根证书，server.pem 是 KCXP Server 的证书，这两个文件由金证公司提供。

通过这个界面设置的内容保存在 kcxp 目录下的 kcxp.ini 中，用户也可编辑这个文件。

8.4.2.2设置 kcbpcp 使用 SSL



SSL 内容为：
YES,ssccCA.pem, client.pem, client, ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH, YES, YES
各项含义如下:状态,根证书,证书,密码,算法表,加密传输,主动发起握手
ssccCA.pem 是根证书，client.pem 是 kcbpcp 使用的证书，都放在 kcbpcp 所在目录。

8.4.3 注意事项

8.4.3.1效率问题

使用 SSL 将导致通讯效率大幅下降，从效率的角度考虑，KCBP 和 KCXP 之间通讯不建议使用 SSL，KCXP 和客户端之间是否使用 SSL 需要根据应用的实际情况来决定。在 KCXP 上可以设置多个侦听地址，每个地址分别设置各自的 SSL 选项。

8.4.3.2编程时如何使用 SSL 和 PROXY

KCBPCLI.h 中定义了连接参数结构 tagKCBPConnectOptionEx：

```
#define KCBP_PROXY_MAX                  128
#define KCBP_SSL_MAX                    256
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];
    char szProxy[KCBP_PROXY_MAX + 1];
    char szSSL[KCBP_SSL_MAX + 1];
}
tagKCBPConnectOptionEx;
```

用户在编程时可以在 szProxy 和 szSSL 设置所需内容。

8.5 性能调优

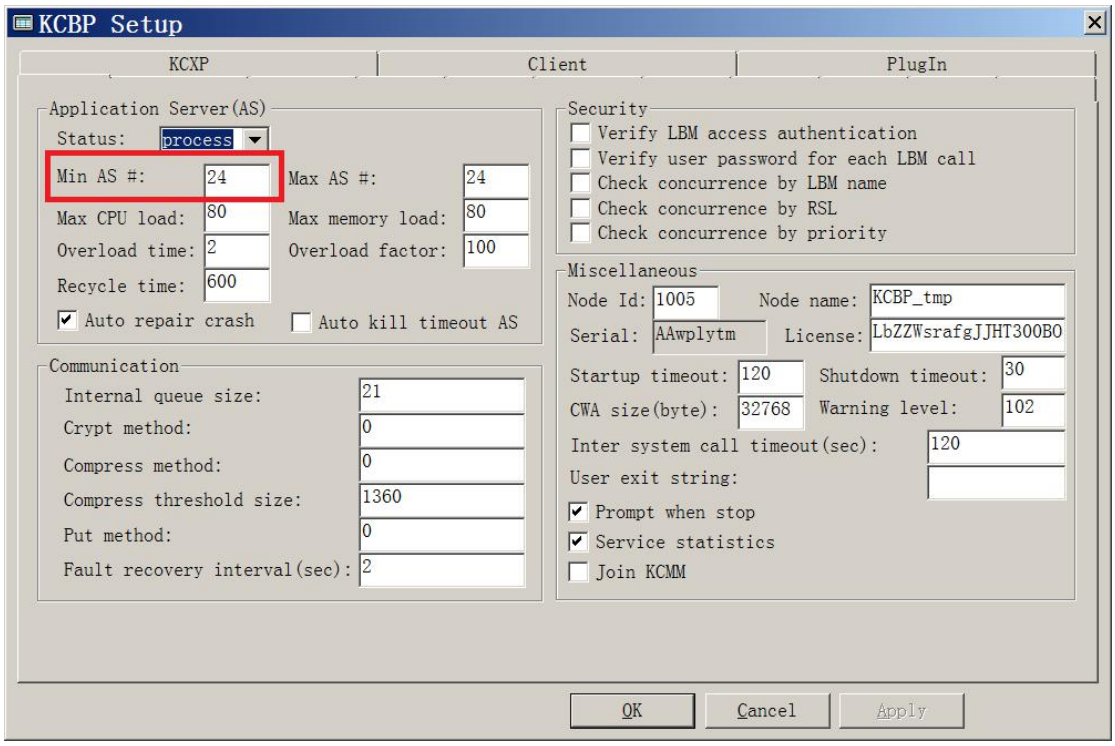
8.5.1 主要因素

KCBP 中对性能影响最大的参数有下面几个：

- AS 进程数目
- 压缩方式和压缩阀值
- 日志显示级别和记录级别
- Listener 线程数目
- LBM cache 设置

下面我们分别介绍这几个参数。

8.5.1.1AS 进程数目

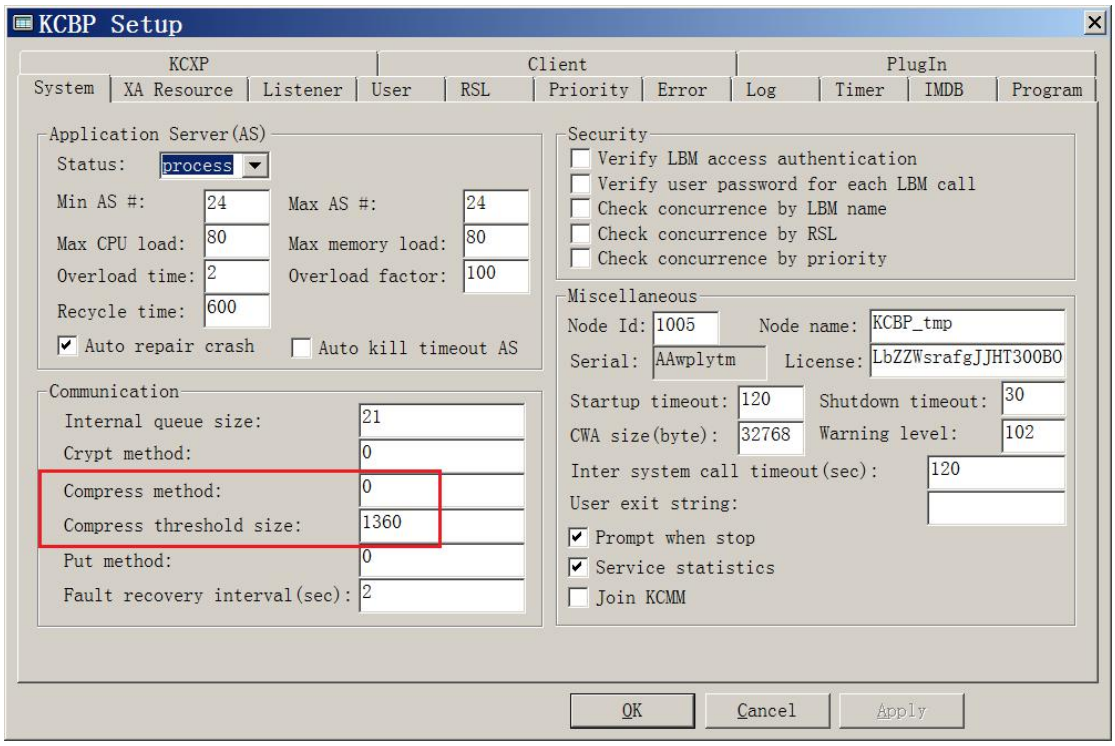


上图中红框内的内容便是 AS 进程数目。这个参数用来控制系统中运行的应用服务处理进程（AS 进程）的数目。

对于系统总体性能来讲，一台机器上的 AS 进程数目需要有一个**相对最优值**。注意这里强调的是最优值，而不是最大值，这是因为系统中的 AS 进程数目并不是越多越好。理论上来讲，如果系统中运行的服务程序不操作数据库，那么当系统中 AS 进程的数目是 CPU 个数的两倍时，系统的运行效率较高；如果系统中运行的服务程序操作数据库，而操作数据库会发生网络交互，网络交互的效率比本地操作的时间要长，AS 进程就可能产生等待，因此，这时为了提高系统的整体处理效率，需要增加 AS 进程的并发数目。至于 AS 进程数目的最优值究竟是多少，需要用户根据自己系统的实际情况去反复测试才能确定。确定最优值可以采用压力测试的方法，用 KCBPTest 进行测试，当调用同一个 LBM 时，使 KCBP 机器的 CPU 达到满负荷的最小 AS 进程数目便是这个 LBM 需要的 AS 进程数目最优值。如果有多个 LBM，需要评估每个 LBM 调用的比例，然后在 KCBPTest 的测试脚本中按比例生成测试数据，能够使 KCBP 机器的 CPU 达到满负荷的最小 AS 进程数目便是 AS 进程数目的最优值。当 KCBP 采用 Cluster 方式部署时，每个 KCBP 节点的最佳 AS 进程数目需要单独确定。KCBPTest 的用法请参阅 KCBP 用户手册。

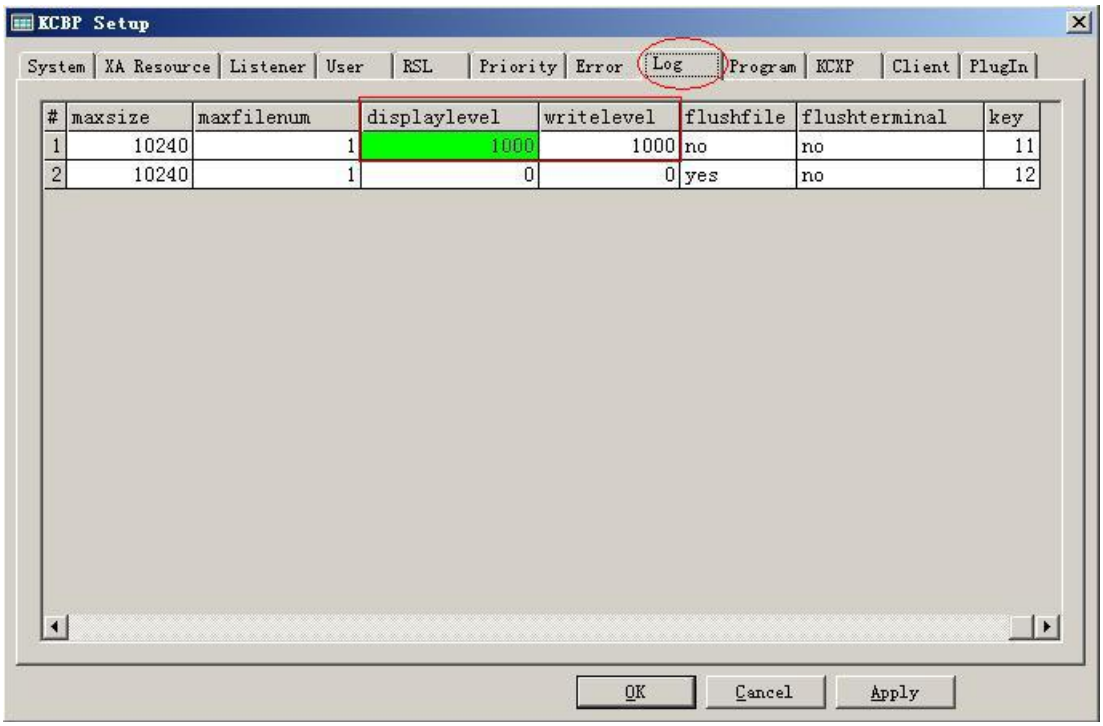
另外需要注意一点，AS 进程的数目在界面中是 Initialize AS#，不是 Max AS#。Max AS#只用来控制资源分配，并不是实际的 kcbpas 进程数目。kcbpas 进程的数目可以在任务管理器中查看到。

8.5.1.2压缩方式和压缩阀值



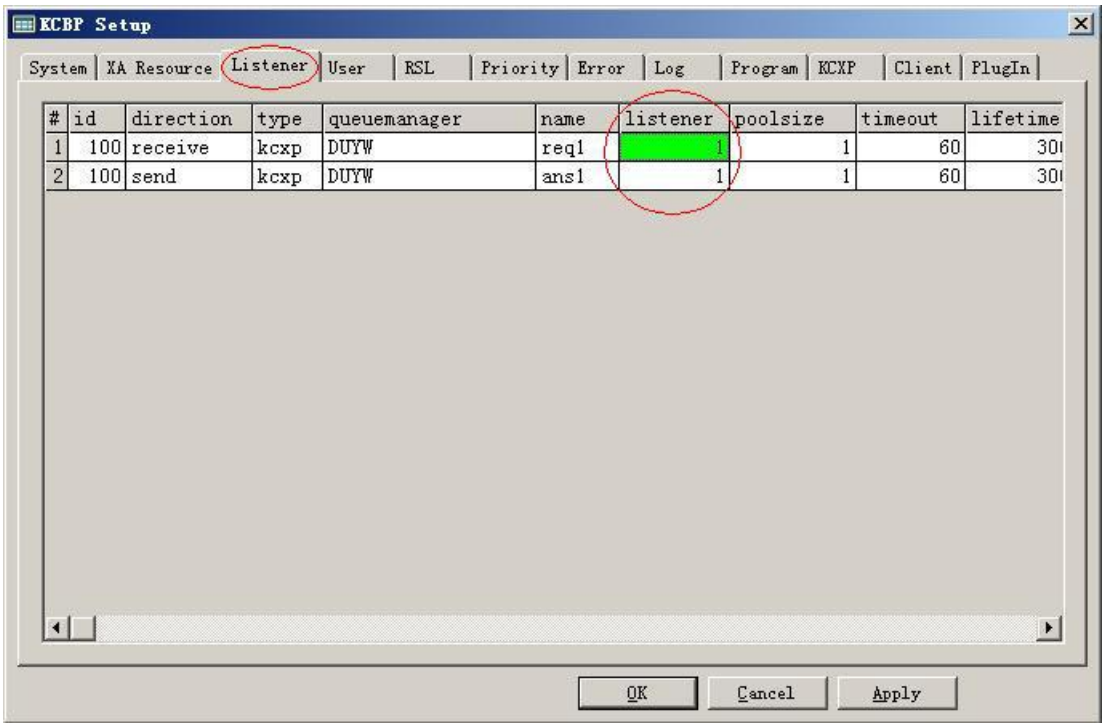
上图中红框内的内容便是压缩方式和压缩阀值。压缩数据可以减小通讯报文的尺寸，但会耗费 KCBP 机器的 CPU，据我们测试，当启用压缩时，会消耗 30%CPU。当压缩方式 Compress method 为 0 时，不压缩。对于局域网上的通讯，由于网络通讯速率快，一般不用压缩。而对于广域网，由于网络通讯速率慢，需要启用压缩。KCBP 可以控制启用压缩的阀值条件，对于长度小于阀值条件的报文，网络传输可能不是瓶颈，因此没必要浪费 CPU 进行压缩，而对于长度大于阀值条件的报文，网络传输可能成为瓶颈，因此需要压缩，这时耗费一些 CPU 也是值得的。当系统中的报文为大报文比例相对多时，压缩也会频繁，KCBP 机器的 CPU 可能变成瓶颈，这时可以增加 KCBP 处理节点，提高系统的处理能力。

8.5.1.3日志显示级别和记录级别



上图中红框内的内容便是日志的显示级别和写级别。一般来讲，日志级别越低，显示的日志越详细。日志级别为 1000 时，只显示错误日志；日志级别为 100 时，可以显示 LBM 调用日志和警告日志；日志级别为 99 时，可显示输入参数；日志级别为 98 时，可显示输出参数；日志级别为 0 时，显示所有日志，这时系统的处理效率会明显下降。生产系统日志显示级别和写级别都设置为 1000。

8.5.1.4Listener 线程数目



上图中红框内的内容便是请求输入队列的侦听线程数目。

这个参数有三点需要注意：

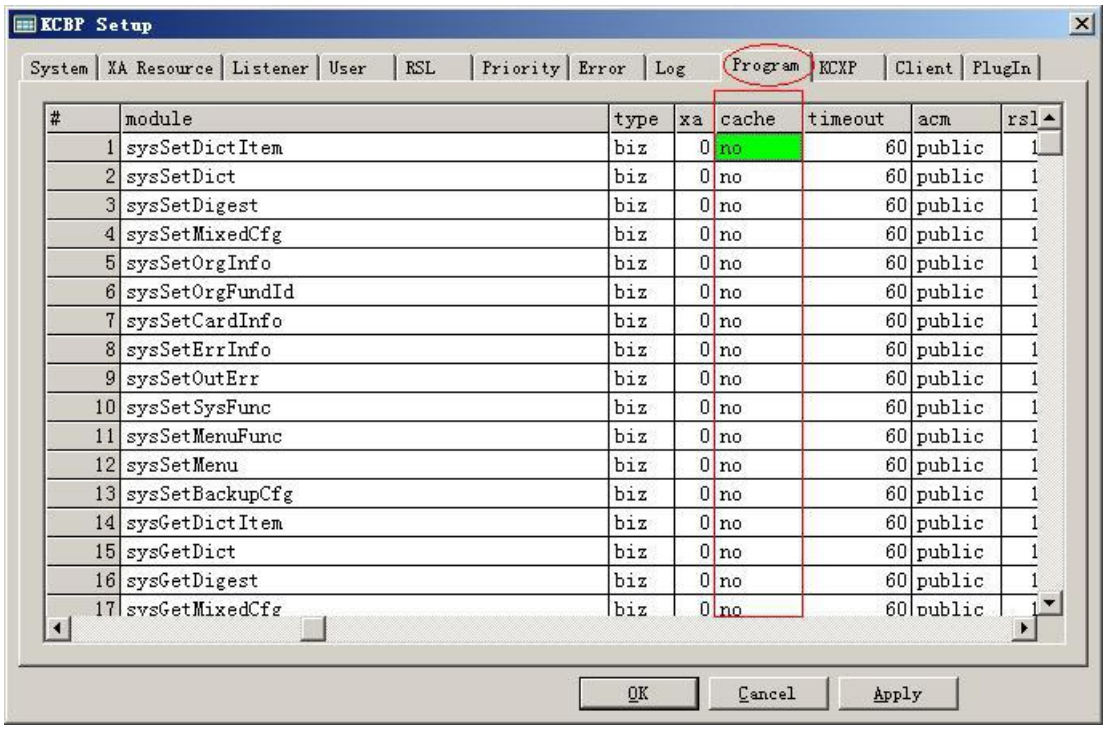
第一点是在压力测试时，当请求都来自同一个队列，这时 receive 队列的侦听线程数目是瓶颈，这就需要调整 receive 队列的侦听线程数目。一般来讲，在 P5 2.4GCPUd 的机器上，1 个侦听线程每秒可以接收请求 800 笔（这个数据和 KCXP 处理性能相关），这时将 listener 设置为 5 个可以取得较高的处理效率。

第二点是在生产系统上，当请求来自不同的队列，这时 KCBP 内部请求缓存队列是瓶颈，特别是当每个队列的请求数目差别较大时，需要适当增加请求多的队列的侦听线程数目。增加一个队列的 Listener 数目，就可以增加这个队列的请求进入 KCBP 内部缓存队列的概率，也就能增加这个队列的请求被 AS 进程处理的概率，达到提高这个队列的处理效率的目的。

第三点是注意请求队列的 Listener 数目要小于 System->max thread per listener 的数目。

此外，还要说明，send 队列的 listener 数目是没有实际意义的，这是 KCBP 的设计。

8.5.1.5LBM cache 设置



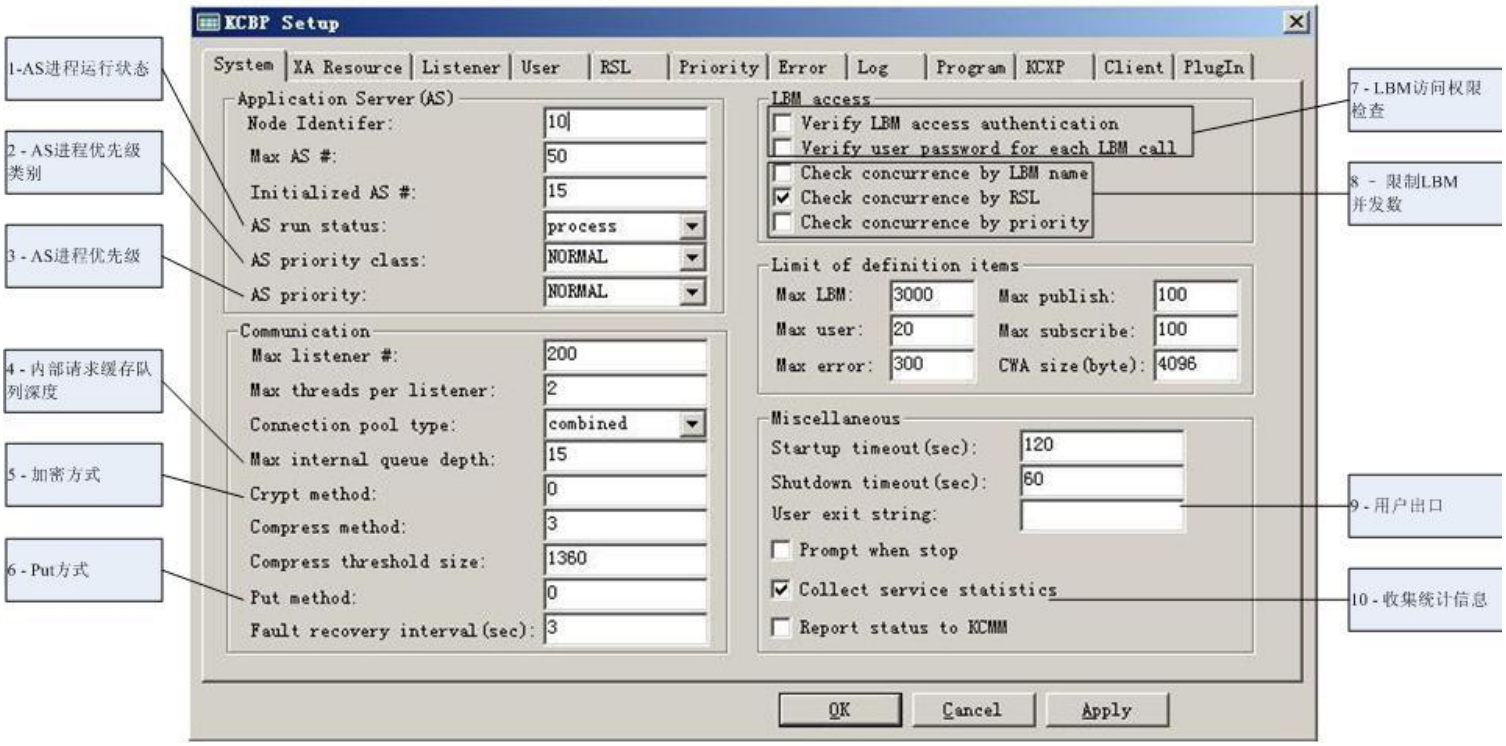
#	module	type	xa	cache	timeout	acm	rsl
1	sysSetDictItem	biz	0	no	60	public	1
2	sysSetDict	biz	0	no	60	public	1
3	sysSetDigest	biz	0	no	60	public	1
4	sysSetMixedCfg	biz	0	no	60	public	1
5	sysSetOrgInfo	biz	0	no	60	public	1
6	sysSetOrgFundId	biz	0	no	60	public	1
7	sysSetCardInfo	biz	0	no	60	public	1
8	sysSetErrInfo	biz	0	no	60	public	1
9	sysSetOutErr	biz	0	no	60	public	1
10	sysSetSysFunc	biz	0	no	60	public	1
11	sysSetMenuFunc	biz	0	no	60	public	1
12	sysSetMenu	biz	0	no	60	public	1
13	sysSetBackupCfg	biz	0	no	60	public	1
14	sysGetDictItem	biz	0	no	60	public	1
15	sysGetDict	biz	0	no	60	public	1
16	sysGetDigest	biz	0	no	60	public	1
17	sysGetMixedCfg	biz	0	no	60	public	1

上图中红框内的内容便是 LBM 的 cache 属性。当 cache 为 yes 时，LBM 的调用效率会提高，这是因为，KCBP 预先把需要 cache 的 LBM 动态库装载到内存中，当调用这个 LBM 时，KCBP 直接使用内存中已经装载的 LBM，在 LBM 调用结束后也不释放它，这样就优化掉了磁盘操作，而大家都知道，内存操作比磁盘操作快，因此，效率就会提高。经我们测试，LBM cache 打开时（cache=”yes”），性能可提高 20%。

我们建议重视性能的用户在生产系统上打开 LBM 的 cache 开关。

8.5.2 次要因素

KCBP 中其他与性能相关的系统参数如下图所示：



8.5.2.1 AS 运行方式

AS 进程方式包括两种：process 和 thread。由于 Windows 操作系统对 Thread 的调度比 process 要快，因此 AS 运行在 thread 方式时性能要比运行在 process 时高，据我们测试，thread 方式的性能高大约 10%。

尽管 thread 方式性能高，但我们仍然建议生产系统以 process 方式运行，因为在 process 比 thread 方式强壮，不怕 LBM 崩溃，更安全可靠，如果 process 的性能是瓶颈，可以通过 KCBP Cluster 解决。

8.5.2.2 AS 进程优先级类别

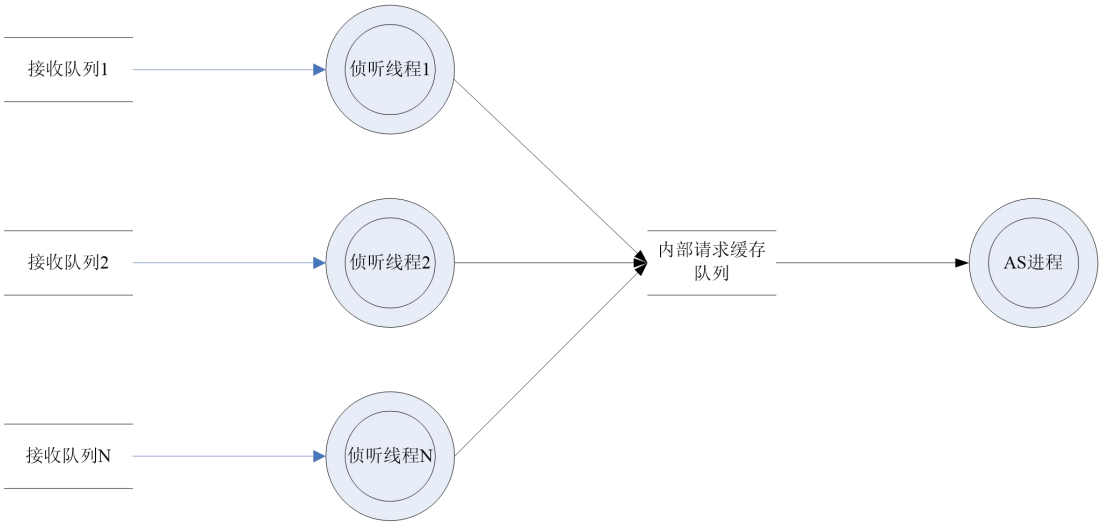
AS 进程优先级类别，当 KCBP 系统需要更高的优先级时可以进行调整。我们一般不建议调整。如果需要调整该参数，请先阅读 Windows 关于进程及线程优先级的技术资料。

8.5.2.3 AS 进程优先级

AS 进程优先级，和上面的参数配合使用，当 KCBP AS 线程需要更高的优先级时可以进行调整。我们一般不建议调整。如果需要调整该参数，请先阅读 Windows 关于进程及线程优先级的技术资料。

8.5.2.4 内部请求缓存队列深度

内部请求缓存队列用于缓存请求，它起到一个集线器（HUB）的作用。



如上图所示，在 KCBP 系统中，可以定义多组请求接收队列，每个请求接收队列都有一组侦听线程，侦听线程从请求接收队列中取请求，并送到内部请求缓存队列排队，等待 AS 进程处理。内部缓存队列的长度一般要大于等于 AS 进程的数目，这样，就能保证请求多时，充分利用 AS 进程。侦听线程组 1、侦听线程组 2 和侦听线程组 N 竞争内部请求缓冲队列资源，线程数目多的侦听线程组有更多机会竞争到内部请求缓冲队列的资源，请求传递速率相对较快，获得 AS 进程处理的概率就大，对外表现为处理性能高。

8.5.2.5 加密方式

加密方式可以有 0-3，其中 0 表示不加密，这时性能最高，1-3 分别表示用不同的算法加密，这时 KCBP 系统的处理效率会稍有下降，一般不会超过 2%。

8.5.2.6 Put 方式

Put 方式用来控制 KCBP 和 KCXP 通讯方法，缺省为 0，这时调用 KCXP_Put 函数通讯，当 Put 方式是 2 时，调用 KCXP_Put2 通讯。据我们测试，Put2 的效率比 Put 的效率大约高 20%。KCXP_Put2 虽然效率高，但有个缺点，就是当 KCXP 和 KCBP 失去连接再恢复时，KCXP_Put2 不能及时检查到连接断开，会造成每个 AS 进程丢失 1 笔应答；而 KCXP_Put 就没有这个问

题。

8.5.2.7 LBM 访问权限检查

当 LBM 访问权限检查打开时，KCBP 在处理 LBM 请求时，会进行 RSL 权限检查，这会降低系统的处理效率，一般不会超过 1%。

8.5.2.8 限制 LBM 并发数

当限制 LBM 并发数时，KCBP 在处理 LBM 请求时，会进行相关的处理，其中包含有进程间信号量的同步操作，这些操作会降低系统的处理效率，一般不超过 10%。至于设置并发处理进程小于 AS 进程数目，会对系统效率产生的影响，影响访问需要用户根据实际情况测定。

8.5.2.9 用户出口使用情况

当使用以下用户出口时，KCBP 的处理效率会受影响而下降，下降的幅度和用户出口的处理效率相关。

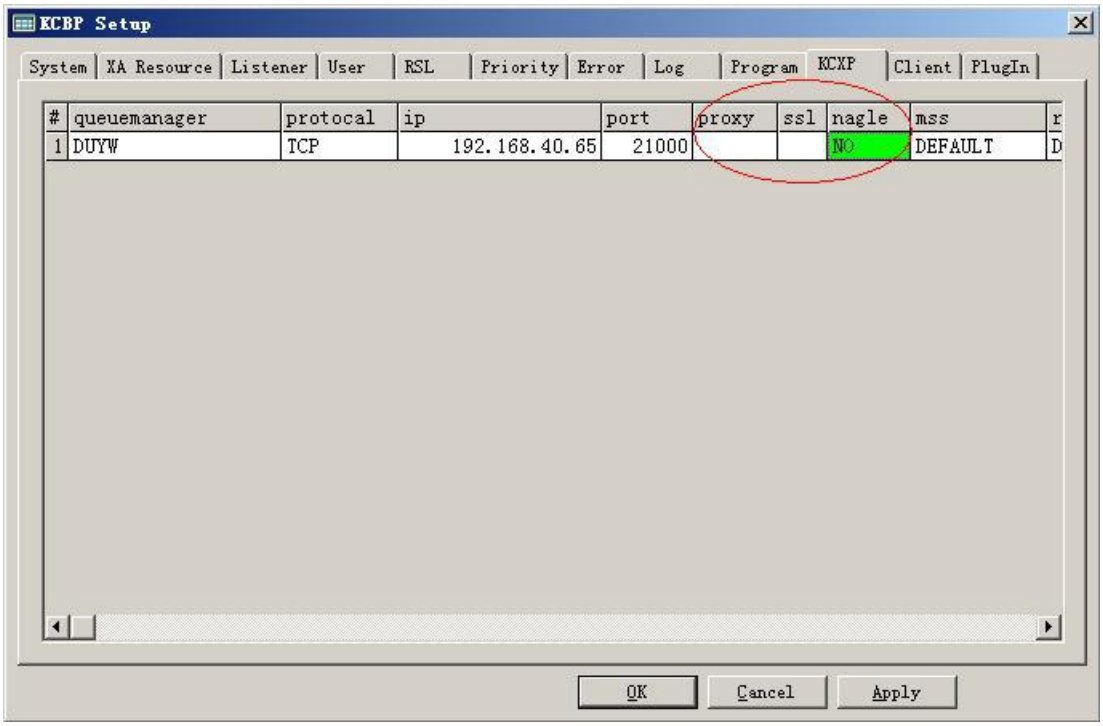
- KCBP_UE_ID_DUMPREQUEST，用户出口编号 4
- KCBP_UE_ID_DUMPANSWER，用户出口编号 23
- KCBP_UE_ID_DUMPLOG，用户出口编号 26
- KCBP_UE_ID_SYNCPOINT，用户出口编号 3
- KCBP_UE_ID_ENDTASK，用户出口编号 5

8.5.2.10 收集统计信息

当收集统计信息开关打开是，KCBP 会耗费一些 CPU 用来记录统计信息，处理效率会有所下降，下降幅度一般不会超过 10%。

8.5.3 其他因素

8.5.3.1 KCXP 设置



KCXP 对系统性主要有 Proxy、SSL、nagle 三个参数对性能有影响。
使用 Proxy 会造成系统地性能下降，降幅与 Proxy 的性能相关。
使用 SSL 会造成系统处理性能大幅度下降，降幅可能超过 50%。
Nagle 算法对通讯效率也有影响，当通过广域网通讯时，建议使用 Nagle 算法，当通过局域网通讯时，建议关闭 Nagle 算法。有关 Nagle 算法的更详细信息请参阅 TCP 相关变成资料。有一种情况需要说明：使用单线程测试压力时，当 Nagle 打开时，每笔请求的通讯开销会增加 100-200 毫秒；使用多线程测试压力时，当 Nagle 打开时，每笔请求的通讯开销不会增加和 Nagle 关闭时接近。

8.5.3.2 LBM 的执行效率

前面我们讲的性能因素都是围绕 KCBP 系统本身的，实际上，我们不能抛开 LBM 单独论述 KCBP 系统的处理效率，因为 LBM 是 KCBP 系统的重要组成部分，而 LBM 的效率高低直接决定 KCBP 系统的总体性能。
根据我们对已经建成的应用系统的分析，我们建议 LBM 编写时要注意减少与数据库交互次数，这样能降低网络传输因素对系统性能的影响，有利于提高系统性能。还有一点要明确：适当使用存储过程有利于提高系统效率，特别是系统中之调用最频繁的关键业务，可以考虑使用存储过程实现。
KCBP 系统提供了一个专用工具 LBMTest 用来测试 LBM 的执行效率，这个工具不通过 KCBP

直接执行 LBM，并能做压力测试。有关 LBMTest 的详细信息参见 LBMTest 使用说明。

8.6 用户出口

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
27	KCBP_UE_GetXAName	获取 XA 名称
28	KCBP_UE_IMDBSync	发出内存数据库同步指令
29	KCBP_UE_LBMTrigger	LBM 触发器，在 LBM 调用后执行
30	KCBP_UE_ChainLBM	另一个 LBM 触发器，作为上一个的补充

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

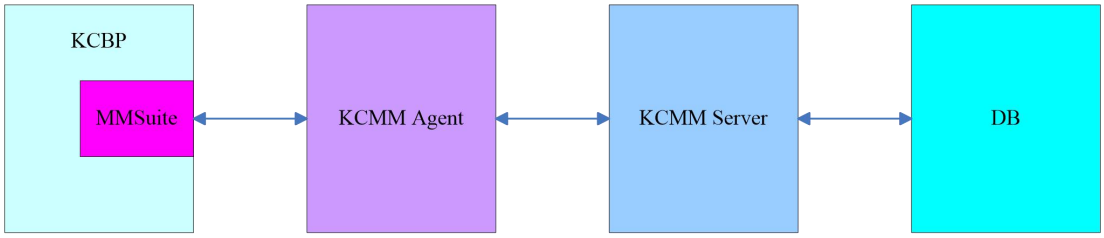
序，而 LBM 是用户编写的业务程序。由于用户出口是和 KCBP 自身的运行相关，因此，它的编写要求相对较高，如果用户出口程序编写质量不高，可能造成 KCBP 运行混乱。

在 UserExit.dll 中增加了关闭 KCBP 调用 shutdown.bat 的操作，与启动时调用 startup.bat 相对应。在 KCBP 界面 userexitstring 配置 1,2 之后，startup.bat 和 shutdown.bat 就被自动动用。举个例子，利用这两个批处理文件可以自动启动和关闭内存行情服务器。

有关用户出口的详细内容参见《KCBP 程序员手册》。

8.7 KCMM 监控

8.7.1 KCBP 与 KCMM 系统关系图



上图可见，KCBP 通过调用 mmsuite.dll 与 KCMMAgent 通讯，KCMM Agent 与 KCMM Server 通讯，监控参数存放到数据库中。KCMM Agent 安装在 KCBP 所在的机器上，mmsuite.dll 是 KCMM Agent 中的一个文件，安装在 windows 系统的 system32 目录。

KCBP 提供以下四个 bool 型监控参数，true 表示正常，false 表示异常：

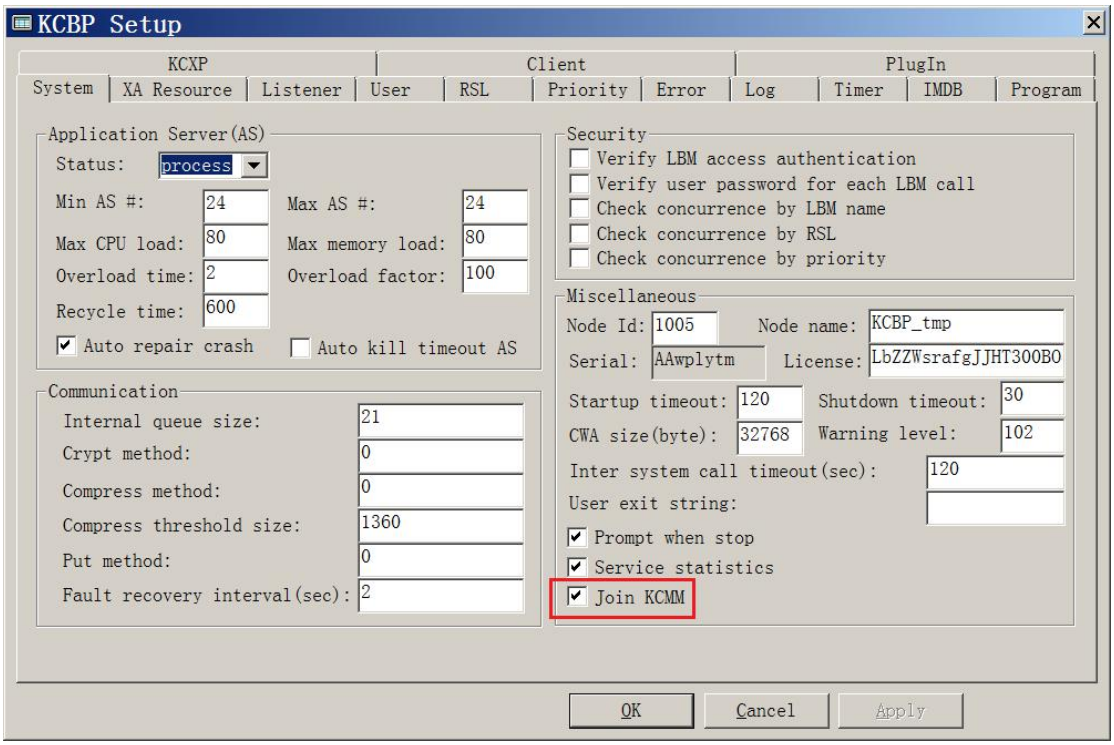
KCXP Listener

KCBP Log

KCBP Daemon

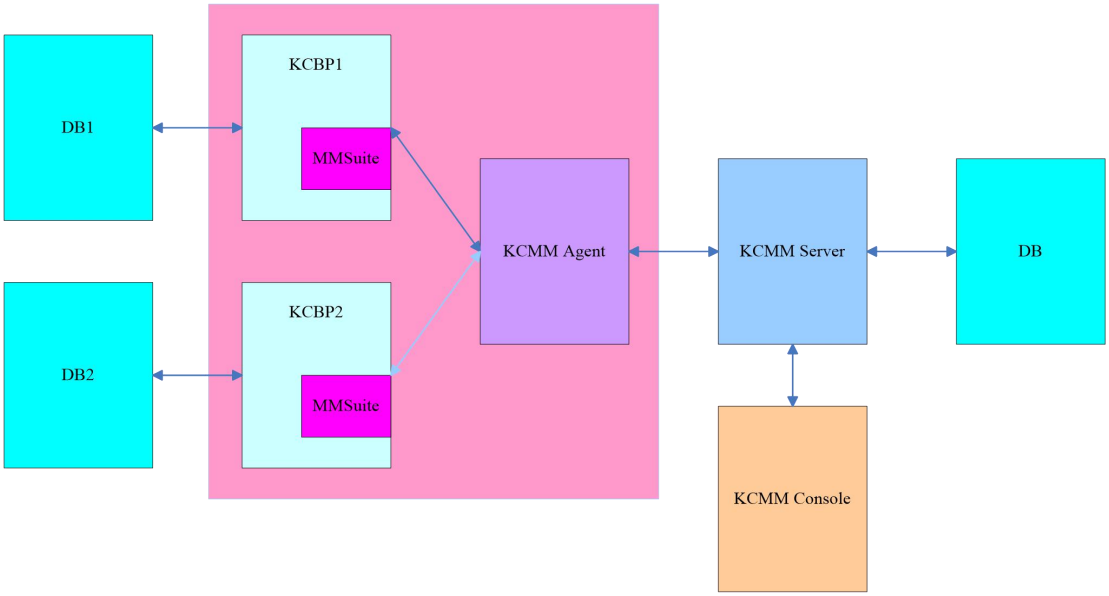
KCBP AS

8.7.2 KCBP 上 KCMM 参数配置



8.7.3 KCBP 主备切换

通过 KCMM 控制台可以启动、关闭 KCBP。
启动命令是：KCBP 安装目录\bin\kcbp start
停止命令是：KCBP 安装目录\bin\kcbp start

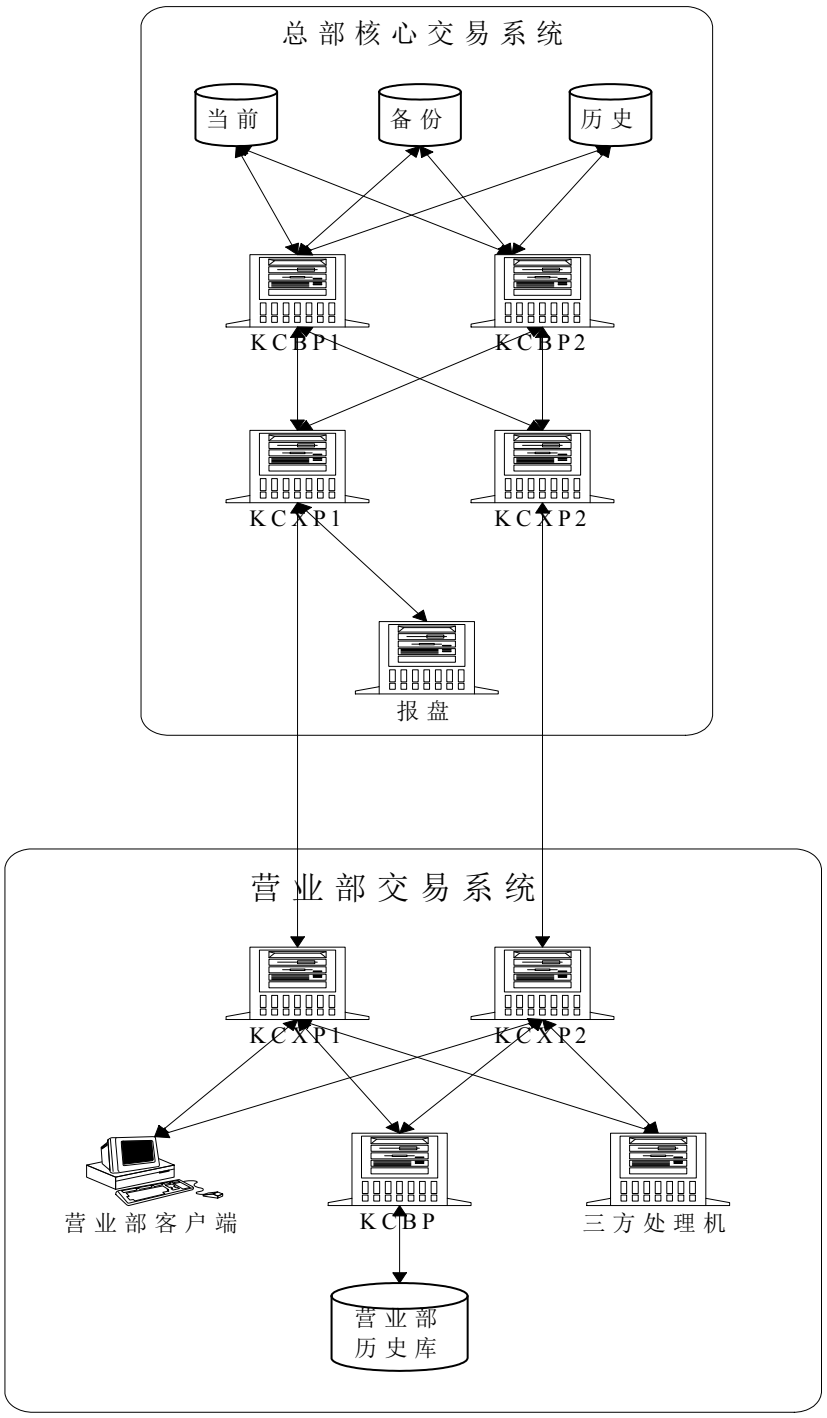


上图中 KCBP1 和 KCBP2 位于同一台机器，分别安装在不同的目录，一个连接主数据库 DB1，另一个连接备份数据库 DB2，当主数据库出现故障时，通过 KCMM Console 发布命令停止 KCBP1，然后再发布命令启动 KCBP2。

8.8 KCBP 在金证新一代集中交易系统中应用实例

8.8.1 KCBP 在集中交易系统中的部署图

KCBP 在集中交易系统中的典型部署图如下：



说明：

1. 在总部的核心交易系统中，部署两个 KCBP，作为负载均衡和冗余，每个 KCBP 的配置完全一样，其中一个 KCBP 出现问题，另一个 KCBP 可以继续工作，对客户端程序来说没有任何影响，是透明的。详细的配置说明见§8.7.3KCBP 的配置。
2. 营业部交易系统中，部署一个 KCBP，作为在营业部端做历史查询用，该 KCBP 只连接一个历书数据库。详细的配置说明见§8.7.3KCBP 的配置。

8.8.2KCXP 的配置

配置文件名	配置文件说明
Kcxp.ini	KCXP 的主配置文件，可以直接修改此文件，也可以通过图形界面 kcxpmanage 来修改 KCXP 的配置，最终修改的还是该文件
Kcxpcli.ini	KCXP 命令管理器 XPCC 的配置文件
Kcxpmanager.ini	KCXP 图形管理器 kcxpmanager 的配置文件，主要配置 KCXP 服务 kcxpsvc.exe 的路径
Exit.ini	KCXP 插件的配置文件，包括 transmit 和 hist 两个插件在业务应用上的配置

8.8.2.1KCXP 基本配置

在集中交易系统中，包括总部和营业部两部分 KCXP 的配置，它们的基本配置是一样的，只是在业务用途上有所差别。具体配置说明如下：

1. 总部核心交易系统的 KCXP 的基本配置：

```
[Node]
Descript=总部核心交易系统 1KCXP
NodeName=core1
NodeCode=10000001
QMgrName=core1
NetAddr1=TCP,10.101.8.56,20000,MCA
NetAddr2=TCP,10.101.8.56,21000,UIS
NetAddr3=TCP,10.103.3.42,20000,MCA
NetAddr4=TCP,10.103.3.42,21000,UIS
[CommServer]
CommServIP=10.103.3.42
CommServPort=5000
CommCliIP=NULL
[Monitor]
IPAddr=10.103.3.42
Port=7020
```

说明：

- 在[Node]项里配置 KCXP 的节点名称，节点代码，队列管理器名称，侦听的 IP 地址和端口；
- [CommServer]项是配置 KCXP 命令行管理器侦听的 IP 地址和端口；与 kcxpcli.ini 中的[CommServer]项配置相对应；
- [Monitor]项是配置 KCXP 监控器侦听的 IP 地址和端口；
- 总部 KCXP 为双网卡，其中 10.101.8.56 连接 KCBP（对内），10.103.3.42 连接营业部的 KCXP 或 KCBP Client（对外）

- UIS 端口 21000 是 kcxp api 程序用的；
- MCA 端口 20000 是两个 KCXP 节点间建路由时使用的；

2. 营业部的 KCXP 的基本配置为：

[Node]
Descript=KCXP01
NodeName=31150spb
NodeCode=10311501
QMgrName=31150spb
NetAddr1=TCP,10.100.33.218,20000,MCA
NetAddr2=TCP,10.100.33.218,21000,UIS
[CommServer]
CommServIP=10.100.33.218
CommServPort=5000
CommCliIP=NULL
[Monitor]
IPAddr=10.100.33.218
Port=7020

注：总部和营业部的 NodeCode 和 QmgrNam 需要统一命名。
总部和营业部的 KCXP 在基本配置上只是 NodeName，NodeCode，QmgrName 和侦听的 IP 地址不同，配置方法一样。

3. 总部和营业部之间使用远程队列来进行通讯，需要在双方都配置路由和远程队列，配置过程如下：

1) 在总部和营业部端的 KCXP 上分别建立路由，通过命令管理器 XPCC 来完成。
命令格式为：
总部端：
addroute –n 10311501 –m 31150spb –p 1 –a 10.100.33.218 –o 20000 –t 0 –u KCXP00 –w 888888 –d 0 –c 1 –r 1
此处，“–n 31150001”表示连接营业部的 KCXP 队列管理器的 NodeCode，
“-c 1”为两个节点间连接数，（可以根据应用的需要增加或减小该值）
“-r 1”表示是配置路由的服务端（总部都使用-r 1）
“-d 0”表示不是缺省路由，（都设置成非缺省路由）
营业部端：
addroute –n 10000001 –m *core1* –p 1 –a 10.103.3.42 –o 20000 –t 0 –u KCXP00 –w 888888 –d 0 –c 1 –r 0
此处，“–r 0”表示是配置路由的客户端（营业部都使用-r 0）
说明：先建服务端的路由，再建客户端的路由。不必重新启动 KCXP。

2) 建立好路由以后，用 checkroute 命令检测路由是否连通
命令格式为：
总部端：checkroute –n 10311501
此处“–n 10311501”为对方 KCXP 的 NodeCode
营业部端：checkroute –n 10000001
此处“–n 10000001”为对方 KCXP 的 NodeCode

3) 路由连通以后，建立远程队列
命令格式为：

总部端：addrdef –n *ans_3115shyp* –c 10311501 –d *ans1* –s *0*
其中 *ans_3115shy* 是总部 KCXP 上的队列，作为营业部端本地队列 *ans1* 的远程队列，10311501 是此营业部队列管理器的节点号 NodeCode。
营业部端：addrdef –n *req1* –c *10000001* –d *req_3115shyp* –s *0*
其中 *req1* 是营业部 KCXP 上的队列，作为总部端本地队列 *req_3115shyp* 的远程队列，*10000001* 是总部队列管理器的节点号 NodeCode。
注：用 sync –a 命令保存设置。

4. 营业部柜台使用的队列为：
RequestQName = req1 （远端队列）
AnswerQName = ans1 （本地队列）
说明：对于客户端来说，发送请求的是远端队列；接收请求的是本地队列。
5. 总部 KCBP 使用的队列为：
<externalqueue direction="receive" host="dispatch" id="311501" listener="1" name="req_3115shyp" node="2001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp" /> （本地队列）
<externalqueue direction="send" host="dispatch" id="311501" listener="1" name="ans_3115shyp" node="2001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp" /> （远端队列）
说明：
I. 对于 KCBP（服务端）来说，direction="receive"接收请求的是本地队列；direction="send"发送请求的是远端队列。
II. KCBP 的侦听队列是一对一对的配，但实际上应答队列不是按照 direction="send"指定的队列来用，而是通过路由直接将请求放到客户端接受应答的队列中。不过在配置方面还是按照这种方式配一对队列。

8.8.2.2KCXP 插件配置

1. KCXP 中现在有 3 种插件：
1) 行情插件：xphq.dll, 供外围系统取行情用。用在营业部的 KCXP 上。
2) 历史查询插件：hist.dll，在营业部做历史查询用，用在营业部的 KCXP 上。
3) 交易转发插件：transmit.dll，做交易转发用，用在总部的 KCXP 上。
2. 插件的配置方法：
1) 在 KCXP 的侦听地址上配置插件，修改 KCXP.ini 中的[Exit]
[Exit] #内挂插件
NetAddr2= xphq.dll ,hist.dll, transmit.dll #该地址为 KCXP 侦听的 UIS 端口的地址。
说明，如果有两块网卡的话，则第二块网卡也配上插件，如下：
NetAddr4= xphq.dll ,hist.dll, transmit.dll
2) 用 XPCC 命令来增加插件的信息
命令格式：addexit –f xphq.dll –c hangqing
addexit –f hist.dll –c histquery
此处“-f xphq.dll”为插件的文件名（dll 文件）。
3) 插件 dll 全部都放在 KCXP 的 exit 目录下。
3. 插件的用法：

1) 行情插件 xphq.dll：

- 将 KCBPPacketOpApi.dll 到和 kcxpsvc.exe 同一个目录下。
- 将内存行情服务器程序 memhq.exe 和其配置文件单独放在同一个目录中。
- 运行 memhq.exe

2) 历史查询插件 hist.dll

修改 exit.ini，如下所示：

```
;报文类型, 请求号, 机构代码, 目标节点编号, 目标队列名(本地队列)
[hist]
TypeOffset = 2
TypeLen = 1
ServiceOffset = 94
ServiceLen = 8
InstOffset = 107
InstLen = 4
bus1 = 2, 82?, NULL, 10311501, req_history
bus2 = 2, 84?, NULL, 10311501, req_history
bus3 = 2, 86?, NULL, 10311501, req_history
bus4 = 2, 88?, NULL, 10311501, req_history
```

只需要修改请求号（需要做历史查询的请求），目标节点编号（营业部 KCBP 连接的 KCXP 的节点编号），目标队列名（是本地队列，在营业部的 KCBP 上配置侦听该队列）请求号的？是通配符。

3) 交易转发插件 transmit.dll

修改 exit.ini，如下所示：

```
[transmit]
;报文类型位置
TypeOffset = 2
;报文类型长度
TypeLen = 1
;功能号位置
ServiceOffset = 94
;功能号长度
ServiceLen = 8
;机构代码位置
InstOffset = 113
;机构代码长度
InstLen = 4
;报文类型, 请求号, 机构代码, 目标节点编号, 目标队列名(本地队列)
bus1 = 2, 998001, 6666, 10311501, req_kcbp2
bus2 = 2, 998002, 3115, 10311501, req_ kcbp1
```

只需要修改请求号，机构代码，目标节点编号，目标队列名。

8.8.2.3KCXP 集群配置

在总部和营业部的两个 KCXP 之间都可以配置集群，配置方法相同，如下所示：

1. KCXP 集群配置对应于 KCXP.ini 的[Cluster]

[Cluster]

ClusterID=00001 说明：前 4 为该节点所在集群的 clusterid,第五位为该节点所在集群的顺序，00000 表明该节点不使用集群

CPU=100 说明：cpu 所占的权重

MEM=0 说明：内存所占的权重

SWAP=0 说明：虚拟内存所占的权重

CONN=0 说明：socket 连接所占的权重

LBValue=8 说明：该节点与集群内其它节点的差值

LocalLoadAddr=10.100.33.218,6006,TCP 说明：该节点集群通道的侦听地址，端口和协议

Cluster1=10.100.33.219,TCP,10311502 说明：集群内其他节点的侦听地址，端口和协议，节点编号

Cluster2=

Cluster3=

... ...

Cluster15= 说明：一个集群内最多有 16 个节点

2. 对于同一集群的节点，在地址项的配置和顺序，除了地址不同外必须完全相同。
3. 对于同一集群的节点，在 KCXP.ini 中的[exit]和[SSL]配置顺序必须完全相同。
4. 对于同一集群的节点，kcxpexit.dat 必须相同。
5. 在 exit.ini 中必须指定目标节点编号。
6. 只有使用新的 KCXP 客户端(KCXPAPI)和新的 KCXP 才能使系统做到冗余。
7. 营业部的 KCXP 集群和总部 KCXP 集群应使用交叉连接，才能充分发挥系统的冗余功能。
8. 应用程序（报盘机，comtrans, sfjy, tlink）即可以更新 kcxpapi.dll 来实现 KCXP 冗余，也可以保持现状。

8.8.3 KCBP 的配置

8.8.3.1KCBP 的基本配置

KCBP Config 文件 KCBPConf.xml 的使用说明

KCBP 的主要配置文件有 kcxpapi.ini，KCBPConf.xml 和 KCBPSPD.xml

1. kcxpapi.ini 文件是配置 KCBP 与 KCXP 的连接，文件内容如下：

[QMgr]

QMgrName=core1

QMgrProtocol=TCP

QMgrIp=10.101.8.56

QMgrPort=21000

其中的 IP 地址和端口是 KCXP 里指定的。

注意：在 KCBPConf.xml 中配置侦听队列时，要修改 queuemanager 为你所要使用的 KCXP 的队列管理器名。

2. 在 kcxpapi.ini 中，配置 KCXP 主动发连接探测包，检测网络是否断线，如下所示：

```
[Option]
Nagle=NO
TcpMss=DEFAULT
TcpRcvBuf=DEFAULT
TcpSndBuf=DEFAULT
#TCP 保活设置：应用程序在 iTCP_KeepLive + 5 * iTCP_KeepLiveAfter 时间内会返回 2054 错误
TcpKeepLive=5       #保活检测时间间隔, 单位是秒, 不设置为 DEFALUT
TcpKeepLiveInv=2 #在 TcpKeepLiveInv 时间段内没有应答时重发保活探测分节间隔, 单位是秒
```

缺省配置为上述配置：在最短时间 5 * iTCP_KeepLiveAfter=5*2=10 秒内,KCBP 检测到 KCXP 网络断线；在最长时间 iTCP_KeepLive + 5 * iTCP_KeepLiveAfter=5+5*2=15 秒内,KCBP 检测到 KCXP 断线。

3. 在 KCBPConf.xml 文件主要是配置 KCBP 的系统参数，侦听的队列和数据库连接，以下只介绍侦听队列，数据库连接的配置，其他参数的详细说明请参考§6 中的说明。

1) 配置侦听队列，如下所示：

- 请求队列和应答队列都是本地队列，直接连接总部 KCXP 的，按这种配置；

```
<!--offer queue of local-->
<externalqueue direction="receive" host="dispatch" id="102" lifetime="300" listener="1" name="req_offer" node="1001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>
<externalqueue direction="send" host="dispatch" id="102" lifetime="300" listener="1" name="ans_offer" node="1001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>
```

- 请求队列是本地队列，应答队列是远端队列。从营业部连接到总部，按这种配置；

```
<!--shanghai ypl queue of remote-->
<externalqueue direction="receive" host="dispatch" id="311501" lifetime="300" listener="1" name="req_3115shyp" node="1001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>
<externalqueue direction="send" host="dispatch" id="311501" lifetime="300" listener="1" name="ans_3115shyp" node="1001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>
```

说明：请求队列 req_3115shyp 是本地队列；
应答队列 ans_3115shyp 是远端队列；

2) 侦听队列的命名规范

- 请求队列按 req_xxxx 这种格式，xxxx 表示该队列的用途，用有意义的名字，可以用英文或字母、数字等的组合；
- 应答队列按 ans_xxxx 这种格式，xxxx 表示该队列的用途，用有意义的名字，可以用英文或字母、数字等的组合；
- 对于营业部连接到总部使用的队列，队列名字 req_xxxxxxxx，ans_xxxxxxxx，其中

xxxx 的前四位用该营业部的营业部代码，如 3115,3111 等，xxxx 的后几位为该营业部的拼音缩写，如 A 营业部表示为 shyp，将营业部代码和营业部缩写组合起来就是该营业部使用的队列名，如 req_3115shyp，ans_3115shyp；

3) 配置侦听队列的注意事项：

- 对于 queuemanager="core1"项，它的值必须和 kcxpapi.ini 中 QmgrName 项的值相同
- 对于 id="102"项，每一对请求队列和应答队列有唯一的 id 值，不能重复。

  说明：

  直接连总部使用的队列的 id 项的值用 3 位数字表示，从 101 开始，依次加一；

  供营业部使用的队列的 id 项的值用 6 位数字表示，前 4 位是该营业部的营业部代码，后两位为 01,02 这样的序数来表示是第几对队列。
- 其余项按缺省配置；

4) 配置与数据库的连接：

通过 XA 项配置与数据库的连接，如下所示：

```
<!--core1-run-->
<xa entry="KCBP_ODBC1PC" name="tradedb" switchloadfile="odbc1pc.dll" xaclose="" xaopen="spb-run,master,gjman,Kya7sBvwynyuTfGDeQ1hBg==" xaoption="" xaserial="all_operation"/>
```

说明如下：

- name="tradedb"，该项用来表示是那种类型的数据库，名称是由业务来规定的，本系统用到的 4 种数据库分别是：
  I. name="tradedb"，表示是当前库；
  II. name="backupdb"，表示是备份库；
  III. name="querydb"，表示是历史库；
- xaopen="spb-run,master,gjman,Kya7sBvwynyuTfGDeQ1hBg==",数据库的连接配置。其中：
  I. spb-run 是 ODBC 数据源的别名，表示连接哪种数据库；
  II. master 是数据库名；
  III. gjman 是登陆数据库的用户名；
  IV. Kya7sBvwynyuTfGDeQ1hBg== 是 gjman 的密码，为密文格式（明文是 1234567890），此处不能用明文，加密方法见“KCBP 命令行管理器使用说明”。
- 其余项按缺省配置；

4. 在 KCBPConf.xml 文件中其他 XA 项的配置

1) 内存行情服务器的配置，（在 KCBP 上使用内存行情数据库）如下：

```
<!--memdb hq-->
<xa entry="KCBP_MDBXA" name="memdbxa" switchloadfile="memdbxa.dll" xaclose="" xaopen=",,,," xaoption="" xaserial="all_operation"/>
```

其中 entry="KCBP_MDBXA"，name="memdbxa"，switchloadfile="memdbxa.dll"为固定格式，不必修改。

2) LBM 中直接使用 KCXP 队列的配置，以报盘队列的配置为例，如下：

```
<xa entry="KCBP_KCXPXA" name="offer" switchloadfile="kcxpxa.dll" xaclose="" xaopen="10.101.8.56:21000,shagoffer,KCXP00,GV2gODkBbGg=" xaoption="" xaserial="all_operation"/>
```

说明：

- entry="KCBP_KCXPXA"，为使用 KCXP 时的固定格式；

- name="offer"，该名字与业务有关，是和 LBM 中约定好的名字，不同的业务用不同的名字。
- switchloadfile="kcxpxa.dll"，装载的 DLL 名字，为固定名字；
- xaopen="10.101.8.56:21000,shagoffer,KCXP00,GV2gODkBbGg="，使用 KCXP 队列的配置。

  其中：

  I. 10.101.8.56:21000 是连接的 KCXP 的 IP 地址和端口；
  II. shagoffer 是使用的 KCXP 队列名称；
  III. KCXP00 是 KCXP 的用户名；
  IV. GV2gODkBbGg=是 KCXP00 的密码，密文格式，（明文是 888888），此处必须用密文。
- 其余项按缺省配置；

5. 在 KCBPConf.xml 文件中日志项的配置：

<!--log-->

<log displaylevel="1000" filename="runlog" filepath="./log/run" flushfile="no" flushterminal="no" key="11" maxfilenum="5" maxsize="1024" recordtype="cycle" type="run" writelevel="1000"/>

说明：

- 只有 type="run"的日志项配置有用，type="biz"的日志项配置暂时没用；
- displaylevel="0"，表示所有内容都显示，”1000”只显示错误信息，“10000”是什么都不显示。
- writelevel="0"表示所有内容都写文件，”1000”只写错误信息，“10000”是什么都不写。

8.8.3.2KCBP 连接多个 KCXP 的配置

1. 在 kcxpapi.ini 中配置多个 KCXP 的地址和端口，如下：

```
[QMgr]
QMgrName=core1
QMgrProtocol=TCP
QMgrIp=10.101.8.56
QMgrPort=21000
[QMgr1]
QMgrName=core1_bak
QMgrProtocol=TCP
QMgrIp=10.101.8.57
QMgrPort=21000
```

如果配 2 个以上的 KCXP，则继续配[QMgr2]、[QMgr3]…等

2. 在 KCBPConf 的侦听队列中用不同的 queuemanager，如下所示：

<!--第一个 KCXP-->

<externalqueue direction="receive" host="dispatch" id="103" lifetime="300" listener="1" name="req_tctd" node="2001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>

<externalqueue direction="send" host="dispatch" id="103" lifetime="300" listener="1" name="ans_tctd" node="2001" poolsize="1" priority="normal" queuemanager="core1" timeout="120" type="kcxp"/>
<!--第二个 KCXP-->
<externalqueue direction="receive" host="dispatch" id="104" lifetime="300" listener="1" name="req_tctd" node="2001" poolsize="1" priority="normal" queuemanager=" core1_bak " timeout="120" type="kcxp"/>
<externalqueue direction="send" host="dispatch" id="104" lifetime="300" listener="1" name="ans_tctd" node="2001" poolsize="1" priority="normal" queuemanager=" core1_bak " timeout="120" type="kcxp"/>

8.8.3.3营业部做历史查询的 KCBP 的配置

1. 基本配置方法与总部的 KCBP 相同，主要配置如下所示：

配 ODBC 连接和 XA（tradedb 和 querydb),

<xa entry="KCBP_ODBC1PC" name="tradedb" switchloadfile="odbc1pc.dll" xaclose="" xaopen="spb-run,master,sa,," xaoption="" xaserial="all_operation"/>
<xa entry="KCBP_ODBC1PC" name="querydb" switchloadfile="odbc1pc.dll" xaclose="" xaopen="spb-his,master,sa,," xaoption="" xaserial="all_operation"/>

2. 只需要配置一对队列就可以，请求队列是在 exit.ini 中指定的队列 req_history,应答队列可以随便配，KCBP 不根据配的应答队列放消息，而是根据实际请求从哪里来就会放回哪里去，一般配成和柜台的应答队列一样就行）

<externalqueue direction="receive" host="dispatch" id="104" lifetime="300" listener="1" name="req_history" node="2001" poolsize="1" priority="normal" queuemanager="core2" timeout="120" type="kcxp"/>
<externalqueue direction="send" host="dispatch" id="104" lifetime="300" listener="1" name="ans1" node="2001" poolsize="1" priority="normal" queuemanager="core2" timeout="120" type="kcxp"/>

3. kcbpspd.xml 和总部的一样就行，LBM 放置的路径也一样，这样就什么都不用改

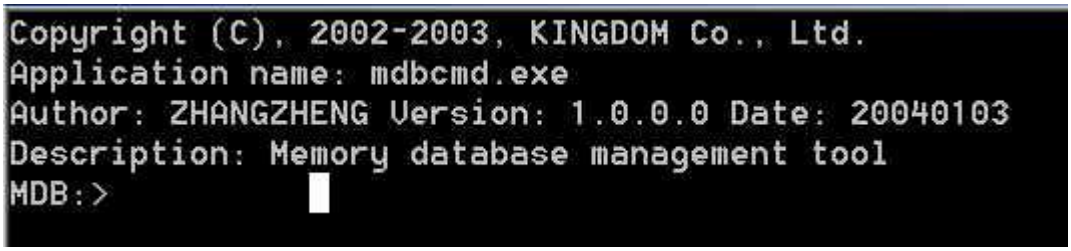
8.8.3.4内存行情服务器的使用

1. 启动内存行情服务器
   - 设置行情路径，如下图所示：

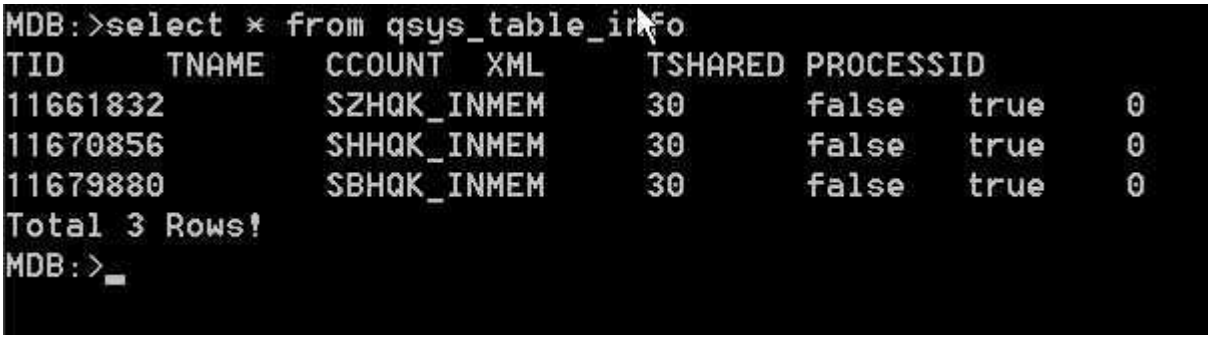
- 点击运行按钮，启动内存行情服务器，启动成功以后，会显示行情库中的记录数，如下图所示

- 如果进度指示标志在移动，表示内存行情服务器在正常运行。

2. 停止内存行情服务器
   - 点击停止按钮，停止内存行情服务器。
   - 点击退出按钮，退出内存行情服务器。
3. 检测内存行情服务器（用 mdbcmd 命令）
   - 运行 mdbcmd 命令，如果能正常启动，则表示内存行情服务器正常运行，如下图所示：



   - 使用 mdbcmd 命令：用 select * from qsys_table_info 查看行情库的表名



- 深圳行情库在 SZHQK_INMEM 表里，上海行情库在 SHHQK_INMEM，三板行情库在 SBHQK_INMEM 里
- 如果查看行情库里的股票行情，使用 select 命令，用法与标准的 SQL 中用法相同。

8.9 KCBP 和 KDMID 集成

8.9.1 KDMID1PC

KDMID1PC 是 KCBP 与 KDMID 集成的适配器，它遵循 KCBPXA 规范。KDMID1PC 使用 Refer.XML 文件来描述 KCBP 和 KDMID 报文转换规则。

8.9.1.1 KDMID1PC 的配置

KDMID1PC 在 KCBPConf.xml 中配置，举例如下：
<xa entry="KCBP_KDMID1PC" name="kdmid30" switchloadfile="kdmid1pc.dll" xaclose="" xaopen="127.0.0.1:28946"
xaoption="userid=9999,password=3Mc+w9uU5lM=,srcnodeid=0000,destnodeid=0000,opcode=3,timeout=30" xaserial="all_operation"/>

其中 xopen 含义如下：
127.0.0.1:28946 是 KDMID 中间件的 ip 地址和端口号
xaoption 各项含义如下：
柜员代码 userid=9999
柜员密码(密文)password=3Mc+w9uU5lM=
源节点编号 srcnodeid=0000
目的节点编号 destnodeid=0000
操作方式 opcode=3
超时(秒)timeout=30

在 kcbp 的图形界面中加密柜员密码方法是输入：
userid=9999,password=encrypt(9999,888888),srcnodeid=0000,destnodeid=0000,opcode=3

使用 kcbpcp 生成密文的柜员密码方法是：
kcbp => encrypt
input key:9999
input plaintext:888888
cipher text is: 3Mc+w9uU5lM=
其中 9999 是用户名,888888 是柜员密码明文，3Mc+w9uU5lM=是柜员密码密文。

8.9.1.2报文转换规则举例

8.9.1.2.1 MID 多结果集形式的返回

以查询市场设置为例。
LBM 名称是 L9000001，对应 MID 的请求编号 20102900。

8.9.1.2.1.1 MID 入参

20102900.bex 中输入参数为：

```
GETREQ
   1 @jysdm
   2 @hbdm
ENDGETREQ
```

8.9.1.2.1.2 MID 出参

在 20102900 中返回结果为多结果集，格式为

```
DBF
   jysdm C 1 0
   jysjc C 6 0
   hbdm    C 1 0
   gddmcd N 2 0
   nbgddmcd N 2 0
   zqdmcd N 2 0
   nbzqdmcd N 2 0
   jysbs    C 4 0
   jymmcd N 2 0
ENDDBF
```

8.9.1.2.1.3 Refer 描述

```
<!-- query market information /-->
<message src="L9000001" dst="20102900">
      <input format="!table">
            <field src="jysdm" dst="1"/>
            <field src="hbdm" dst="2"/>
      </input>
      <output format="table">
            <field src="jysdm" dst="jysdm"/>
            <field src="jysjc" dst="jysjc"/>
            <field src="hbdm" dst="hbdm"/>
            <field src="gddmcd" dst="gddmcd"/>
            <field src="nbgddmcd" dst="nbgddmcd"/>
```

```
            <field src="zqdmcd" dst="zqdmcd"/>
            <field src="nbzqdmcd" dst="nbzqdmcd"/>
            <field src="jysbs" dst="jysbs"/>
            <field src="jymmcd" dst="jymmcd"/>
        </output>
</message>
```

其中，src 定义前端请求的服务名称，dst 定义 KDMID 的请求编号，input 定义入参转换规则，output 定义出参转换规则。input 格式!table 说明前端请求不是二维表。入参的 Filed 中定义如参的具体内容，其中 src 表示前端输入的入参名称，dst 表示转换到 KDMID 的请求顺序，output 格式 table 表示 KDMID 返回的数据格式是二维表，返回给前端的列在 field 中定义，每个 Field 定义一列，src 表示 KDMID 返回的列名称，dst 表示回送给前端的列名称。

8.9.1.2.2 KDMID 非结果集形式的返回

以委托为例。
委托的 LBM 名称是 L9000002，对应 MID 的请求编号 20103900。

8.9.1.2.2.1 MID 入参

```
GETREQ
  1 @jysdm
  2 @gddm
  3 @mmlb
  4 @zqdm
  5 @wtgs
  6 @wtjg
  7 @bzxx
  8 @wldz
ENDGETREQ
```

8.9.1.2.2.2 MID 出参

```
RET $sql
  &htxh
  &zjzh
  &hbdm
  &wtje
  &wtdjje
  &zjbckys
  &gffss
  &gfbckys
ENDRET
```

8.9.1.2.2.3 Refer 描述

```
<!-- order /-->
<message src="L9000002" dst="20103900">
    <input format="!table">
        <field src="jysdm" dst="1"/>
        <field src="gddm" dst="2"/>
        <field src="mmlb" dst="3"/>
        <field src="zqdm" dst="4"/>
        <field src="wtsl" dst="5"/>
        <field src="wtjg" dst="6"/>
        <field src="wkdz" dst="7"/>
    </input>
    <output format="!table">
        <field src="1" dst="htxh"/>
        <field src="2" dst="zjzh"/>
        <field src="3" dst="hbdm"/>
        <field src="4" dst="wtje"/>
        <field src="5" dst="wtdjje"/>
        <field src="6" dst="zjbckys"/>
        <field src="7" dst="gffss"/>
        <field src="8" dst="gfbckys"/>
    </output>
</message>
```

注意，output 的格式!table 表示输出 KDMID 返回参数不是二维表，output 的 field 定义输出列转换规则，src 是 KDMID 返回的参数编号，dst 是回送给前端的列名称。

8.9.1.2.3 缺省值及字典的使用

```
<dict name="market_out">
    <entry name="0" value="2"/>
    <entry name="1" value="1"/>
    <entry name="2" value="H"/>
    <entry name="3" value="D"/>
</dict>

<dict name="inputtype_in">
    <entry name="1" value="1"/>
    <entry name="2" value="0"/>
    <entry name="D" value="3"/>
    <entry name="H" value="2"/>
    <entry name="Z" value="Z"/>
    <entry name="K" value="C"/>
```

```
</dict>

<message src="P410301" dst="20102905">
  <input format="!table">
    <field src="inputtype" dst="1" dict="inputtype_in"/>
    <field src="inputid" dst="2"/>
    <field src="trdpwd" dst="3"/>
    <field src="mode" dst="4" default="2"/>
    <field src="netaddr" dst="5"/>
  </input>
  <output format="table">
    <field src="jysdm" dst="market" dict="market_out"/>
    <field src="gddm" dst="secuid"/>
    <field src="gdxm" dst="name"/>
    <field src="zjzh" dst="fundid"/>
    <field src="yybdm" dst="orgid"/>
  </output>
</message>
```

其中 market_out 和 inputtype_in 是字典，entry 定义字典条目，name 表示字典条目名称，value 表示字典条目含义。
<field src="inputtype" dst="1" dict="inputtype_in"/>表示将 inputtype 的值依照 inputtype_in 字典进行转换，比如，当 inputtype 是”D”时，转换成”3”放到第一个请求字段。
<field src="mode" dst="4" default="2"/> 中使用了缺省值，当前端未输入 mode 时，KDMID1PC 将缺省值”2”放到第 4 个请求字段。

8.9.1.3KDMID1PC 的错误处理方法

8.9.1.3.1 初始化连接错误

置连接失败标志。

8.9.1.3.2 失去与 MID 的连接

断线后，置连接失败标志，但不尝试重连，当前交易作失败处理。

8.9.1.3.3 与 MID 连接错误

连接成功后，置成功标志，如果错误，置失败标志。
每次向 MID 发送请求时，检查连接标志，如果已建立，则发送，如果未建立，重新建立连接，如连接成功再发送，如不成功则当前交易作失败处理。

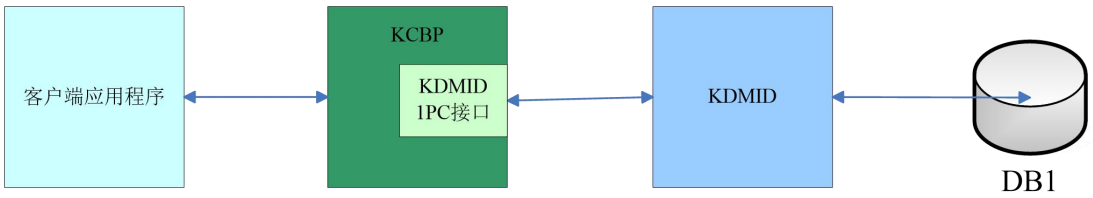
8.9.1.3.4 MID 返回错误

MID 报文头中，fhm!=0，表示遇到错误，MID 返回数据区包含错误代码和错误信息两项内容。

MID 调用失败时，不按 Refer 中描述的应答格式进行转换，直接返回一个错误结果集，结果集有 3 列，名称分别是 LEVEL,CODE,MSG，CODE 存放错误码，MSG 存放错误信息。

8.9.2 应用模式举例

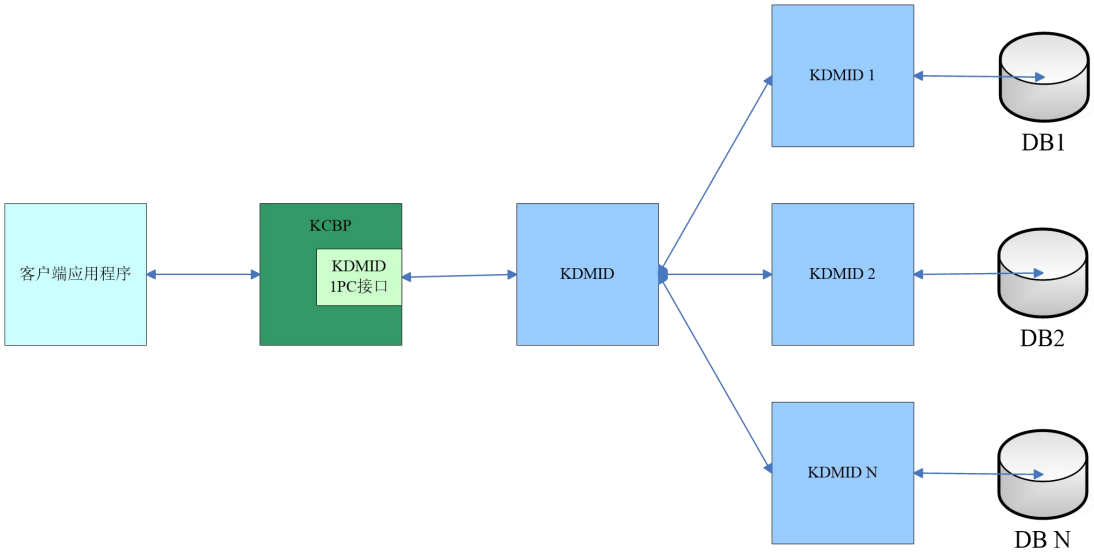
8.9.2.1 单 KCBP-单 KDMID



KCBP和KDMID系统关系图

这是最简单的应用模式。

8.9.2.2 单 KCBP-单 KDMID-多 KDMID

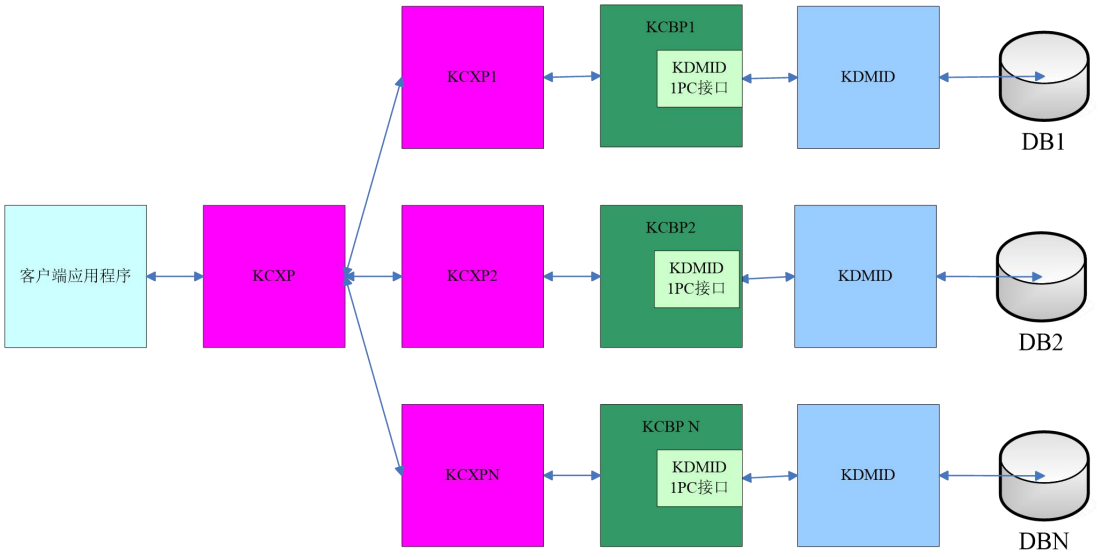


KCBP和KDMID系统关系图

这种模式，客户端通过编程提供目的节点编号，KCBP 将目的节点编号发送给 KDMID，

KDMID 根据目的节点编号进行转发。

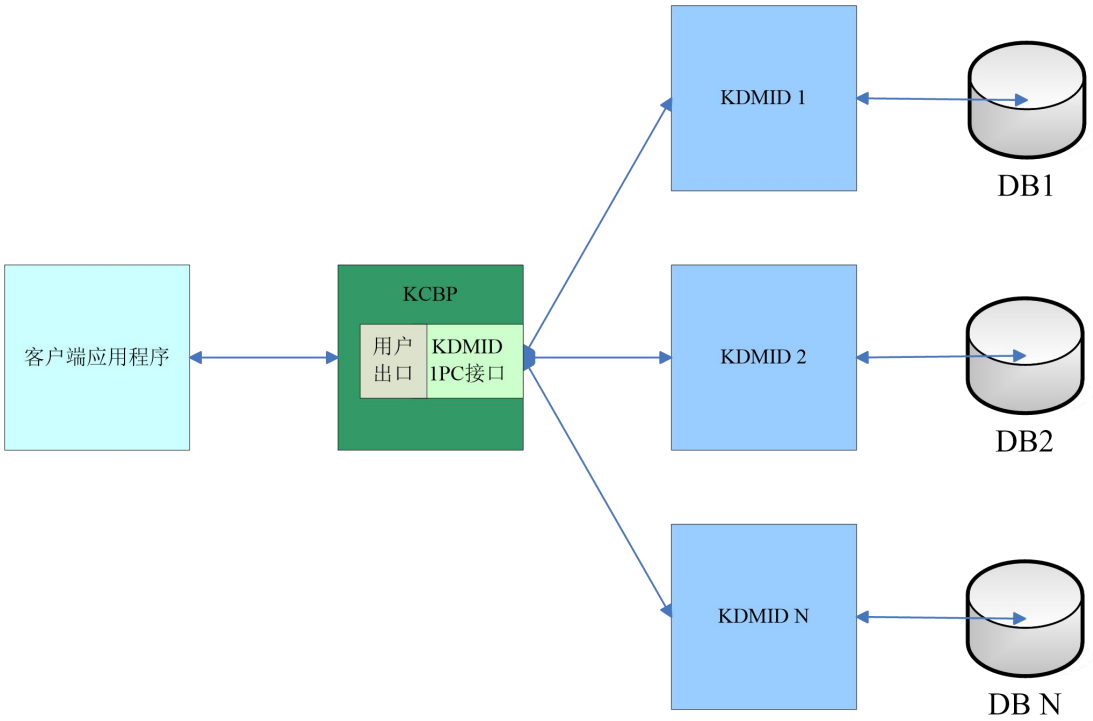
8.9.2.3KCXP 按目的节点编号转发



KCBP和KDMID系统关系图

这种模式中，客户端通过编程提供目的节点编号，KCXP 根据目的节点编号进行转发。转发规则定义在 KCXP 的 Exit.ini 中。

8.9.2.4单 KCBP-多 KDMID



KCBP和KDMID系统关系图

这种模式中，客户端通过编程提供目的节点编号，KCBP 使用用户出口识别转发节点。

上图是 KCBP 和 KDMID 进行集成的系统关系图，KCBP 是接入网关；KCBP 和 KDMID 是一对多的关系，在 KCBP 的 XA Resource 中，可配置多个 KDMID1PC 资源，每个 KDMID1PC 管理一个 KDMID；所有 KDMID 都提供相同的服务，但连接的数据库不同。
KCBP 如何确定前端送来的请求需要交给那个 KDMID 处理呢呢？解决问题的关键点有两个：一是请求报文中要通过保留字段传递目的节点名称；二是 KCBP 在进行转发处理时调用用户出口 22 来获得目的节点名称，并根据目的节点名称确定目的 XA 的名称，当前端传递的目的节点名称与 XA 名称一致时（比如都是节点编号），用户出口就有最简单实现：写一行代码，直接返回目的节点名称即可。

这种模式的处理流程如下：

1. 客户端发起请求
2. KCBP 接收到请求，检查 KCBPSPD.xml 中的 LBM 定义，当 type=”deputy”时，如果用户出口中定义 22，则调用用户出口取得 XA 的名称，如果没有定义 22，则使用 xa=”xaname”中定义的 XA 名称，并根据该 XA 名称取得 KDMID1PC 实例。
3. KCBP 调用 KDMID1PC 中的 write 方法转换数据格式并转发到 KDMID 上。数据格式转换时，注意 KCBP 的入参是按名称存取，MID 的入参按顺序存取，每个位置的含义由前后端程序约定，对应 Refer.xml 中参数转换规则的 Input 部分。对于前端未输入的参数，转换规则规定使用缺省值，如果没有定义缺省值，使用空字符串。
4. KDMID1PC 调用 KDMIDAPI 的 SendRequest 方法发送给 MID。

5. KDMID1PC read 函数模块调用 KDMIDAPI 的 GetReply 方法接收 MID 的返回结果。
6. KDMID1PC read 函数模块转换结果数据格式，KDMID 返回的数据格式包括单结果集和多结果集格式，根据《消息转换协议》统一转换成 KCBP 的多结果集应答格式。
   a) 单结果集：转换成多结果集格式，第一个结果集有 3 列，名称分别是 LEVEL,CODE,MSG，当 CODE 是"0"时，有后续结果集，后续列名在 Output 中定义，第二个结果集只有一行；当 CODE 不是"0"时，没有后续结果集，MSG 存放错误信息。
   b) 多结果集：KCBP 返回给前端的数据格式是多结果集格式，第一个结果集有 3 列，名称分别是 LEVEL,CODE,MSG，当 CODE 是"0"时，有后续结果集，后续结果集的行数等于 KDMID 返回的行数；当 CODE 不是"0"时，没有后续结果集，MSG 存放错误信息。

   后续结果集的列名由 Refer.xml 中的 Output 定义，可不局限于 KDMID 返回的内容，用户可以定义自己的返回列，并为其赋缺省值。KDMID 结果集的行转换成 KCBP 结果集的行。
7. KCBP 收到结果 KDMID1PC 返回的应答
8. KCBP 回送结果到前端。

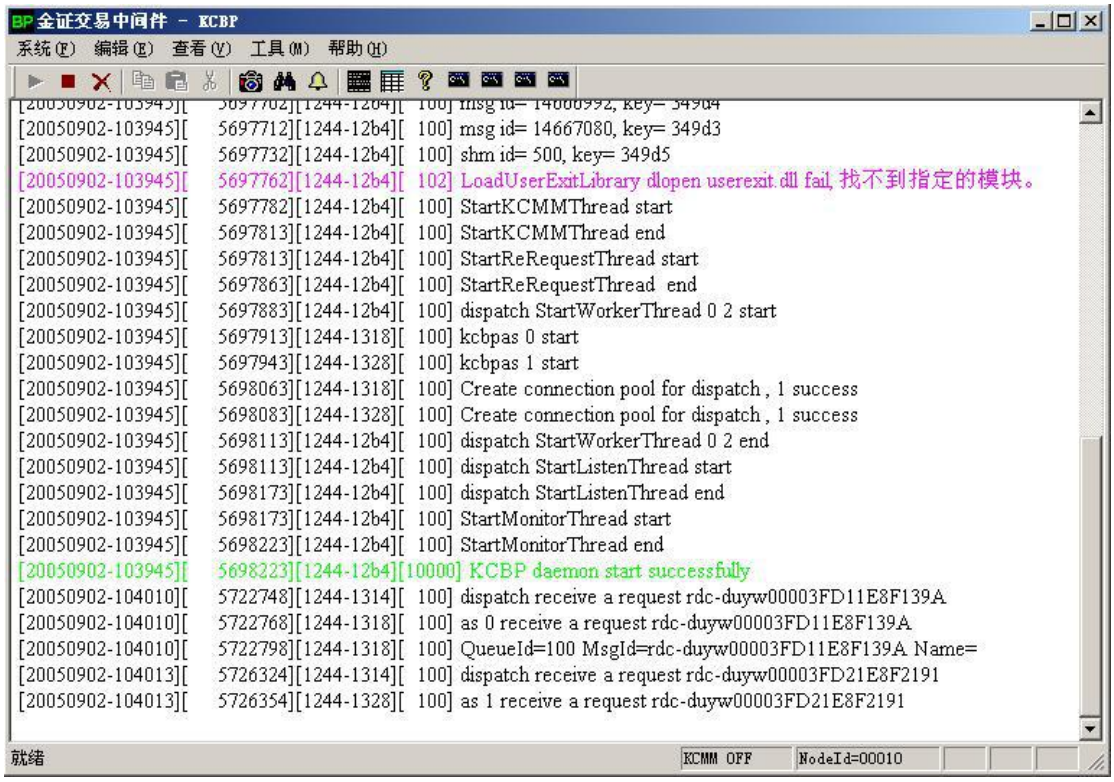
注：前端与 KDMID1PC 无关，它直接与 KCBP 交互。

用户出口的详细内容参见《KCBP 程序员手册》。

8.10错误定位和排除

8.10.1 日志

KCBP 通过日志记录系统的运行信息，发生错误时可以从日志中查找错误。日志分为显示日志和文件日志两种，显示日志只提供 200 行的显示内容，文件日志大小用户可定制。日志级别可以定制。日志级别越低，日志越详细，级别 100 以下是普通运行日志（98 记录输入参数，99 记录输出参数），100-999 是警告，1000-9999 是错误日志，10000 是 LBM 调试日志。



上图是 KCBP 的运行截面，其中黑色文字表示普通运行信息，红紫色文字表示警告信息，绿色文字表示调试信息和系统启动、停止信息，红色文字表示错误。

下面内容是日志的 1 行：

[20050808-155143][-1444662656][408- 924][100] as 0 receive a request CCXP_AS0000FFBB9EED5631

其中：

[20050808-155143]记录的是日期和时间

[-1444662656]表示的是以毫秒为单位的系统时间

[408- 924] 前面的 408 表示输入日志的进程号，924 表示线程号

[100]表示日志级别

as 0 receive a request CCXP_AS0000FFBB9EED5631 是日志的具体内容，as 0 表示第一个 AS 进程。

8.10.2 LBM 崩溃重现方法

当 LBM 程序编写得不规范时，可能导致系统抛出异常或崩溃，KCBP 在日志中记录导致异常或崩溃的输入参数，如下面的日志片段(特征是包含 LUW dump)：

[20050809-141149][-1364256656][b98- ad0][10000] LUW dump=0120000040351200000010000000000000000AEB9C331BKCXP00GV2gODkBbGg=

[20050809-141149][-1364256656][b98- ad0][10000]

1332000100146000000 170_CA=2.2&_ENDIAN=0&PageOff
[20050809-141149][-1364256656][b98- ad0][10000]
set=0&p_gybh=1&p_gnbh=13320001&zllxbh=&p_czzd=10.18.10.48&p_fzdm=&PageCount=21
47
[20050809-141149][-1364256656][b98- ad0][10000]
483647&p_gymm=cti110&khbh=2126846&p_kzcs=

对日志进行分析，首先去掉每行的固定部分，得到导致崩溃的入参如下：
012000004035120000001000000000000000000AEB9C331BKCXP00 GV2gODkBbGg=
1332000100146000000
170_CA=2.2&_ENDIAN=0&PageOffset=0&p_gybh=1&p_gnbh=13320001&zllxbh=&p_czzd=1
0.18.10.48&p_fzdm=&PageCount=2147483647&p_gymm=cti110&khbh=2126846&p_kzcs=

将入参交给 kcbpcp 进行格式化分析：
kcbp => format
input raw packet:012000004035120000001000000000000000000AEB9C331BKCXP00
GV2gODk
BbGg= 1332000100146000000
170_CA=2.2&_ENDIAN=0&PageOffset=0&p_gybh=1&p_gnbh=13320001&zllxbh=&p_czzd=1
0.18.10.48&p_fzdm=&PageCount=2147483647&p_gymm=cti110&khbh=2126846&p_kzcs=
0-Version:[01]
2-PackageType:[2]
3-PackageTypeExtend:[0]
4-ConnectionID:[0000403512]
14-PacketIndex:[000000]
20-EOP:[1]
21-OriginalNode:[00000]
26-SendNode:[00000]
31-DestNode:[00000]
36-AdapterAddress:[000AEB9C331B]
48-Username:[KCXP00]
56-Password:[GV2gODkBbGg=]
68-KCBPSerial:[ ]
94-SerivceName:[13320001]
102-PacketLength:[00146]
107-ReturnCode:[000000]
113-Reserved:[ ]
123-Parity:[170]

可以看到请求号是 13320001

输入参数是(去掉两个 kcbpcp 的系统参数_CA=2.2&_ENDIAN=0)：
&PageOffset=0&p_gybh=1&p_gnbh=13320001&zllxbh=&p_czzd=10.18.10.48&p_fzdm=&Page
Count=2147483647&p_gymm=cti110&khbh=2126846&p_kzcs=

然后，转换参数到 kcbpcp 的 execute 能识别的格式：
PageOffset:0,p_gybh:1,p_gnbh:13320001,zllxbh:,p_czzd:10.18.10.48,p_fzdm:,PageCount:2147483647,p_gymm:cti110,khbh:2126846,p_kzcs:

使用 kcbpcp 的 exec 命令
kcbp => exec
input program name:13320001
input parameter:PageOffset:0,p_gybh:1,p_gnbh:13320001,zllxbh:,p_czzd:10.18.10.48
,p_fzdm:,PageCount:2147483647,p_gymm:cti110,khbh:2126846,p_kzcs:
这样就可以重现错误，程序员可以据此对 LBM 进行调试。

8.10.3 LBM 资源消耗过高问题定位

LBM 资源消耗过高包括以下几种情况：
A. 某个 LBM 出现异常而占用太多 CPU 资源，导致系统中出现 CPU 100%情况。
当出现上述情况时，先在任务管理器中记录下正在运行异常的 LBM 的 kcbpas 进程号，然后使用 dumpluw 保存 LBM 运行信息，然后使用异常 kcbpas 进程的输入参数调试 LBM，找到问题，再解决它。
B. 某些ＬＢＭ效率低下，每次执行时间都很长。
使用 KCBP 的服务统计功能，将统计信息保存到日志文件中。然后编辑日志文件，将其中的统计内容复制到 excel 中，计算每个 LBM 的平均运行时间。找出执行时间超过预期的 LBM，对它进行优化，解决效率问题。

8.10.4 LBM 死锁问题处理方法

当多个 LBM 间出现锁资源竞争，出现死锁，会导致 KCBP 系统挂起，不能处理请求。
遇到 KCBP 挂起（死锁）问题时的处理方法：
1 在 KCBP 机器上安转 Windows 的调试工具及符号
网页：http://www.microsoft.com/whdc/devtools/debugging/default.mspx
Install Debugging Tools for Windows 32-bit Version
Download Windows Symbol Packages
注：这项工作不会影响生产系统。

2 当出问题时采取的 2 个操作：
dumpluw > lum.txt
adplus -Hang -pn kcbp.exe -pn kcbpas.exe -pn memhq.exe
注：这个操作平时可以演练，要准备好足够的磁盘空间。adplus 在调试工具目录下，dumpluw 在 kcbp 的 bin 目录。

3 重新启动 KCBP 和 Memhq。

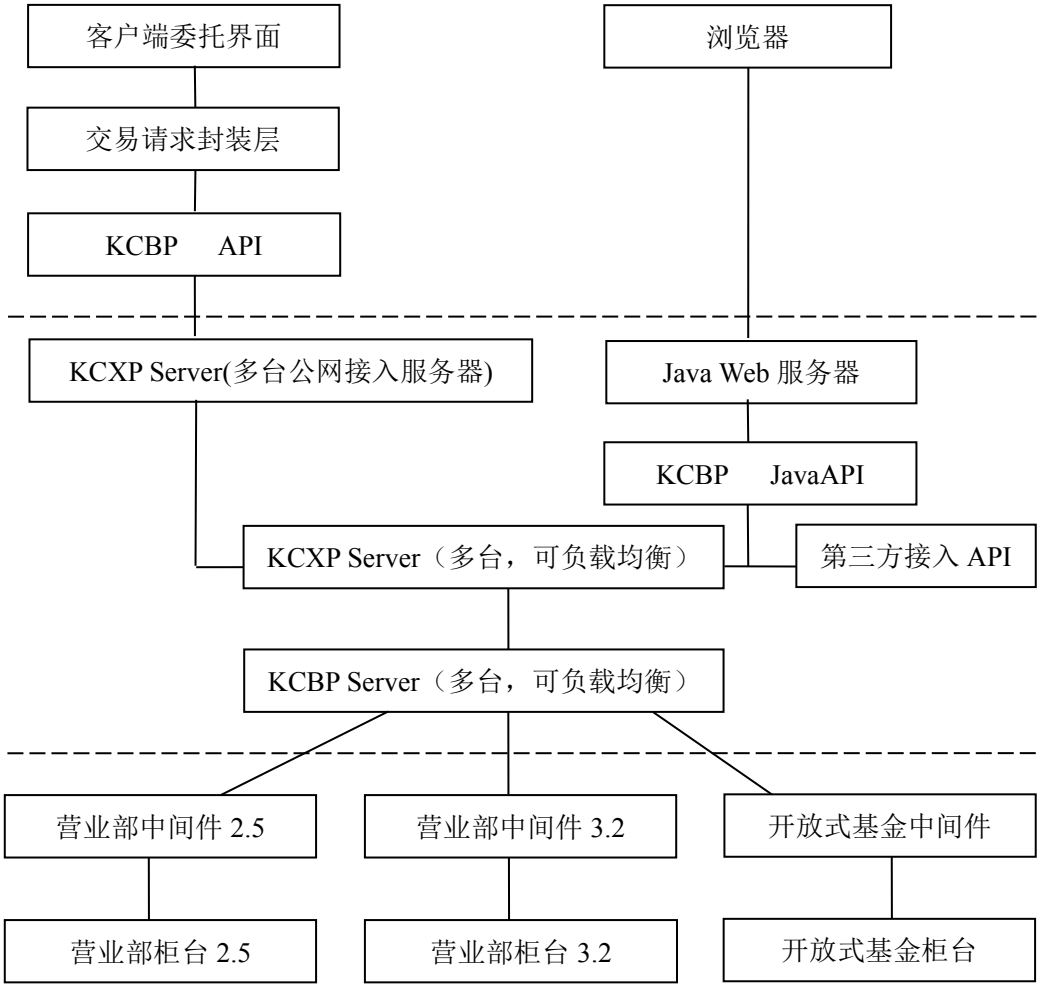
4 压缩 Hang_Mode__Date_*目录，和 luw.txt 一起发送给开发人员。

8.11 应用专题探讨

8.11.1 KCBP 部署规模估算一例

8.11.1.1 引言

我们的一个客户基于现有的分散式的交易系统使用 KCBP 和 KCXP 建设了一个总部集中接入的网上交易系统。他们现在的交易系统，有一些是区域集中系统，还有一些是以营业部为单位的交易系统，他们，由于数据分布在不同的区域中心和营业部，这个系统实际上是一个分布式交易系统。系统结构如下图所示：



总部的 KCXP 用作集中接入，KCBP 用作将交易分发到需要的营业部或区域中心，并负责各系统间报文转换。

系统建成后，需要根据系统的压力情况确定 KCBP 部署规模，下面，我们举例说明系统规模估算的过程。

8.11.1.2 统计服务平均执行时间

8.11.1.2.1 启用服务统计

打开 KCBP 配置界面，在右下角服务统计前面加标志，通知 KCBP 系统在运行时进行统计服务调用数据，统计内容主要包括：服务名称、总调用次数、总执行时间。如果新设置服务统计标致，需要重新启动 KCBP。

8.11.1.2.2 查看服务统计信息

在 KCBP 系统运行一段时间后，比如收市后，打开 KCBP 功能菜单服务统计，出现一个服务统计表格，选择保存按钮，服务统计信息保存到 KCBP 运行日志中。

打开日志，将看到下面形式的统计信息：

[20060405-152430][820094110][bf4- 5ec][10000] 序号 服务名称 调用总数 本次增加 总耗时毫秒

[20060405-152430][820094110][bf4- 5ec][10000] 14 20101245 8 8 2078

[20060405-152430][820094110][bf4- 5ec][10000] 21 20101440 2 2 93

[20060405-152430][820094110][bf4- 5ec][10000] 22 20101445 2 2 219

[20060405-152430][820094110][bf4- 5ec][10000] 23 20101800 1 1 219

其中，最后一项是合计，由总耗时毫秒除以调用总数得到每次服务调用在 KCBP 上的

平均执行毫秒。

业务的平均执行时间是 13696433/44539=307 毫秒。

8.11.1.3 服务统计数据

8.11.1.3.1 按业务调用频率排序

业务	服务名称	调用总数	总耗时毫秒	平均耗时毫秒
开放基金取可申购赎回数量	20104951	2	31	15
交易所开放式基金申购赎回	20101445	3	172	57
基金分红方式设置	30000170	3	296	99
取交易所开放式基金可申购赎回数量	20101440	4	626	156
基金历史委托信息查询	30000280	4	859	215
预售要约收购	20107960	5	562	112
	21102017	5	985	197
新股配号查询(合同号合并)	20102924	5	1126	225
修改股东密码	20104900	7	389	55
成本修改	20104903	10	409	40
基金历史成交信息查询	30000400	10	3061	306
批量委托	20117910	10	6970	697
ETF 网下认购流水查询	20101245	10	29595	2959
基金赎回/预约赎回	30000140	12	2500	225
基金认购/申购/预约认购/预约申购业务	30110130	16	1689	100
开通或取消权证业务	20105913	22	704	32
基金委托信息查询	30000190	24	5925	246
柜台客户资料修改	20107943	27	1638	60
银证转帐查询银行帐户余额	20104901	33	1968	59
银证转帐从资金帐户转到银行帐户	20103902	39	2593	76
查询基金公司	30000210	41	11854	289
历史委托查询	20102921	42	9706	231
银证转帐银行查询	20107909	59	14076	238
基金行情查询	30000110	59	21400	362
当日成交汇总查询，按证券代码和买卖方向合并	20102916	68	15699	230
客户详细资料查询	20107936	78	7906	101
资金流水查询（对帐单）	20102925	95	50686	533

历史成交查询(合同号合并)	20102922	114	70691	620
银证转帐查询	20102917	129	32798	254
客户查询	20102908	155	34576	223
存折炒股委托撤单	20104922	218	5456	25
委托撤单	20103901	305	24264	79
券商发起存折炒股委托确认	20104924	657	581989	885
委托确认	20103900	1124	377487	335
存折炒股查询银行资金	20104925	2090	2895535	1385
当日成交查询(合同号合并)	20102915	2213	457531	206
客户登录验证	20112905	3325	2622066	788
基金份额查询	30000120	3543	1011872	285
查询客户资金	20102906	3753	1087122	389
当日委托查询	20102913	4151	665754	160
证券代码校验	20102909	6570	655549	99
股东股份查询	20102907	15499	2980318	192
合计		44539	13696433	307

注意：由于客户的系统是跨广域网的分布式系统，表格中的平均耗时毫秒，包含系统间跨广域网通讯的时间，客户的广域网带宽是 2M。

8.11.1.3.2 按业务执行时间排序

业务	服务名称	调用总数	总耗时毫秒	平均耗时毫秒
开放基金取可申购赎回数量	20104951	2	31	15
存折炒股委托撤单	20104922	218	5456	25
开通或取消权证业务	20105913	22	704	32
成本修改	20104903	10	409	40
修改股东密码	20104900	7	389	55
交易所开放式基金申购赎回	20101445	3	172	57
银证转帐查询银行帐户余额	20104901	33	1968	59
柜台客户资料修改	20107943	27	1638	60
银证转帐从资金帐户转到银行帐户	20103902	39	2593	76
委托撤单	20103901	305	24264	79
基金分红方式设置	30000170	3	296	99
证券代码校验	20102909	6570	655549	99
基金认购/申购/预约认购/预约申购业务	30110130	16	1689	100
客户详细资料查询	20107936	78	7906	101
预售要约收购	20107960	5	562	112

取交易所开放式基金可申购赎回数量	20101440	4	626	156
当日委托查询	20102913	4151	665754	160
股东股份查询	20102907	15499	2980318	192
206	21102017	5	985	197
当日成交查询(合同号合并)	20102915	2213	457531	206
基金历史委托信息查询	30000280	4	859	215
客户查询	20102908	155	34576	223
新股配号查询(合同号合并)	20102924	5	1126	225
基金赎回/预约赎回	30000140	12	2500	225
当日成交汇总查询，按证券代码和买卖方向合并	20102916	68	15699	230
历史委托查询	20102921	42	9706	231
银证转帐银行查询	20107909	59	14076	238
基金委托信息查询	30000190	24	5925	246
银证转帐查询	20102917	129	32798	254
基金份额查询	30000120	3543	1011872	285
查询基金公司	30000210	41	11854	289
基金历史成交信息查询	30000400	10	3061	306
委托确认	20103900	1124	377487	335
基金行情查询	30000110	59	21400	362
查询客户资金	20102906	3753	1087122	389
资金流水查询（对帐单）	20102925	95	50686	533
历史成交查询(合同号合并)	20102922	114	70691	620
批量委托	20117910	10	6970	697
客户登录验证	20112905	3325	2622066	788
券商发起存折炒股委托确认	20104924	657	581989	885
存折炒股查询银行资金	20104925	2090	2895535	1385
ETF 网下认购流水查询	20101245	10	29595	2959

8.11.1.4 估算 KCBP 系统部署规模

KCBP 从上午 9 点 2 分启动，运行到统计时间 15 点 24 分，共进行了 44539 次服务调用，我们按 4 个小时进行计算平均每秒调用的次数：

44539/（4*60*60）=3.09 笔/秒

计算 KCBP 每个线程每秒的平均处理能力：

1000/307=3.25 笔/秒。

估算 KCBP 处理线程数量：

3.09/3.25=0.95 个。

可见，1 个 KCBP 处理线程即能满足要求 4 个小时处理 44539 的要求。

考虑到请求的执行时间和调用频率存在差异，如存折炒股查询银行资金平均执行时间 1385 毫秒、ETF 网下认购流水查询平均执行时间 2959 毫秒，实际部署时，需要适当提高 KCBP 处理线程的数目，以此提高 KCBP 系统的并行处理能力。在这个系统中，KCBP 处理线程数初步设置为 5 个，以后如果用户增加，压力增大，再考虑适当增加数目；考虑到银证通业务正常情况和异常情况下的通讯时间可能差异较大，异常时，会造成一些 KCBP 线程一定时间内阻塞，因此，还需要适当提高 KCBP 线程数，增加的个数应大于或等于银证通类业务并发数，总数不能超过 kcbpas 线程限制（目前 KCBP 程序 AS 处理线程总数上限是 200 个）；另外，为适应交易系统高可靠性的需求，KCBP 系统需要做双机集群部署。这样的部署的系统，处理能力是 2*5*3.25=32.5 笔/秒。

8.11.1.5 结束语

运维人员可以定期或不定期地观察 KCBP 的服务统计数据，根据系统的压力情况配置一个合理的 KCBP 参数。注意这里强调的是合理值，而不是最大值，如果采用最大值，会占用更多的系统资源（如 socket 连接数目等资源），而系统资源是需要合理配置的，这样的系统才会优化。

当系统中 KCBP 进行集群部署时，估算时每个节点可以独立估算。

8.11.2 KCBP/Win 与 KCXP 系统间连接句柄数目优化

8.11.2.1 问题

在集中交易系统中，如果有很多营业部连接到同一个 KCXP 上，并且每个营业部在 KCXP 在建立了一对请求队列/应答队列，系统中就会存在大量的队列，而当 KCBP 使用缺省的 Isolate 模式与 KCXP 通讯时，每个 KCBP AS 线程为每个队列都建立一个到 KCXP 的连接，当 KCBP AS 线程数目是 M，请求/应答队列数是 N，系统中打开的 socket 数目是 M*N 个，每个 socket 上打开一个队列。当 KCBP 关闭时，用 netstat –n 观察，会发现有很多 Socket 处于 TIME_WAIT 状态，而在 Windows 2003 上，TIME_WAIT 状态缺省保持 2 分钟，如果 KCBP 在这两分钟内重新启动，可能会遇到连接不到 KCXP 的现象，造成这个问题的原因是

系统中的可用 Socket 句柄不足。当 KCBP 与 KCXP 通信时，KCBP 调用 KCXPAPI 发起通讯。KCXPAPI 使用的端口，是由操作系统动态分配的，操作系统查询系统中空闲的 TCP 端口，动态分配的端口也称为临时端口，在 Windows Server 2003 中，临时端口缺省范围从 1025 到 5000 。也就是说，如果 KCBP 在 2 分钟内进行多次启动/关闭操作，当系统中处于 TIME_WAIT 状态的端口大于 3975 个时，KCBP 启动时就产生连接 KCXP 失败的现象。

8.11.2.2 解决方法

解决这个问题的方法有以下几种：

8.11.2.2.1 增加 Windows Server 2003 MaxUserPort 参数

在 Windows Server 2003 中，由 Windows 套接字分配给应用程序的临时 TCP 或 UDP 端口号的最大值是由注册表设置 MaxUserPort 控制的，该参数的默认值为 5000。临时端口从端口号 1025 开始编号。因此，默认情况下，Windows XP 或 Windows Server 2003 会为执行通配绑定的应用程序分配一个范围从 1025 到 5000 的号码。
要在运行 Windows XP 或 Windows Server 2003 的计算机上更改临时端口的最大值，请执行以下操作：
1. 单击开始，再单击运行，键入 regedit.exe，然后单击确定。
2. 找到而后单击以下注册表子项：
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
3. 在编辑菜单上，指向新建，然后单击双字节值。
4. 键入 MaxUserPort，然后按 ENTER。
5. 双击 MaxUserPort 值，然后以十进制或十六进制键入最大值。
键入的数值必须在 5000-65534（十进制）之间。如果此参数设置的值超出有效范围，则使用最接近的有效值（5000 或 65534）。
6. 单击确定。
7. 退出注册表编辑器。
注意：
1. 必须重新启动计算机，方可使 MaxUserPort 注册表设置更改生效。
2. 如果应用程序同时打开大量连接，可以改这个值，缓解句柄不足问题，但当应用程序用尽所有可用的临时端口，系统还是会发生打不开 Socket 的问题。
3. 注意设置保留端口：对于系统中供其他服务程序使用的端口，要在注册表中设置为保留，保留端口不会被系统动态分配给其他应用程序，保留端口范围时，所选择的端口号连续范围必须是从 1025 到 MaxUserPort 设置值（默认值为 5000）或从 49152 到 65535。多个客户端应用程序可保留相同的范围。取消保留（删除保留）时，Windows 套接字会删除它找到的第一个完全包含在取消保留请求内的条目。
还可以执行以下操作，通过注册表来指定保留端口的范围：
1. 单击开始，再单击运行，键入 regedit.exe，然后单击确定。
2. 找到而后单击以下注册表子项：
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

3. 在编辑菜单上，指向新建，然后单击多字符串值。
4. 键入 ReservedPorts，然后按 ENTER。
5. 双击 ReservedPorts 值，使用以下语法键入端口范围：x-y,
要指定单个端口，请对 x 和 y 使用相同的值。例如，要指定端口 4000，请键入 4000-4000。
6. 单击确定。
7. 退出注册表编辑器。

对于 Windows Server 2003 SP1，端口范围如下所示：

- 众所周知的端口（由 IANA 保留）：0 到 1023
- 临时端口：1025 到 MaxUserPort 注册表设置值
- 特定端口：从 0 到 65535 的任何未封锁端口
- 保留端口的可用范围：1025 到 MaxUserPort 以及 49152 到 65535
- 封锁端口的可用范围：MaxUserPort + 1 到 65535

8.11.2.2.2 减少 Windows Server 2003 TIME_WAIT 时间

TcpTimedWaitDelay 位于注册表子项：
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
项：Tcpip\Parameters
数值类型：REG_DWORD - 时间（以秒为单位）
有效范围：30-300（十进制）
默认值：十进制 120
说明：此参数确定连接被关闭时，它停留在 TIME_WAIT 状态的时间长度。当连接处于 TIME_WAIT 状态时，不能重新使用套接字对。这也称为“2MSL”状态，因为根据 RFC，此值应该是网络上最大片段生命周期的两倍。有关更多信息，请参阅 RFC793。
注意：
1. 必须重新启动计算机，注册表设置才能生效。
2. 当KCBP服务器上运行有其他网路服务程序时，调整该值前需要进行整体评估。

8.11.2.2.3 多个营业部共用一个请求队列，减少队列数目

集中交易系统部署时，可以给多个营业部配置同一个请求队列，即在需要共享队列的营业部的 KCXP 上，将请求队列映射为总部 KCXP 上的同一个请求队列。
通常，在总部的 KCBP 中，为每个营业部的请求队列都配置了一个应答队列，而这个应答队列，常用的方法是将它设置成营业部 KCXP 上的远程队列，实际上，应答队列也可以不配远程队列，只设置为总部 KCXP 的本地队列，因为，在现在的 KCBP 上，发送应答采用的是路由方式，即使不映射应答队列为营业部 KCXP 上的远程队列，总部 KCXP 也能正确地将应答送到营业部的应答队列中，之所以能这样，是因为 KCBP 客户端在发送请求消息时，将应答需要回送的队列管理器和应答队列名称同时放到了请求消息中，KCBP 据此通知

KCXP 采用路由方式回送应答。
通过这种方法，我们可以减少 KCBP 与 KCXP 间的 Socket 数目。
注意：

- 这种方法也可以和第一种方法配合使用。
- 如果多个营业部共用一个请求队列时，可能会出现请求响应慢的情况，这时可以增加请求队列的侦听线程数目来提高处理能力。

8.11.2.2.4 KCBP 配置为 combined 通讯方式

当 KCBP 以 combined 方式与 KCXP 通讯时，每个 KCBP AS 线程与请求/应答涉及的每个 KCXP 队列管理器，建立一个连接，在这个连接上，打开所有属于该 KCXP 的队列。当 KCBP AS 线程数目是 M，分别属于 S 个队列管理器，每个请求/应答队列总数目是 Sn 个，系统中打开的 socket 数目是 M*S 个，各 socket 上打开队列是响应队列器上队列总数 Sn。
Combined 方式与 Isolated 方式的区别在于，Combined 方式使用的 socket 少，每个 socket 上打开的队列多，而 Isolated 方式使用的 socket 多，每个 socket 只打开一个队列。
在低版本的 KCXP API 和 Server 上，对每个 socket 能够上打开的队列数目有所限制，打开的队列数目超过 32 个；目前的安装包中，KCXP server 和 KCXP API 上限是 256。如果需要超过 256 个，请与 KCXP 开发部门联系。
这种方式，与第一种方式无冲突，但由于该方式 socket 数目不再是瓶颈，因此，没有必要在去调整 MaxUserPort 参数。

8.11.3 WIN 集中交易系统分离请求到备机优化方案

8.11.3.1 前言

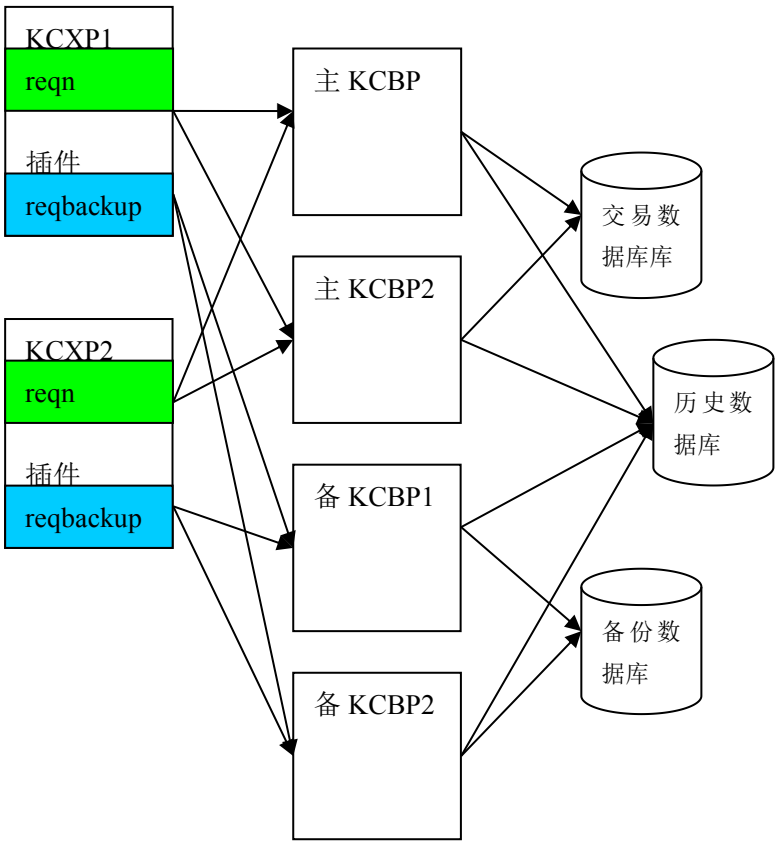
为了提高交易系统的效率，充分利用数据库备机的处理能力，我们可以在备机上部署查询用 KCBP，在总部 KCXP 上配置转发插件，将查询类业务转发到专用查询队列中，由查询 KCBP 专门处理。

8.11.3.2 实现方案

8.11.3.2.1 KCBP 配置

1) 每个核心，部署 2 个专用 KCBP 节点，每个节点分配一个唯一的节点号。这两个节点，可以装在独立的服务器上，也可以装主交易系统的 KCBP 同一台机器的不同目录上。

系统结构如下：



2) 请求/应答队列配置如下：
请求队列 reqbackup
应答队列 ansbackup

注：由于 KCBP 客户端在发送请求时已经指定应答队列，而 KCBP 从请求报文中获得应答回送队列，因此 ansbackup 实际不会用到，配置它只为匹配语法。

3) XA 配置
将 tradedb 配置为备机数据库。
4) Listener 配置
需要将 reqbackup 的 Listener 配为 5 个，这样，请求读取速度快。
5) 其他与主系统上的配置相同。

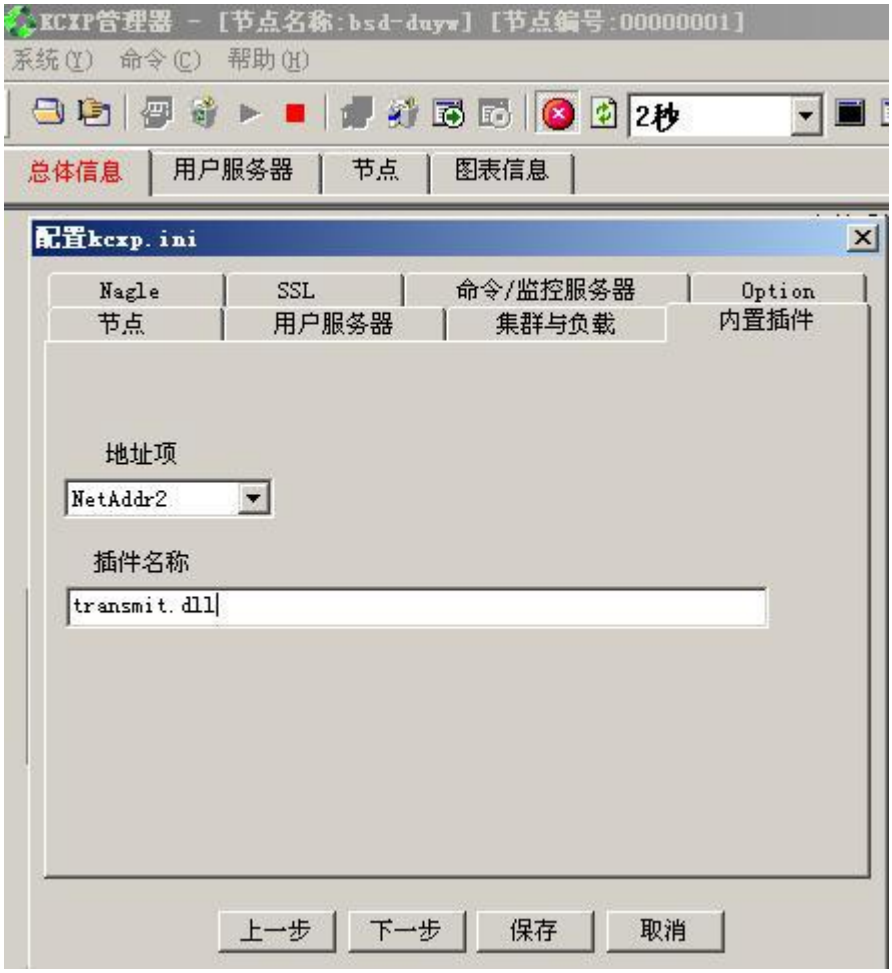
8.11.3.2.2 总部 KCXP 配置

之所以在总部 KCXP 上配置转发插件，而不是配在营业部，是因为这样做的工程量比每个营业部都配置要小得多。当然，如果配在营业部，系统总体的通讯效率要比配在总部高。

1) 运行 xpcc
KCXP:/>user system

KCXP:/>dispexit –a
如果没有 transmit.dll 执行下面命令：
KCXP:/>addexit -f transmit.dll -c transmit
KCXP:/>sync –a

2) 手工配置



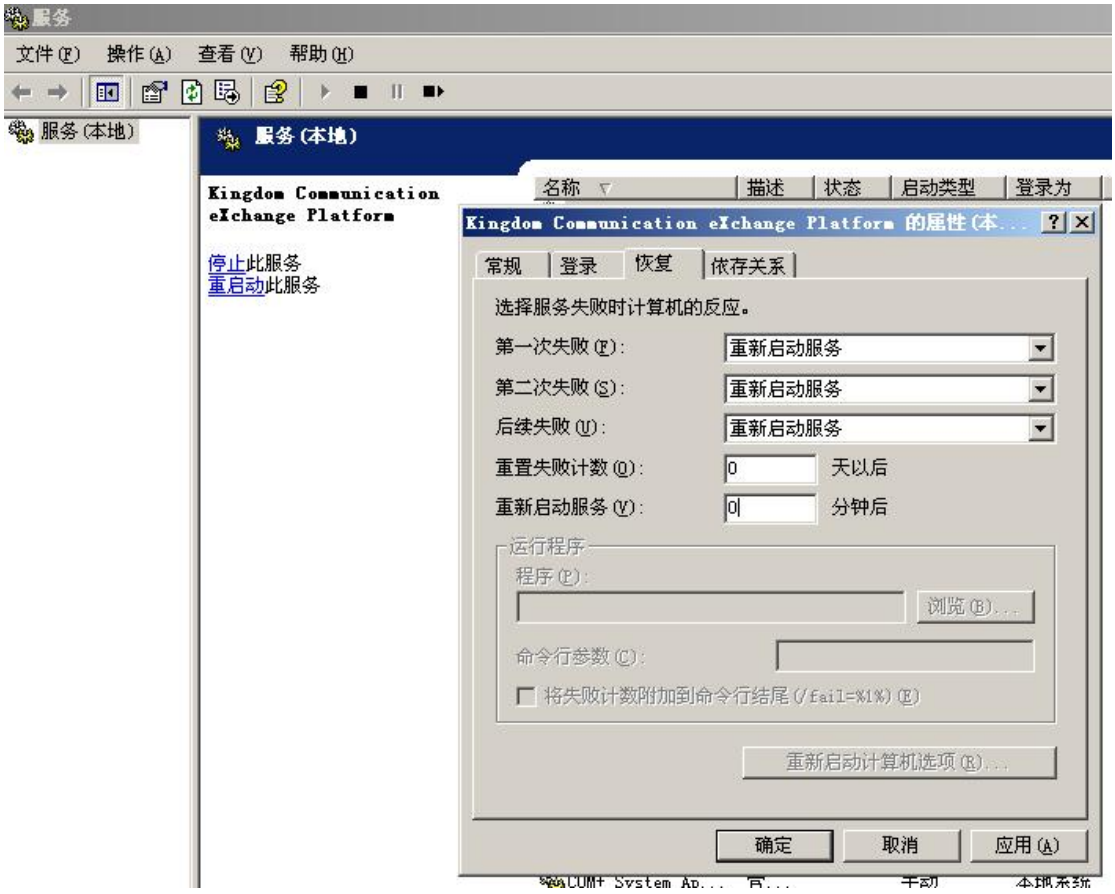
注意：如果还有其他 NetAddr 需要转发，也要同样配置。

3) exit.ini 配置，transmit 节如下配置

```
[transmit]
;报文类型位置
TypeOffset = 2
;报文类型长度
TypeLen = 1
;功能号位置
ServiceOffset = 94
;功能号长度
ServiceLen = 8
;机构代码位置
InstOffset = 113
```

;机构代码长度
InstLen = 4
;报文类型, 请求号, 机构代码, 目标节点编号, 目标队列名(本地队列)
bus1 = 2, 请求号 1, NULL, NULL, reqbackup
;bus2 = 2, 请求号 2, NULL, NULL, reqbackup
;busn = 2, 请求号 n, NULL, NULL, reqbackup

4) 配置总部 KCXP 失败恢复（应急方案）
在服务管理中，选 Kingdom Communication eXchange Platform 服务，按鼠标右键，选者属性，进行下面设置：



8.11.3.3 效率评估

按前面方案部署的系统，可能出现的效率瓶颈在总部的转发插件上，转发插件要对每一个请求进行判断处理，但由于这个插件是 C 写的，代码很短，执行很快，再考虑 KCXP 的处理效率一般不是系统的瓶颈，因此不会对系统效率造成多大影响。至于具体情况，可以测试确认。

8.11.3.4 其他方案探讨

KCBP 从 Build 051123 版本开始，支持自动选择缺省 XA，只要在 KCBPSPD.xml 的 program 行的 xa 属性配置一下，LBM 在执行时就会自动切换到这个 XA 上。
考虑到 GeneralLBMAPI 有设置缺省 XA 的操作，因此，需要对 GeneralLBMAPI 做相应的调整，将这个操作去掉。而有的 LBM 在执行过程中多次切换 XA，这样的 LBM 也需要修改，修改的方法是在切换 XA 前保存当前的 XA，操作完成后，再切换回来。
这个方案的好处是不需要增加 KCBP 节点，只需要配置 LBM 使用的缺省数据库即可，缺点是需要改 GeneralLBMAPI 和部分 LBM。

8.11.4 银证跨系统调用多机协同解决方案

8.11.4.1 引言

跨系统调用，广泛存在于证券交易系统之中。最典型的就是银证转账和银证通业务。这类跨系统调用的业务模式，要求证券交易系统与银行系统之间要进行联机交易，并且，由于证券交易系统与银行系统分别隶属于不同公司，各方只能控制自己系统正常运行，无法确保对方系统的是否正常运行，这样，当一方系统运出现运行故障时或双方通讯系统出现故障，往往会引发另一方系统也发生问题。

下面我们对银证系统间通讯的各种问题进行分析，并论述如何运用金证交易中间件 KCBP 的独特功能设计，解决跨系统调用的各种问题。

8.11.4.2 跨系统调用面临的问题

目前国内各证券交易系统厂商提供的交易系统中，银证业务主流处理技术是不落地方式，即由交易中间件负责处理银证业务调用，而证券行业使用的较易中间件，都是各交易系统厂商自行开发的，这些中间件在处理银证业务时，目前存在以下几个典型的问题：

8.11.4.2.1 系统不可用

目前多数的交易系统中，系统间通讯业务都由交易中间件上的业务逻辑程序完成，对于

银证类业务，业务逻辑要与银证平台通讯，调用方式一般都采用同步调用，每个调用都有一定的超时时间，当银行端系统发生异常时，中间件的业务逻辑处理线程处于等超时的状态，当这类请求很多时，可能造成交易中间件的全部处理线程都处于等超时状态，这时，交易中间件系统就没有可用线程处理其他业务，系统在一段时间内就处于不可用状态。

8.11.4.2.2 银行间互相干扰

有的交易中间件提供了业务名称或业务分组控制并发数的功能，这可以限制银证类业务异常对交易中间件系统造成的影响，但交易中间件无法根据请求报文内容控制并发数，因此，当一个银行出问题时，可能使这个业务或该类业务占用所有并发数，影响其他银行的可用性。

8.11.4.2.3 系统繁忙

有的交易系统中，针对各个银行都设了一个并发计数器，通过它来解决银行互相干扰问题，但这种设计会遇到问题：

首先有可能会出现这样的情况，当一家银行并发数超过预定的并发数限制时，由于超过限制的请求不能等待（如等待又会造成处理线程阻塞而导致系统不可用），都直接向前端返回失败，提示系统繁忙；

其次，系统可能连接有许多家银行，每家银行最少要给一个并发数，当总并发数超过交易中间件的处理线程数(或这类业务的总并发数上限)时，在某一个时刻，如果每家银行都有请求需要处理时，交易系统也会提示前端系统繁忙。

8.11.4.3 解决方案

前面问题的根源，在于业务程序对交易中间件处理线程的争夺上。不论交易系统厂商开发的交易中间件还是国外商品化的交易中间件，都存在同样的问题。这类问题实质是并行处理算法的设计问题，它对交易中间件的设计提出了挑战。对于这类并行处理问题，可以通过多个交易中间件协同来解决，在主处理机之外，增加用于处理系统间通讯业务的辅助处理机。主处理机和辅助处理机之间要有一套严密的调度机制。

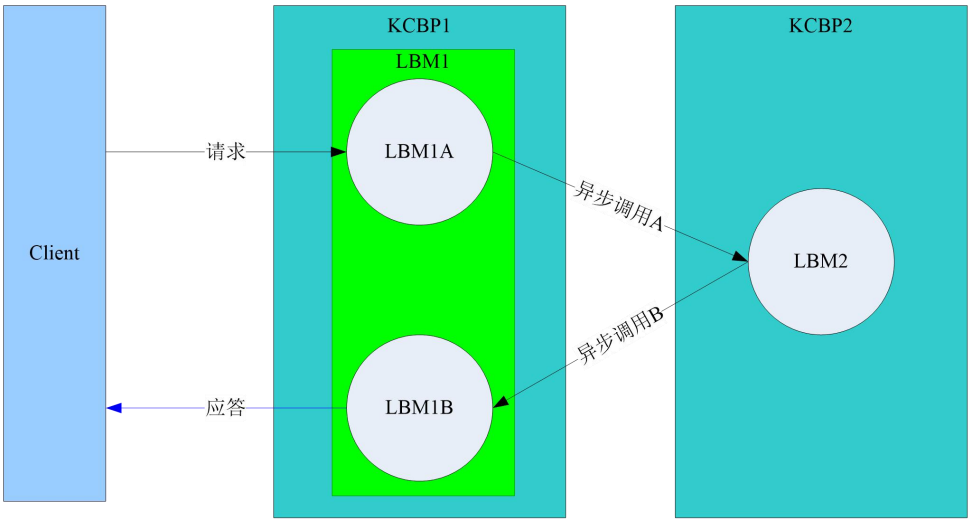
下面，我们结合金证公司 KCBP 的独特功能设计，提出以下两种基于多 KCBP 协同的

解决方案。

8.11.4.3.1 分治法

分治法，就是通过拆分业务程序，解决并行处理问题。业务拆分就是将传统证券系统中的 1 个银证业务分为几个子业务，两个业务分别交给两个 KCBP 处理。第一个 KCBP 做银证通讯前和通讯后的处理，第二个 KCBP 做银证通讯处理。第一 KCBP 是处理所有业务的主系统，第二个 KCBP 是负责处理银证业务的辅助系统。第一个子业务在主 KCBP 上运行，当需要进行银证通讯时，通过编程实现转移处理环境（包含请求数据）给第二个 KCBP，不向前端返回数据，然后结束自己，被释放的 KCBP 业务处理线程就可以处理其他业务。第二个 KCBP 进行银证间通讯处理，在取得应答后回调第一个 KCBP，并将转移出来的处理环境回送给第一个 KCBP，第一个 KCBP 继续进行后续业务处理，完成后向前端返回数据。两个 KCBP 间的通讯，采用异步调用方式。所谓异步，就是调用发出后不等结果，立即返给调用者，调用者不取结果，或可以在合适时刻查询结果。这个解决方案，可以解决前面所述问题。

8.11.4.3.1.1 业务处理流程



业务处理程序，在 KCBP 上叫做 LBM。

业务处理由 LBM1A，LBM1B，LBM2 组成，处理流程如下：

1. Client 同步调用 KCBP1 上的 LBM1A。
2. LBM1A 进行处理，当需要 KCBP2 上的 LBM2 参与处理时，异步调用 KCBP2 上的 LBM2。
3. LBM1A 结束，但不返回数据给 Client。

4. LBM2 处理结束后，异步调用 KCBP1 上的 LBM1B。
5. LBM1B 完成 LBM1A 的后续处理，然后返回应答给 Client。

8.11.4.3.1.2 程序实现注意事项

8.11.4.3.1.2.1 客户端

客户端调用业务名称是 LBM1，它不关心后台的处理流程。

8.11.4.3.1.2.2 LBM1A 实现注意事项

- 调用 KCBP_Call 前准备 LBM2 需要的参数，同时，也要准备 LBM1B 需要的参数（LBM2 调用 LBM1B 时再将这些参数送回）。LBM1B 的参数，包括四个 KCBP 系统参数，是 char szMsgId[25]、char szGroupId[25]、char szDestNode[10]、char szDestQ[33]，这几个参数使用下面的代码获得：
  - KCBP_System(hHandle, 4,szMsgId);
  - KCBP_System(hHandle, 6,szGroupId);
  - KCBP_System(hHandle, 19, szDestNode);
  - KCBP_System(hHandle, 20, szDestQ);
- 使用 KCBP_Call 的异步方式调用 KCBP2 上的 LBM2，代码如下：KCBP_Call(hHandle, “LBM2”, KCBP_NOBLOCK); //要在 KCBP1 上定义远程 LBM2
- 在调用 KCBP_Exit 之前，需要调用 KCBP_System(hHandle, KCBP_SET_RETURNFLAG,0);这行代码的作用是让 KCBP_Exit 函数执行时不返回数据给客户端。

8.11.4.3.1.2.3 LBM2 实现注意事项

- LBM2 收到 LBM1A 传递过来的参数中包括 LBM1B 需要的参数，它需要将这项参数传递给 LBM1B。
- LBM2 使用 KCBP_Call 异步调用 LBM1B。
- 在调用 KCBP_Exit 之前，需要调用 KCBP_System(hHandle, KCBP_SET_RETURNFLAG,0);这行代码的作用是让 KCBP_Exit 函数执行时不返回数据给 LBM1A。

8.11.4.3.1.2.4 LBM1B 实现注意事项

- 需要注意接收 LBM2 传递的四个 KCBP 系统参数，是 char szMsgId[25]、char szDestNode[10]、char szDestQ[33]，并使用这几个参数，代码如下：
  - KCBP_System(hHandle, 31,szMsgId);
  - KCBP_System(hHandle, 33,szGroupId);
  - KCBP_System(hHandle, 34, szDestNode);
  - KCBP_System(hHandle, 35, szDestQ);

8.11.4.3.1.2.5 其他

- 上图中，KCBP2 可以换成银证平台，银证平台可以使用 KCBP 客户端的异步调用

函数 KCBPCLI_ACallProgramAndCommit 调用 LBM1B。

- LBM1A 也可以直接使用 KCXPXA 提供的写队列功能与银证平台通讯，这是目前金证公司新一代集中交易系统采用的方式。
- LBM1A 通过 LBM2 传递给 LBM1B 的参数，也可以通过数据库传递，这时，LBM1A，只须通过 LBM2 传递给 LBM1 一个索引值（通过它定位数据库中保存的参数）。除了数据库之外，也可以通过 KCXP 队列或其他可共享资源保存参数。

8.11.4.3.2 单线程多任务

目前的交易中间件产品，包括商业中间件 CICS、TUXEDO、也包括各种 J2EE 中间件，以及各种国产中间件，事实上都遵循线程单并发原则，这个原则是这样的：

中间件有 M 个业务处理线程（进程），在某一时刻最大并发处理能力是 M 笔业务，这时每个线程处理一笔业务。每个线程，在任何时间段内，对业务程序的处理都是串行的，一个业务程序一旦开始，就必须运行到结束。系统以业务程序为是最小执行单位。

现在，在 KCBP 上，交易中间件产品的思维定式已经不复存在，我们已经在 KCBP/WIN 上成功做到：在一定的条件下，单线程多并发处理，即一个线程在一个时间段内并发处理 N（N >=1）笔业务。这时，业务程序不再串行处理，系统不再以整个业务程序为最小执行单位，KCBP 自动将业务程序分割成多个程序单元，以业务单元为最小执行单位。一个线程，在一定时间段内，可以并发运行多个业务程序，并在不同的业务程序的业务单元间自动切换处理。

有了这项技术，前面的问题就迎刃而解。

KCBP 在处理存在系统间通讯的 LBM 时，会自动将业务程序拆分为业务单元，然后按业务单元进行处理。具体地讲，在第一个 KCBP 处理的业务程序 A 将请求（比如转账）发送给负责银证通讯处理的第二个 KCBP 后，第一个 KCBP 自动保存业务执行环境，并将业务程序 A 从线程中卸掉，然后处理其他的业务程序，当第二个 KCBP 向第一个 KCBP 返回结果时，第一个 KCBP 在当前正在处理的业务程序完成一个单元后，恢复前面保存过的业务程序 A 的执行环境，继续业务程序 A 的处理。系统间通讯进行等待时，不占用中间的处理线程，可以并行处理其他业务。

对业务程序来讲，KCBP 单线程多任务比分治法更简单，主 KCBP 上的业务程序不用分拆,通讯前、通讯中、通讯后的各种处理，业务程序不用关心，完全由 KCBP 自动处理，KCBP 的业务处理线程也不会为等待系统间通讯结果而阻塞。

KCBP 系统单线程多任务功能由 KCBP1PCAsync.dll 接口提供。

8.11.4.4 结束语

上面提出的解决方案，是基于多交易中间件协同的一种解决方案，而其之所以能够实现，完全依赖于 KCBP 的强大功能，包括系统间异步调用、前端调用保持、单线程多任务等。据我们所知，到目前为止，还没有哪种其他厂商提供的交易中间件，能提供同样的功能。

9. 常见问题 Q&A

9.1 常见问题

1. 向 KCBP 发送请求，KCBP 没有反映。处理办法：首先请设置 KCBP 的 license，重启 KCBP。然后检查请求/应答队列是否与 KCBP Listener 配置相符。
2. 返回码 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。
3. Kcbpcp 连接报错：Login error : Authentication error。处理办法： Connect 使用的用户需要在 KCBP 服务端的 user 中配置 group 为 manage。KCBP 的 User 支持 manage 和 application 两个组，manage 有管理权限，application 有 LBM 调用权限。
4. 1002 被调用的 LBM 在 KCBP Server 端未定义。
5. 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)。
6. 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。
7. 2003 表示连接句柄错误。
8. 2011 表示调用超时，主要发生在等待结果阶段。
9. 2054、2055 表示 Socket 连接发生中断。
10. 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。
11. 2016 表示 KCXP 的插件加载失败，原因是找到数据文件 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。
12. 2300 SSL 连接错误，原因是客户端和 KCXP Server 端 SSL 设置不匹配。
13. KCBP 启动时报”2054 KCXP_RC_CONNECTION_BROKEN”错误：
    检查/kcbp/bin/目录下的 kcxpapi.ini，看此文件中配的 KCXP 的 IP 地址和端口是否正确，检查该 KCXP 是否正常运行。
14. KCBP 启动时报” 2067 KCXP_RC_READ_CONF_Q_MGR_NAME_ERROR”错误：
    检查/kcbp/bin/目录下的 kcxpapi.ini。看此文件的 QmgrName 是否和 KCBPConf.xml 中队列配置中的 queuemanager 是否一致，两处的值一致才可以。
15. KCBP 启动时报” 2103 KCXP_RC_PLUGIN_NOT_FOUND” 错误：
    在/kcbp/bin/目录下没有 kcxpplugin.dat 或 kcxplugin.dat 文件版本不对。
16. 客户端返回”2011 KCXP_RC_NO_MSG_AVAILABLE”错误：
    客户端没有取得应答结果，此时要检查 KCBP 和 KCXP 的队列配置及 KCBP 的运行状态，可以用 XPCC 命令查看客户端使用队列的消息进出数，以确定在哪个环节存在问题。
17. KCBP 启动时报”2003 KCXP_RC_HCONN_ERROR”错误：

请求队列配置成了远端队列，改为本地队列即可。

18. 在调用 LBM 时，报”1002KCBP_ERROR_UNDEFINED_LBM”：
在 KCBPSPD.xml 中没有定义该 LBM 的名字，在此文件中增加该 LBM 的定义即可。
19. 在调用 LBM 时，报”1004KCBP_ERROR_INVALID_LBM_PATH”：
在 KCBPSPD.xml 的 LBM 定义中，DLL 的路径错误，修改该路径即可。
20. 在调用 LBM 时，报”1006KCBP_ERROR_INVALID_LBM_MODULE”：
在 KCBPSPD.xml 的 LBM 定义中，LBM 的函数名定义错误，修改该 LBM 的函数名即可
21. 在配置 ODBC 别名时，应使用系统 DSN 项的配置。
22. 为何在任务管理器中不能终止 KCBP 进程和 KCBPAS 进程？
这是因为 KCBP 的错误修复技术在起作用。KCBPAS 进程是 KCBP 进程产生的，KCBP 进程对 KCBPAS 进程进行监控，如果发现有 KCBPAS 进程异常终止，KCBP 会进行修复，产生新的 KCBPAS 进程。而 KCBP 进程之所以不能被终止，是因为有 KCBPDaemon 进程对其进行监控，发现其异常终止，会重新创建 KCBP 进程。如果确实要终止 KCBP 进程，请先终止 KCBPDaemon 进程。
23. 在同一台电脑上运行两个 KCBP 需要注意什么？
注意两点：一是 KCBP 安装到不同的目录下，二是 KCBP 配置中的 nodeid 不能相同。

9.2 KCBP 错误码

错误代码	错误宏定义	说明
1000	KCBP_ERROR_BIZ_READ_REQUEST	读请求错误
1001	KCBP_ERROR_BIZ_LOADLBM_FAILURE	加载 LBM 失败
1002	KCBP_ERROR_UNDEFINED_LBM	没有定义的 LBM
1003	KCBP_ERROR_UNDEFINED_LBM_PATH	没定义的 LBM 路径
1004	KCBP_ERROR_INVALID_LBM_PATH	不合法的 LBM 路径
1005	KCBP_ERROR_UNDEFINED_LBM_MODULE	没有定义 LBM 模块
1006	KCBP_ERROR_INVALID_LBM_MODULE	不合法的 LBM 模块
1007	KCBP_ERROR_UNDEFINED_LBM_PRIORITY	没有定义的 LBM 优先级
1008	KCBP_ERROR_UNDEFINED_LBM_ACM	没有定义的 LBM ACM
1009	KCBP_ERROR_UNDEFINED_LBM_RSL	没有定义的 LBM RSL
1010	KCBP_ERROR_UNDEFINED_LBM_TIMEOUT	没有定义的 LBM 超时
1011	KCBP_ERROR_LBM_DLOPEN_FAILURE	LBM 动态库打开失败
1012	KCBP_ERROR_LBM_DLSYM_FAILURE	LBM 函数打开失败
1013	KCBP_ERROR_INSUFFICIENT_SPACE	空间不足
1014	KCBP_ERROR_BIZ_EXCEPTION	LBM 异常
1015	KCBP_ERROR_REALLOC	重新分配内存失败
1016	KCBP_ERROR_NESTED_LEVEL	LBM 调用嵌套级别过多
1017	KCBP_ERROR_SERVICE_UNAVAILABLE	LBM 状态为不可用
1018	KCBP_ERROR_OUT_OF_SERVICE	LBM 不在可用时间段内

1020	KCBP_ERROR_BIZ_READ_REQUEST_TIMEOUT	读请求超时
1021	KCBP_ERROR_BIZ_READ_REQUEST_FAILURE	读请求失败
1022	KCBP_ERROR_BECANCELED	业务被关闭
1023	KCBP_ERROR_BIZ_OVERLOAD_REQUEST	接收请求超过了需要的长度
1024	KCBP_ERROR_READ_ANSWER_FAILURE	读应答失败
1025	KCBP_ERROR_ANSWER_OVERFLOW	内部调用接收应答出错
1026	KCBP_ERROR_UNDEFINED_TRANSFER_NODE	没有定义的 transfer 结点
1027	KCBP_ERROR_CIRCULAR_TRANSFER	循环 transfer 错误
1028	KCBP_ERROR_TRANSFER_IPC_READ	IPC_Read 错误
1029	KCBP_ERROR_TRANSFER_ANSWER	Transfer 时，读应答错误
1030	KCBP_ERROR_TRANSFER_IPC_WRITE	Transfer 时，发送失败
1031	KCBP_ERROR_ISC_INIT_INTERNAL_QUEUE	系统间调用初始化内部队列失败
1032	KCBP_ERROR_ISC_INVALID_RECEIVE_QUEUE	系统间调用不合法的接收队列
1033	KCBP_ERROR_ISC_INVALID_SEND_QUEUE	系统间调用不合法的发送队列
1034	KCBP_ERROR_ISC_WRITE_INTERNAL_QUEUE	系统间调用写内部队列错误
1035	KCBP_ERROR_ISC_READ_INTERNAL_QUEUE	系统间调用读内部队列错误
1036	KCBP_ERROR_OPEN_MSGQ_FOR_WRITE	写时打开队列失败
1037	KCBP_ERROR_WRITE_INTERNAL_QUEUE_TIMEOUT	写内部队列超时
1038	KCBP_ERROR_WRITE_INTERNAL_QUEUE	写内部队列失败
1039	KCBP_ERROR_OPEN_MSGQ_FOR_READ	读时打开队列失败
1040	KCBP_ERROR_READ_INTERNAL_QUEUE_TIMEOUT	读内部队列超时
1041	KCBP_ERROR_READ_INTERNAL_QUEUE	读内部队列失败
1042	KCBP_ERROR_INVALID_PACKET_INDEX	包索引不合法
1043	KCBP_ERROR_INVALID_BIZ_INDEX	工作线程索引不合法
1044	KCBP_ERROR_LISTEN_TIMEOUT	侦听队列超时
1045	KCBP_ERROR_INVALID_XML	
1046	KCBP_ERROR_HANDLE	句柄错误
1047	KCBP_ERROR_GETFUNCTION	管理命令串取功能名错误
1048	KCBP_ERROR_INVALID_COMMAND	管理命令串中命令不合法
1049	KCBP_ERROR_INVALID_ANSWER	管理命令执行后应答不合法
1050	KCBP_ERROR_INVALID_SPD	
1051	KCBP_ERROR_NO_AUTHENTICATION	执行权限不够
1052	KCBP_ERROR_SOP_PACKET_OVERFLOW	
1053	KCBP_ERROR_INVALID_LBM_NAME	LBM 名不合法
1054	KCBP_ERROR_NULL_SPD	SPD 为空
1055	KCBP_ERROR_INVALID_SPD_HOST	LBM 定义的 HOST 值不合法
1056	KCBP_ERROR_UNAVAILABLE_FUNCTION	
1057	KCBP_ERROR_INVALID_SENDNODE	
1058	KCBP_ERROR_INVALID_PACKAGETYPE	不合法的包类型
1059	KCBP_ERROR_DEPUTY_ANSWER	XA 的应答写入内部队列时有错

		误
1060	KCBP_ERROR_DEPUTY_READ	读 XA 应答时有错误
1061	KCBP_ERROR_DEPUTY_WRITE	写 XA 请求时有错误
1062	KCBP_ERROR_INVALID_REQUEST	不合法的请求
1063	KCBP_ERROR_QUEUED_TIMEOUT	执行请求超时
1064	KCBP_ERROR_XA_WRITE_TIMEOUT	写 XA 超时
1065	KCBP_ERROR_XA_READ_TIMEOUT	读 XA 超时
1066	KCBP_ERROR_ISC_NOAVAILABLE_BUFFER	异步远程调用后系统可用空间不足
1067	KCBP_ERROR_XA_READ_INVALID_TIMESTAMP	异步远程调用后时间戳不合法
1068	KCBP_ERROR_INVALID_LICENSE	授权不合法
1080	KCBP_ERROR_UNDEFINED_ID	发布订阅中没有定义的 ID
1081	KCBP_ERROR_UNSUBSCRIBE	发布订阅中取消订阅错误
1082	KCBP_ERROR_SUBSCRIBE	发布订阅中订阅失败
1083	KCBP_ERROR_PUBLISH	发布订阅中发布出错
1084	KCBP_ERROR_UNDEFINED_TOPIC	发布订阅中缺少主题
1085	KCBP_ERROR_UNDEFINED_EXPIRY	发布订阅中缺少过期时间
1086	KCBP_ERROR_UNDEFINED_INITSTRING	发布订阅中缺少 initstring
1087	KCBP_ERROR_UNDEFINED_SYSTEM_CALL	发布订阅中缺少功能号
1088	KCBP_ERROR_INVALID_SYSTEM_CALL	发布订阅中功能号错误
1089	KCBP_ERROR_DEREGPUB	发布订阅中删除发布者失败
1090	KCBP_ERROR_REGPUB	发布订阅中注册发布者失败
1091	KCBP_ERROR_UNDEFINED_PUBLISH_PRIORITY	发布中缺少优先级
1092	KCBP_ERROR_UNDEFINED_PUBLISH_EXPIRY	发布中缺少过期时间
1100	KCBP_ERROR_UNDEFINED_XA	没定义的 XA
1101	KCBP_ERROR_NO_XA	没有指定 XA
1102	KCBP_ERROR_NO_XA_ITEM	没有指定 XA 名字
1103	KCBP_ERROR_XA_END	XA End 错误
1201	KCBP_ERROR_DEPUTY_NO_TPM	无可用代理
1202	KCBP_ERROR_DEPUTY_UNDEFINED_TPM	代理结点没有定义
1203	KCBP_ERROR_DEPUTY_TPM_CALL_FAILURE	代理调用失败
1204	KCBP_ERROR_DEPUTY_TPM_NEXT_CALL_FAILURE	代理调用失败
1301	KCBP_ERROR_DISPATCH_UNDEFINED_PROGRAM	没有定义的 LBM
1205	KCBP_XA_CONCURRENCE_LIMIT	XA 并发达到上限

9.3 KCBPFactory 错误码

错误代码	错误宏定义	说明
1501	KDMID1PC_ERROR_UNRESOLVED_API	动态库中找不到 Factory 的函数
1502	KDMID1PC_ERROR_LOADXML	加载 XML 失败
1503	KDMID1PC_ERROR_LOCKTIMEOUT	加锁超时
1504	KDMID1PC_ERROR_INVALID_PARAMETER	参数错误
1505	KDMID1PC_ERROR_UNDEFINED_RULE	没有定义的规则
1506	KDMID1PC_ERROR_CREATE_REQ_OBJECT	请求句柄初始化失败
1507	KDMID1PC_ERROR_CREATE_ANS_OBJECT	应答句柄初始化失败
1508	KDMID1PC_ERROR_SAVE_REQUEST	请求转换失败
1509	KDMID1PC_ERROR_SAVE_ANSWER	应答转换失败
1510	KDMID1PC_ERROR_INVALID_INVOKE	不合法的调用
1511	KDMID1PC_ERROR_INVALID_MESSAGE_INPUT	输入消息不合法
1512	KDMID1PC_ERROR_INVALID_MESSAGE_INPUT_DEFINITION	输入消息定义不合法
1513	KDMID1PC_ERROR_INVALID_DICTIONARY_ENTRY	不合法的字典入口
1514	KDMID1PC_ERROR_BUFFER_OVERFLOW	缓冲区溢出
1515	KDMID1PC_ERROR_SENDREQUEST	发送请求失败
1516	KDMID1PC_ERROR_WRITE_EXCEPTION	发送请求异常
1517	KDMID1PC_ERROR_INSUFFCIENT_MEMORY	内存不足
1518	KDMID1PC_ERROR_OPEN_CURSOR	打开结果集失败
1519	KDMID1PC_ERROR_FETCHROW	结果集取下一行失败
1520	KDMID1PC_ERROR_INVALID_ANSWER	应答不合法
1521	KDMID1PC_ERROR_CALL_FAIL	调用失败
1522	KDMID1PC_ERROR_INVALID_COLUMN_INFO	不合法的列信息
1523	KDMID1PC_ERROR_INVALID_COLUMN_COUNT	不合法的列数
1524	KDMID1PC_ERROR_APPEND_TABLE	增加新结果集失败
1525	KDMID1PC_ERROR_INVALID_ANSWER_BUFFER	应答缓冲区错误
1526	KDMID1PC_ERROR_READ_ANSWER_BUFFER	读应答缓冲区失败

1527	KDMID1PC_ERROR_COLUMN_INFO_OVERFLOW	列信息太长溢出
1528	KDMID1PC_ERROR_INVALID_ANSWER_TYPE	不合法的应答类型
1529	KDMID1PC_ERROR_KDMIDAPI_READ_ANSWER	读应答失败
1530	KDMID1PC_ERROR_APPEND_ANSWER_ROW	增加应答行失败
1531	KDMID1PC_ERROR_SET_COLUMN_VALUE	设置列值失败
1532	KDMID1PC_ERROR_SAVE_ANSWER_ROW	保存应答行失败
1533	KDMID1PC_ERROR_RECURSIVE_ERROR	递归调用越过最大次数
1534	KDMID1PC_ERROR_INVALID_ANSWER_HANDLE	不合法的应答句柄
1535	KDMID1PC_ERROR_FORMAT_REQUEST	请求格式错误
1536	KDMID1PC_ERROR_INVALID_MESSAGE_OUTPUT	Output 节错误
1537	KDMID1PC_ERROR_INVALID_MESSAGE_OUTPUT_DEFINITION	Outout 节定义错误
1538	KDMID1PC_ERROR_XML_SYNTAX	XML 定义错误
1539	KDMID1PC_ERROR_CREATE_DBF_FAIL	建立 DBF 失败
1540	KDMID1PC_ERROR_SQLMORERESULT	取下一结果集失败
1541	KDMID1PC_ERROR_APPEND_DBF_RECORD	DBF 追加记录失败
1542	KDMID1PC_ERROR_INVALID_EXPRESSION	不合法的表达式
1543	KDMID1PC_ERROR_UNDEFINED_FUNCTION	没有定义的函数
1544	KDMID1PC_ERROR_UNLOADED_FUNCTION	没有加载的函数
1545	KDMID1PC_ERROR_FUNCTION_PARAMETER_MISMATCH	函数的参数不匹配
1546	KDMID1PC_ERROR_FUNCTION_FAIL	调用函数失败
1547	KDMID1PC_ERROR_INVALID_IMDB_HANDLE	不合法的内存数据库句柄
1548	KDMID1PC_ERROR_SQL_STATEMENT_OVERFLOW	SQL 语句超长溢出
1549	KDMID1PC_ERROR_EXEC_SQL_STATEMENT	执行 SQL 错误
1550	KDMID1PC_ERROR_CREATE_TEMP_OBJECT	建立临时 KCBP 解析器失败
1551	KDMID1PC_ERROR_WAIT_CACHE_FAILED	缓存操作加锁超时

9.4 KCXP 错误码说明

错误代码	错误宏定义	说明
10	KCXP_RC_ALLOC_MEMORY_FAILED	分配内存失败
100	KCXP_RC_INVALID_PARM	非法参数
101	KCXP_RC_MESSAGE_OVER_LIFETIME	消息生命周期已过了
110	KCXP_RC_IDENTIFY_NAME	用户名错误
111	KCXP_RC_IDENTIFY_PASSWORD	用户密码错误
120	KCXP_RC_INVALID_QMGRNAME	队列管理器名不合法
121	KCXP_RC_INVALID_NODECODE	结点错误
200	KCXP_RC_ROUTE_NONODE	没有路由结点
201	KCXP_RC_NO_QUEUE	没有队列
202	KCXP_RC_NOT_FREE_COMM_SLOT	没有空闲的通讯槽了
301	KCXP_RC_QUEUE_FILLED	队列已填充
302	KCXP_RC_QUEUE_INSERT	队列已经插入了
303	KCXP_RC_QUEUE_GET	队列已经取了
304	KCXP_RC_QUEUE_MAXMESSAGE_LENGTH	队列已达最大消息长度
305	KCXP_RC_QUEUE_PROHIBIT_PUT	
306	KCXP_RC_QUEUE_MAX_DEPTH	队列已达最大深度
400	KCXP_RC_RECEIVE_FAILED	接收失败
401	KCXP_RC_SEND_FAILED	发送失败
2002	KCXP_RC_LIFETIME_ERROR	生命周期错误
2003	KCXP_RC_HCONN_ERROR	HCONN 错误
2004	KCXP_RC_HOBJ_ERROR	OBJ 错误
2005	KCXP_RC_MAX_CONNS_LIMIT_REACHED	到达最大连接限制
2006	KCXP_RC_MD_ERROR	MD 错误
2007	KCXP_RC_MISSING_RESPONSE_Q	丢失队列响应
2008	KCXP_RC_MSG_TYPE_ERROR	消息类型错误
2009	KCXP_RC_MSG_TOO_BIG_FOR_Q	队列太多了
2010	KCXP_RC_MSG_TOO_BIG_FOR_QMGR	队列管理器太多了
2011	KCXP_RC_NO_MSG_AVAILABLE	没有可用的消息，接收超时
2012	KCXP_RC_NO_MSG_UNDER_CURSOR	游标里没有消息了
2013	KCXP_RC_NOT_OPEN_FOR_BROWSE	不能以 BROWSE 方式打开队列
2014	KCXP_RC_NOT_OPEN_FOR_GET	不能以 GET 方式打开队列
2015	KCXP_RC_NOT_OPEN_FOR_INQUIRE	不能以 INQUIRE 方式打开队列
2016	KCXP_RC_NOT_OPEN_FOR_PUT	不能以 PUT 方式打开队列
2017	KCXP_RC_NOT_OPEN_FOR_SET	不能以 SET 方式打开队列
2018	KCXP_RC_OBJECT_CHANGED	OBJ 改变了
2019	KCXP_RC_OBJECT_IN_USE	OBJ 已在使用
2020	KCXP_RC_OBJECT_TYPE_ERROR	OBJ 类型错误

2021	KCXP_RC_OD_ERROR	OD 错误
2022	KCXP_RC_OPTION_NOT_VALID_FOR_TYPE	TYPE 选项不合法
2023	KCXP_RC_OPTION_ERROR	选项错误
2024	KCXP_RC_PERSISTENCE_ERROR	权限错误
2025	KCXP_RC_PERSISTENT_NOT_ALLOWED	权限不允许
2026	KCXP_RC_PRIORITY_EXCEEDS_MAXIMUM	优先级超过了最大值
2027	KCXP_RC_PRIORITY_ERROR	优先级错误
2028	KCXP_RC_Q_DELETED	队列已删除
2029	KCXP_RC_Q_FULL	队列已满
2030	KCXP_RC_Q_NOT_EMPTY	队列不为空
2031	KCXP_RC_Q_TYPE_ERROR	队列类型错误
2032	KCXP_RC_Q_MGR_NAME_ERROR	队列管理器名错误
2033	KCXP_RC_Q_MGR_NOT_AVAILABLE	队列管理器不可用
2034	KCXP_RC_REPORT_OPTIONS_ERROR	
2035	KCXP_RC_TRIGGER_CONTROL_ERROR	
2036	KCXP_RC_TRIGGER_DEPTH_ERROR	
2037	KCXP_RC_TRIGGER_MSG_PRIORITY_ERROR	
2038	KCXP_RC_INTR_ERROR	
2039	KCXP_RC_AGAIN_ERROR	
2040	KCXP_RC_IO_ERROR	
2041	KCXP_RC_INVAL_ERROR	
2042	KCXP_RC_FAULT_ERROR	
2043	KCXP_RC_SOCKET_ERROR	SOCKET 错误
2044	KCXP_RC_CONNECT_ERROR	连接错误
2045	KCXP_RC_NO_USER_ERROR	没有这个用户
2046	KCXP_RC_PASSWORD_ERROR	口令错误
2047	KCXP_RC_LB_ERROR	LB 错误
2048	KCXP_RC_NODE_TOO_BUSY	结点太忙
2049	KCXP_RC_CHANNEL_NAME_ERROR	通道名错误
2050	KCXP_RC_NODE_CODE_ERROR	结点代码错误
2051	KCXP_RC_NOT_ROUTE_ERROR	没有路由
2052	KCXP_RC_QM_STOP	队列管理器停止运行了
2063	KCXP_RC_QM_PAUSE	队列管理器暂停
2064	KCXP_RC_QM_DISABLED	队列管理器不可用
2053	KCXP_RC_Q_NAME_ERROR	队列名错误
2054	KCXP_RC_CONNECTION_BROKEN	连接中断
2055	KCXP_RC_CONNECTION_STOPPING	连接停止了
2056	KCXP_RC_SELECTOR_COUNT_ERROR	
2057	KCXP_RC_SELECTOR_ERROR	
2058	KCXP_RC_SELECTOR_LIMIT_EXCEEDED	
2059	KCXP_RC_CHAR_ATTRS_TOO_SHORT	
2060	KCXP_RC_INT_ATTR_COUNT_TOO_SMALL	
2061	KCXP_RC_SELECTOR_NOT_FOR_TYPE	

2062	KCXP_RC_OPTIONS_ERROR	选项错误
2065	KCXP_RC_OPER_OBJECT_ERROR	打开 OBJ 失败
2066	KCXP_RC_GMO_ERROR	GMO 错误
2067	KCXP_RC_READ_CONF_Q_MGR_NAME_ERROR	读配置中的队列管理器名错误
2068	KCXP_RC_READ_CONF_Q_MGR_IP_ERROR	读配置中的队列管器 IP 错误
2069	KCXP_RC_READ_CONF_Q_MGR_USP_ERROR	读配置中的队列管理器 USP 错误
2070	KCXP_RC_Q_MGR_NAME_DEFINE_ERROR	队列管理器名字定义错误
2071	KCXP_RC_Q_MGR_NAME_MATCH_ERROR	队列管理器匹配错误
2072	KCXP_RC_READ_CONF_Q_MGR_USER_NAME_ERROR	读配置中的队列管理器用户名错误
2073	KCXP_RC_READ_SHADOW_ERROR	
2074	KCXP_RC_Q_MGR_EXIST	队列管理器已存在
2075	KCXP_RC_Q_MGR_UN_CREATE	队列管理器没建立
2076	KCXP_RC_PARAMTER_ERROR	参数错误
2078	KCXP_RC_OPEN_USER_INFO_FILE_ERROR	
2079	KCXP_RC_READ_USER_INFO_FILE_ERROR	
2080	KCXP_RC_OPEN_PROC_LOADAVG_ERROR	
2081	KCXP_RC_OPEN_PROC_MEMINFO_ERROR	
2082	KCXP_RC_DATA_ERROR	数据错误
2083	KCXP_RC_GET_FLOW_ERROR	
2084	KCXP_RC_ARRIVE_VAL_DEPTH	
2085	KCXP_RC_OPEN_FILE_ERROR	
2086	KCXP_RC_EVENT_LIST_FULL	事件列表满
2087	KCXP_RC_NOT_FOUND_EVENT_DEFINE	没有找到事件定义
2088	KCXP_RC_ENCRYPT_ERROR	加密错误
2089	KCXP_RC_DECRYPT_ERROR	解密错误
2090	KCXP_RC_COMPRESS_ERROR	压缩错误
2091	KCXP_RC_UNCOMPRESS_ERROR	解压错误
2092	KCXP_RC_READ_DATA_TIMEOUT_ERROR	读数据超时
2093	KCXP_RC_NO_WAIT_TIME	
2094	KCXP_RC_LOADBALANCE_FAIL	
2095	KCXP_RC_MALLOC_ERROR	分配内存错误
2096	KCXP_RC_DLOPEN_COMPRESS_SO_ERROR	
2097	KCXP_RC_DLSYM_COMPRESS_SO_ERROR	
2098	KCXP_RC_DLOPEN_CRYPT_SO_ERROR	
2099	KCXP_RC_DLSYM_CRYPT_SO_ERROR	
2100	KCXP_RC_OPEN_PLUGIN_FILE_ERROR	打开插件文件错误
2101	KCXP_RC_READ_PLUGIN_FILE_ERROR	读插件文件错误
2102	KCXP_RC_INVALID_PARM_PLUGIN_FILE	不合法的插件文件参数
2103	KCXP_RC_PLUGIN_NOT_FOUND	插件没找到
2104	KCXP_RC_PLUGIN_DLOPEN_ERROR	插件 dlopen 错误
2105	KCXP_RC_PLUGIN_DLSYM_ERROR	插件 dlsym 错误
2106	KCXP_RC_PLUGIN_CALL_ERROR	插件调用错误

2107	KCXP_RC_MALLOC_HCONN_SLOT_ERROR	分配 hconn 槽错误
2108	KCXP_RC_APP_PROGRAM_ABEND	
2109	KCXP_RC_OPEN_TOO_MANY_OBJECT_ERROR	打开太多的 obj
2110	KCXP_RC_MT_NOT_TIME_INFINITE	
2111	KCXP_RC_READ_CONF_Q_MGR_PROTOCOL_ERROR	读配置队列管理器协议错误
2112	KCXP_RC_MSG_LIFETIME_OVER	消息生命周期已到
2200	KCXP_RC_EXIT_HANDLE_FAIL	出口句柄失败
2290	KCXP_RC_ROUTE_NOT_EXIST	路由不存在
2401	KCXP_RC_OVER_FLOW1	超过流量控制阀值，消息被废弃
2402	KCXP_RC_OVER_FLOW2	超过流量控制阀值,但消息仍被处理
2403	KCXP_RC_OVER_FLOW3	超过流量控制阀值，需重置连接
2300	KCXP_RC_SSL_NEGOCIATE_FAIL	
2400	KCXP_RC_ARRIVED_MAXCONN	到达最大连接
2411	KCXP_RC_CLIENT_NAME_NOT_ALLOWED	不允许客户端名字
2412	KCXP_RC_QUEUE_NOT_ALLOWED_OPEN	队列不允许打开
2413	KCXP_RC_CLIENT_MAC_NOT_ALLOWED	不允许客户端 MAC 码
2420	KCXP_RC_API_CANNOT_SESSION	
2421	KCXP_RC_DATA_CHECK_IN_ERROR	//api 端验证码失败
2422	KCXP_RC_DATA_CHECK_OUT_ERROR	//数据（被篡改）验证通不过
4000	KCXP_RC_QD_NOT_FOUND	
4001	KCXP_RC_QU_MALLOC_ERROR	
4002	KCXP_RC_NO_WRITE_PERM	
4003	KCXP_RC_REACH_MAX_DEPTH	到达最大队列深度
4004	KCXP_RC_REACH_VAL_DEPTH	到达可用深度
4005	KCXP_RC_MSG_TOO_LARGE	消息太大了
4006	KCXP_RC_QUEUE_EMPTY	队列为空
4007	KCXP_RC_NO_READ_PERM	没有读权限
4008	KCXP_RC_QUEUE_TYPE_ERROR	队列类型错误
4009	KCXP_RC_QUEUE_SHARE_ERROR	队列共享错误
4010	KCXP_RC_QUEUE_PERM_ERROR	队列权限错误
4011	KCXP_RC_CREATE_QUEUE_ERROR	建立队列错误
4012	KCXP_RC_SET_ATTR_NOT_PERM	SET 属性没有权限
4013	KCXP_RC_ATTR_NOT_FOUND	属性没有找到
4014	KCXP_RC_INVALID_MODE	不合法的方式
4015	KCXP_RC_REMOVE_QS_ERROR	
4016	KCXP_RC_EMPTY_Q_ERROR	
4017	KCXP_RC_MSG_NOT_FOUND	消息没有找到
4018	KCXP_RC_TRIGGER_FIRST_MSG	
4019	KCXP_RC_TRIGGER_DEPTH_MSG	
4020	KCXP_RC_TRIGGER_EACH_MSG	
5000	KCXP_RC_CHANNEL_OPERATION_TOBEGIN	
5001	KCXP_RC_CHANNEL_OPERATION_NO_TOEND	

5002	KCXP_RC_CHANNEL_OPERATION_TOEND	
5003	KCXP_RC_CHANNEL_EXIST	
5004	KCXP_RC_CHANNEL_PATH_EXIST	
5005	KCXP_RC_CHANNEL_NOT_EXIST	
5006	KCXP_RC_CHANNEL_INVALID_CHANNEL	
5007	KCXP_RC_CHANNEL_INVALID_PATH	
5008	KCXP_RC_CHANNEL_STATUS_STOP	
6000	KCXP_RC_REMOTEDEF_EXIST	
6001	KCXP_RC_REMOTEDEF_NO_EXIST	
7000	KCXP_RC_PLUGIN_OPEN	KCXP 插件打开错误
7001	KCXP_RC_PLUGIN_GET	KCXP 插件 GET 错误
7002	KCXP_RC_PLUGIN_CALL	KCXP 插件调用错误
7003	KCXP_RC_PLUGIN_NOT_FIND	KCXP 插件没有找到
7010	KCXP_RC_MESSAGE_PLUGIN	
9999	KCXP_RC_SYSTEM_ERROR	系统错误

9.5 Winsock 错误代码

本节按错误编号列出了所有 Winsock 错误代码。但要注意的是，该列表没有包括标记为“BSD 特有”的 Winsock 错误，也没有包括那些尚未正式列入规范的错误。此外，与 Win32 错误有着直接对应关系的 Winsock 错误列在本节末尾。

10004—WSAEINTR

函数调用中断。该错误表明由于对 WSACancelBlockingCall 的调用，造成了一次调用被强行中断。

10009—WSAEBADF

文件句柄错误。该错误表明提供的文件句柄无效。在 MicrosoftWindowsCE 下，socket 函数可能返回这个错误，表明共享串口处于“忙”状态。

10013—WSAEACCES

权限被拒。尝试对套接字进行操作，但被禁止。若试图在 sendto 或 WSASendTo 中使用一个广播地址，但是尚未用 setsockopt 和 SO_BROADCAST 这两个选项设置广播权限，便会产生这类错误。

10014—WSAEFAULT

地址无效。传给 Winsock 函数的指针地址无效。若指定的缓冲区太小，也会产生这个错误。

10022—WSAEINVAL

参数无效。指定了一个无效参数。例如，假如为 WSAIoctl 调用指定了一个无效控制代码，便会产生这个错误。另外，它也可能表明套接字当前的状态有错，例如在一个目前没有监听的套接字上调用 accept 或 WSAAccept。

10024—WSAEMFILE

打开文件过多。提示打开的套接字太多了。通常，Microsoft 提供者只受到系统内可用资源数量的限制。

10035—WSAEWOULDBLOCK
资源暂时不可用。对非锁定套接字来说，如果请求操作不能立即执行的话，通常会返回这个错误。比如说，在一个非暂停套接字上调用 connect，就会返回这个错误。因为连接请求不能立即执行。
10036—WSAEINPROGRESS
操作正在进行中。当前正在执行非锁定操作。一般来说不会出现这个错误，除非正在开发 16 位 Winsock 应用程序。
10037—WSAEALREADY
操作已完成。一般来说，在非锁定套接字上尝试已处于进程中的操作时，会产生这个错误。比如，在一个已处于连接进程的非锁定套接字上，再一次调用 connect 或 WSAConnect。另外，服务提供者处于执行回调函数（针对支持回调例程的 Winsock 函数）的进程中时，也会出现这个错误。
10038—WSAENOTSOCK
无效套接字上的套接字操作。任何一个把 SOCKET 句柄当作参数的 Winsock 函数都会返回这个错误。它表明提供的套接字句柄无效。
10039—WSAEDESTADDRREQ
需要目标地址。这个错误表明没有提供具体地址。比方说，假如在调用 sendto 时，将目标地址设为 INADDR_ANY（任意地址），便会返回这个错误。
10040—WSAEMSGSIZE
消息过长。这个错误的含义很多。如果在一个数据报套接字上发送一条消息，这条消息对内部缓冲区而言太大的话，就会产生这个错误。再比如，由于网络本身的限制，使一条消息过长，也会产生这个错误。最后，如果收到数据报之后，缓冲区太小，不能接收消息时，也会产生这个错误。
10041—WSAEPROTOTYPE
套接字协议类型有误。在 socket 或 WSASocket 调用中指定的协议不支持指定的套接字类型。比如，要求建立 SOCK_STREAM 类型的一个 IP 套接字，同时指定协议为 IPPROTO_UDP，便会产生这样的错误。
10042—WSAENOPROTOOPT
协议选项错误。表明在 getsockopt 或 setsockopt 调用中，指定的套接字选项或级别不明、未获支持或者无效。
10043——WSAEPROTONOSUPPORT
不支持的协议。系统中没有安装请求的协议或没有相应的实施方案。比如，如果系统中没有安装 TCP/IP，而试着建立 TCP 或 UDP 套接字时，就会产生这个错误。10044—WSAESOCKTNOSUPPORT 不支持的套接字类型。对指定的地址家族来说，没有相应的具体套接字类型支持。比如，在向一个不支持原始套接字的协议请求建立一个 SOCK_RAW 套接字类型时，就会产生这个错误。
10045—WSAEOPNOTSUPP
不支持的操作。表明针对指定的对象，试图采取的操作未获支持。通常，如果试着在一个不支持调用 Winsock 函数的套接字上调用了 Winsock 时，就会产生这个错误。比如，在一个数据报套接字上调用 accept 或 WSAAccept 函数时，就会产生这样的错误。
10046—WSAEPFNOSUPPORT
不支持的协议家族。请求的协议家族不存在，或系统内尚未安装。多数情况下，这个错误可与 WSAEAFNOSUPPORT 互换（两者等价）；后者出现得更为频繁。
10047—WSAEAFNOSUPPORT

地址家族不支持请求的操作。对套接字类型不支持的操作来说，在试着执行它时，就会出现这个错误。比如，在类型为 SOCK_STREAM 的一个套接字上调用 sendto 或 WSASendTo 函数时，就会产生这个错误。另外，在调用 socket 或 WSASocket 函数的时候，若同时请求了一个无效的地址家族、套接字类型及协议组合，也会产生这个错误。

10048—WSAEADDRINUSE

地址正在使用。正常情况下，每个套接字只允许使用一个套接字地址（例如，一个 IP 套接字地址由本地 IP 地址及端口号组成）。这个错误一般和 bind、connect 和 WSAConnect 这三个函数有关。可在 setsockopt 函数中设置套接字选项 SO_REUSEADDR，允许多个套接字访问同一个本地 IP 地址及端口号（详情见第 9 章）。

10049—WSAEADDRNOTAVAIL

不能分配请求的地址。API 调用中指定的地址对那个函数来说无效时，就会产生这样的错误。例如，若在 bind 调用中指定一个 IP 地址，但却没有对应的本地 IP 接口，便会产生这样的错误。另外，通过 connect、WSAConnect、sendto、WSASendTo 和 WSAJoinLeaf 这四个函数为准备连接的远程计算机指定端口 0 时，也会产生这样的错误。

10050—WSAENETDOWN

网络断开。试图采取一项操作时，却发现网络连接中断。这可能是由于网络堆栈的错误，网络接口的故障，或者本地网络的问题造成的。

10051—WSAENETUNREACH

网络不可抵达。试图采取一项操作时，却发现目标网络不可抵达（不可访问）。这意味着本地主机不知道如何抵达一个远程主机。换言之，目前没有已知的路由可抵达那个目标主机。

10052—WSAENETRESET

网络重设时断开了连接。由于“保持活动”操作检测到一个错误，造成网络连接的中断。若在一个已经无效的连接之上，通过 setsockopt 函数设置 SO_KEEPALIVE 选项，也会出现这样的错误。

10053—WSAECONNABORTED

软件造成连接取消。由于软件错误，造成一个已经建立的连接被取消。典型情况下，这意味着连接是由于协议或超时错误而被取消的。

10054—WSAECONNRESET

连接被对方重设。一个已经建立的连接被远程主机强行关闭。若远程主机上的进程异常中止运行（由于内存冲突或硬件故障），或者针对套接字执行了一次强行关闭，便会产生这样的错误。针对强行关闭的情况，可用 SO_LINGER 套接字选项和 setsockopt 来配置一个套接字（欲知详情，请参阅《微软 Windows 网络编程》第 9 章）。

10055—WSAENOBUFS

没有缓冲区空间。由于系统缺少足够的缓冲区空间，请求的操作不能执行。

10056—WSAEISCONN

套接字已经连接。表明在一个已建立连接的套接字上，试图再建立一个连接。要注意的是，数据报和数据流套接字均有可能出现这样的错误。使用数据报套接字时，假如事先已通过 connect 或 WSAConnect 调用，为数据报通信关联了一个端点的地址，那么以后试图再次调用 sendto 或 WSASendTo，便会产生这样的错误。

10057—WSAENOTCONN

套接字尚未连接。若在一个尚未建立连接的“面向连接”套接字上发出数据收发请求，便会产生这样的错误。

10058—WSAESHUTDOWN

套接字关闭后不能发送。表明已通过对 shutdown 的一次调用，部分关闭了套接字，但事后

又请求进行数据的收发操作。要注意的是，这种错误只会在已经关闭的那个数据流动方向上才会发生。举个例子来说，完成数据发送后，若调用 shutdown，那么以后任何数据发送调用都会产生这样的错误。

10060—WSAETIMEDOUT

连接超时。若发出了一个连接请求，但经过规定的时间，远程计算机仍未作出正确的响应（或根本没有任何响应），便会发生这样的错误。要想收到这样的错误，通常需要先在套接字上设置好 SO_SNDTIMEO 和 SO_RCVTIMEO 选项，然后调用 connect 及 WSAConnect 函数。要想了解在套接字上设置 SO_SNDTIMEO 和 SO_RCVTIMEO 选项的详情，可参考《微软 Windows 网络编程》第 9 章。

10061—WSAECONNREFUSED

连接被拒。由于被目标机器拒绝，连接无法建立。这通常是由于在远程机器上，没有任何应用程序可在那个地址之上，为连接提供服务。

10064—WSAEHOSTDOWN

主机关闭。这个错误指出由于目标主机关闭，造成操作失败。然而，应用程序此时更有可能收到的是一条 WSAETIMEDOUT（连接超时）错误，因为对方关机的情况通常是在试图建立一个连接的时候发生的。

10065—WSAEHOSTUNREACH

没有到主机的路由。应用程序试图访问一个不可抵达的主机。该错误类似于 WSAENETUNREACH。

10067—WSAEPROCLIM

进程过多。有些 Winsock 服务提供者对能够同时访问它们的进程数量进行了限制。

10091—WSASYSNOTREADY

网络子系统不可用。调用 WSAStartup 时，若提供者不能正常工作（由于提供服务的基层系统不可用），便会返回这种错误。

10092—WSAVERNOTSUPPORTED

Winsock.dll 版本有误。表明不支持请求的 Winsock 提供者版本。

10093—WSANOTINITIALISED

Winsock 尚未初始化。尚未成功完成对 WSAStartup 的一次调用。

10101—WSAEDISCON

正在从容关闭。这个错误是由 WSARecv 和 WSARecvFrom 返回的，指出远程主机已初始化了一次从容关闭操作。该错误是在像 ATM 这样的“面向消息”协议上发生的。

10102—WSAENOMORE

找不到更多的记录。这个错误自 WSALookupServiceNext 函数返回，指出已经没有留下更多的记录。这个错误通常可与 WSA_E_NO_MORE 互换使用。在应用程序中，应同时检查这个错误以及 WSA_E_NO_MORE。

10103—WSAECANCELLED

操作被取消。这个错误指出当 WSALookupServiceNext 调用仍在处理期间，发出了对 WSALookupServiceEnd（服务中止）的一个调用。此时，WSALookupServiceNext 便会返回这个错误。这个错误代码可与 WSA_E_CANCELLED 互换使用。作为应用程序，应同时检查这个错误以及 WSA_E_CANCELLED。

10104—WSAEINVALIDPROCTABLE

进程调用表无效。该错误通常是在进程表包含了无效条目的情况下，由一个服务提供者返回的。欲知服务提供者的详情，可参考第 14 章。

10105—WSAEINVALIDPROVIDER

无效的服务提供者。这个错误同服务提供者关联在一起，在提供者不能建立正确的 Winsock 版本，从而无法正常工作的前提下产生。

10106—WSAEPROVIDERFAILEDINIT

提供者初始化失败。这个错误同服务提供者关联在一起，通常见于提供者不能载入需要的 DLL 时。

10107—WSASYSCALLFAILURE

系统调用失败。表明绝对不应失败的一个系统调用却令人遗憾地失败了。

10108—WSASERVICE_NOT_FOUND

找不到这样的服务。这个错误通常与注册和名字解析函数相关，在查询服务时产生（《微软 Windows 网络编程》第 10 章对这些函数进行了详尽解释）。该错误表明，在给定的名字空间内，找不到请求的服务。

10109—WSATYPE_NOT_FOUND

找不到类的类型。该错误也与注册及名字解析函数关联在一起，在处理服务类（ServiceClass）时发生。若注册好一个服务的实例，它必须引用一个以前通过 WSAInstallServiceClass 安装好的服务。

10110—WSA_E_NO_MORE

找不到更多的记录。这个错误是自 WSALookupServiceNext 调用返回的，指出已经没有剩下的记录。该错误通常可与 WSAENOMORE 互换使用。作为一个应用程序，应同时检查这个错误以及 WSAENOMORE。

10111—WSA_E_CANCELLED

操作被取消。该错误指出在对 WSALookupServiceNext 的调用尚未完成的时候，又发出了对 WSALookupServiceEnd（中止服务）的一个调用。这样，WSALookupServiceNext 就会返回该错误。这个错误代码可与 WSAECANCELLED 互换使用。作为一个应用程序，应同时检查这个错误以及 WSAECANCELLED。

10112—WSAEREFUSED

查询被拒。由于被主动拒绝，所以一个数据库查询操作失败。

11001—WSAHOST_NOT_FOUND

主机没有找到。这个错误是在调用 gethostbyname 和 gethostbyaddr 时产生的，表明没有找到一个授权应答主机（AuthoritativeAnswerHost）。

11002—WSATRY_AGAIN

非授权主机没有找到。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，表明没有找到一个非授权主机，或者遇到了服务器故障。

11003—WSANO_RECOVERY

遇到一个不可恢复的错误。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，指出遇到一个不可恢复的错误，应再次尝试操作。

11004—WSANO_DATA

没有找到请求类型的数据记录。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，指出尽管提供的名字有效，但却没有找到与请求类型对应的数据记录。

11005—WSA_QOS_RECEIVERS

至少有一条预约消息抵达。这个值同 IP 服务质量（QoS）有着密切的关系，其实并不是一个真正的“错误”（QoS 的详情见《微软 Windows 网络编程》第 12 章）。它指出网络上至少有一个进程希望接收 QoS 通信。

11006—WSA_QOS_SENDERS

至少有一条路径消息抵达。这个值同 QoS 关联在一起，其实更像一种状态报告消息。它指

出在网络上，至少有一个进程希望进行 QoS 数据的发送。

11007—WSA_QOS_NO_SENDERS

没有 QoS 发送者。这个值同 QoS 关联在一起，指出不再有任何进程对 QoS 数据的发送有兴趣。请参阅《微软 Windows 网络编程》第 12 章，了解在发生这样的错误时，对所发生情况的一系列完整说明。

11008—WSA_QOS_NO_RECEIVERS

没有 QoS 接收者。这个值同 QoS 关联在一起，指出不再有任何进程对 QoS 数据的接收有兴趣。请参阅《微软 Windows 网络编程》第 12 章，查阅对这个错误的完整说明。

11009—WSA_QOS_REQUEST_CONFIRMED

预约请求已被确认。QoS 应用可事先发出请求，希望在批准了自己对网络带宽的预约请求后，收到通知。若发出了这样的请求，一旦批准，便会收到这样的消息。请参阅《微软 Windows 网络编程》第 12 章，了解对此消息的详细说明。

11010—WSA_QOS_ADMISSION_FAILURE

缺乏资源致错。资源不够，以至于无法满足 QoS 带宽请求。

11011—WSA_QOS_POLICY_FAILURE

证书无效。表明发出 QoS 预约请求的时候，要么用户并不具备正确的权限，要么提供的证书无效。

11012—WSA_QOS_BAD_STYLE

未知或冲突的样式。QoS 应用程序可针对一个指定的会话，建立不同的过滤器样式。若出现这一错误，表明指定的样式类型要么未知，要么存在冲突。请参阅《微软 Windows 网络编程》第 12 章，了解对过滤器样式的详细说明。

11013—WSA_QOS_BAD_OBJECT

无效的 FILTERSPEC 结构或者提供者特有对象。假如为 QoS 对象提供的 FILTERSPEC 结构无效，或者提供者特有的缓冲区无效，便会返回这样的错误，详见《微软 Windows 网络编程》第 12 章。

11014—WSA_QOS_TRAFFIC_CTRL_ERROR

FLOWSPEC 有问题。假如通信控制组件发现指定的 FLOWSPEC 参数存在问题（作为 QoS 对象的一个成员传递），便会返回这样的错误。

11015—WSA_QOS_GENERIC_ERROR

常规 QoS 错误。这是一个比较泛泛的错误；假如其他 QoS 错误都不适合，便返回这个错误。

6—WSA_INVALID_HANDLE

指定的事件对象无效。若使用与 Win32 函数对应的 Winsock 函数，便有可能产生这样的 Win32 错误。它表明传递给 WSAWaitForMultipleEvents 的一个句柄是无效的。

8—WSA_NOT_ENOUGH_MEMORY

内存不够。这个 Win32 错误指出内存数量不足，无法完成指定的操作。

87—WSA_INVALID_PARAMETER

一个或多个参数无效。这个 Win32 错误表明传递到函数内部的参数无效。假若事件计数参数无效，那么在执行 WSAWaitForMultipleEvents 的时候，也会发生这样的错误。

258—WSA_WAIT_TIMEOUT

操作超时。这个 Win32 错误指出重叠 I/O 操作未在规定的时间内完成。

995—WSA_OPERATION_ABORTED

重叠操作被取消。这个 Win32 错误指出由于套接字的关闭，造成一次重叠 I/O 操作的取消。除此以外，该错误也可能在执行 SIO_FLUSH 这个 I/O 控制命令时出现。

996—WSA_IO_INCOMPLETE

重叠 I/O 事件对象未处于传信状态。这个 Win32 错误也和重叠 I/O 操作密切相关，在调用 WSAGetOverlappedResults 函数的时候产生，指出重叠 I/O 操作尚未完成。

997—WSA_IO_PENDING

重叠操作将在以后完成。用 Winsock 函数发出一次重叠 I/O 操作时，若出现这样的 Win32 错误，便表明操作尚未完成，而且会在以后的某个时间完成。有关重叠 I/O 的深入讨论，可参阅《微软 Windows 网络编程》第 8 章。