



机密

Building Application With KCXP

API 编程指南




深圳市金证科技股份有限公司

SHENZHEN KINGDOM TECHNOLOGY CO.,LTD.

<http://www.szkingdom.com>

研制单位	深圳市金证科技股份有限公司	项目编号	
文档名称	《KCXP API》C 语言编程指南	文档编号	
文档状态	初稿	文档版本	V2.0
提 交	田雪	提交日期	2003-3-25
审 批		审批日期	

	深圳市金证科技股份有限公司	部门 #：研发中心 编号 #：DS001 密级 #： 机密
---	----------------------	--

文档信息

项目名称	金证通讯交换平台（ Kingdom Communication eXchange Platform ）		
标题	《KCXP API》编程指南		
类别			
子类别			
摘要			
当前版本	V2.0		
日期	2003.03		
作者	郑一、王鑫、田雪、黄文光		
文档拥有者			
送交人员	徐岷波（ 副总裁 ） 王海航（ 技术总监 ）		
文件	《KCXP API》编程指南.doc: Microsoft Word		
修改历史			
版本号	日期	修改人	摘要
1.0	2002.8	郑一	KCXP 1.0 仅支持 C/C++调用
2.0	2003.03	田雪	增加了 FTP 接口，并编写了 JAVA 版的 KCXP API
2.0	2003.09	田雪	修改了 JAVA 版的 KCXP API

目 录

1 引言.....	1
1.1 编写目的.....	1
1.2 背景.....	1
1.3 定义.....	1
1.4 参考资料.....	1
2 KCXP API 简介	2
3 KCXP 的数据类型.....	2
4 KCXP API 函数调用描述	11
5 KCXP API 的选项参数 - 功能扩展	15
6 KCXP 描述符.....	16
7 KCXP API 的基本语法参考	17
8 KCXP API 调用实例	20
附录.....	64



1 引言

1.1 编写目的

- 目的：
 - 为 KCXP 用户提供详尽的开发和维护资料；
 - 推动公司的项目开发标准化流程的建设；
- 读者：
 - 公司领导（分管技术的副总裁）
 - 公司技术总监
 - 公司各技术部门负责人及技术骨干
 - 总工办
 - 质量保证部
 - 项目组所有成员
 - 编码测试人员
 - 系统测试员
 - KCXP 用户

1.2 背景

- ◆ 软件系统名称：KCXP（金证通讯交换平台）；
- ◆ 项目的任务提出者：徐岷波（副总裁） 王海航（技术总监）；
- ◆ 项目开发者：公司各产品技术部门抽调的技术骨干；
- ◆ 软件系统用户：公司所有产品部门；
- ◆ 软件系统定位：公司级的通讯应用平台；

1.3 定义

1.4 参考资料

- ◆ 《计算机软件工程规范国家标准汇编 2000》
- ◆ 《KCXP 需求说明书》
- ◆ 《KCXP 概要设计说明书》
- ◆ 《KCXP 详细设计说明书》

2 KCXP API(C/C++)简介

KCXP 提供了一个可用 C/C++/JAVA 编程语言调用的应用程序编程接口 (API)。在 API 中包含较少的调用,在本文档中我们将“调用”称为操作 (Operations),因为这样的称谓更适合 KCXP 的队列。每种操作含有一定数目的基本参数,参数一般都是结构体,结构体中的域给出了一些控制操作行为的选项,一般情况下,这些参数可以合理使用缺省值,在定义一个结构体变量的时候,可以初始化变量等于 KCXP_DEFAULT_XXX (XXX 表示不同的结构体其名称不一样)。

用户应用程序跟 KCXP 之间的交互,也就是应用程序和队列发生交互,其实用户的应用程序根本就不需要太多的操作,最常用的是放一条消息到队列和从队列中取走一条消息,虽然 KCXP 提供给用户的 API 函数只有 9 种,但允许用户使用选项 (Options) 和描述符 (Descriptor) 来控制操作行为的细节,同时对选项和描述符也提供了缺省值,使用户在进行一般的操作时无需修改这些参数。

KCXP API 提供如下的功能:

- 建立与本地队列管理器的连接;
- 打开一个对象以供操作;
- 从本地应用队列中取出消息;
- 向队列中放置消息;
- 查询或修改已打开对象的属性;

KCXP API 使用句柄来代表一个正在操作的对象,也就是 API 中的形式参数 Hobj,该句柄对于应用程序而言没有实在的意义,KCXP 用它来区别不同对象的实例,因为用户可以同时打开多个对象进行操作。

3 KCXP(C/C++)的数据类型

◆ KCXP 基本数据类型

在 KCXP API 编程中,所用到的数据类型有:

- 基本数据类型
- 结构体

其中的基本数据类型有:

- ◇ KCXP_BYTE: 无符号字符类型
- ◇ KCXP_BYTEn: 无符号字符数组类型
- ◇ KCXP_CHAR: 字符类型
- ◇ KCXP_CHARn: 字符数组类型
- ◇ KCXP_LONG: 有符号长整型
- ◇ KCXP_INT: 有符号整型
- ◇ KCXP_HCONN: 连接句柄 (它属于有符号长整型)

- ✧ KCXP_HOBJ : 队象句柄 (它属于有符号长整型)
- ✧ KCXP_PTR : void 指针类型

● KCXP_GMO 结构

域名	描述	数据类型	缺省值
strStruId	结构体标识	KCXP_CHAR8	GMObbbb
strVersion	版本号	KCXP_CHAR8	1.0.0.0
Options	选项	KCXP_LONG	0L
WaitInterval	等待时延	KCXP_LONG	(秒为单位)
MatchOptions	匹配选项	KCXP_LONG	0L
GroupStatus	满足组 ID 的消息选项	KCXP_CHAR	‘ ‘
SegmentStatus	满足分段标识选项	KCXP_CHAR	‘ ‘
Segmentation	分段	KCXP_CHAR	‘ ‘
MsgToken	消息标识	KCXP_LONG	0L
ReturnLength	消息返回长度	KCXP_LONG	0L

KCXP_GMO 结构体主要用在 KCXP_Get 操作时控制它具体从队列管理器上读取什么样的消息和怎样读取等行为 , 一般它是作为 KCXP_Get 函数的一个输入 / 输出参数 , 在程序中声明其变量时赋初值为 KCXP_DEFAULT_GMO , 可以根据不同的需要进行修改其某些字段的值 , 那么取得的消息可能也不尽相同。

· strStruId

标识 KCXP_GMO 结构体 , 其值为 “ GMObbbb ” (b 表示一个空格);

特别申明 : 假如有应用程序要使用该字段 , 请务必使用内存操作函数 (例如 memcpy、memcmp 等) 而不要使用字符串操作函数 (例如 strcpy、strcmp 等);

· strVersion

表示 KCXP_GMO 结构体的版本号 , 其值为 “ 1.0.0.0 ” ;

· Options

该字段主要用来控制 KCXP_Get 函数的行为 , 也就是说 , 怎样读取消息 , 它又可以分为以下 3 种选项来进行组合 , 但是每种选项内又会有不同的值 , 它们是互相排斥的 :

· Wait Option :

1. KCXP_GMO_WAIT :
等待方式读取消息 , 其相关联的值有 WaitInterval 字段 , 它表示等待的时延 , 其单位是秒 ;
2. KCXP_GMO_NO_WAIT :
非等待方式读取消息 , 这时的 WaitInterval 字段就不起作用 ;
3. KCXP_GMO_FAIL_IF_QUIESCING :
这个值比较特殊 , 它可以跟其他任何值进行组合 , 因为它是用来

判断队列管理器是否是处于正常状态；

· **Browse Option :**

1. KCXP_GMO_BROWSE_FIRST :
2. KCXP_GMO_BROWSE_NEXT:

目前这两种选项都具有相同的功效，该值说明 KCXP_Get 操作仅仅是想浏览队列中的第一个消息，而并非读取，换句话说，消息浏览之后仍在队列里，不删除掉；

- 3 . KCXP_GMO_BROWSE_MSG_UNDER_CURSOR:

这种选项主要用在用户应用程序基于前一次浏览操作的基础上，在光标的当前位置需要继续往下浏览消息，此时的当前位置可能是队列的某条消息上。它用在消息是按照时间先后顺序进行排序的队列上效果比较好，如果用在消息是按照优先级排序的队列上有可能导致某些消息不能被及时浏览到。

- 4 . CXP_GMO_MSG_GET :

该值是读取消息，队列里就删除掉了。

· **WaitInterval**

该值告诉队列管理器，API 等待的时延；

· **MatchOptions**

该字段主要用来控制在 KCXP_Get 操作中，究竟是读取什么样的消息，同时也就告诉队列管理器，是否是要使用用户填充的消息描述符中的控制信息，该值有如下几种：

- KCXP_MATCH_MSG_ID :
按照消息描述符中定义的消息 ID 来索引；
- KCXP_MATCH_CORREL_ID :
按照消息描述符中定义的相关消息 ID 来索引；
- KCXP_MATCH_GROUP_ID :
按照消息描述符中定义的消息组 ID 来索引；
- KCXP_MATCH_MSG_SEQ_NUMBER :
按照消息描述符中定义的消息逻辑序号来索引；
- KCXP_MATCH_OFFSET :
按照消息的偏移量来进行索引；
- KCXP_MATCH_NONE :
任何选项都不进行匹配；

· **GroupStatus (暂时没用)**

· **SegmentStatus (暂时没用)**

· **Segmentation (暂时没用)**

· **MsgToken (暂时没用)**

· ReturnLength (暂时没用)

● KCXP_MD 结构

域名	描述	数据类型	缺省值
StrStrucId	结构体标识	KCXP_CHAR8	KCXPbbb
StrVersion	版本号	KCXP_CHAR8	1.0.0.0
CbReport	产生报告消息的标志	KCXP_BYTE	0
CbMsgType	消息类型	KCXP_BYTE	数据报消息
LifeTime	消息的生命周期	KCXP_LONG	(毫秒为单位)
InitTime	消息进入队列管理器的时间	KCXP_LONG	0L
IFeedback	反馈码	KCXP_LONG	0L
IEncoding	用户数据加密标记	KCXP_BYTE	不加密
ICodeCharSetId	数据编码字符集	KCXP_LONG	无
IPriority	消息的优先级	KCXP_LONG	队列的优先级
CbPersistence	消息的可靠性	KCXP_BYTE	实时消息
CbFlow	队列流量的控制位	KCXP_BYTE	
StrMsgId	消息的 ID	KCXP_CHAR24	
StrReplyToQ	应答消息队列名	KCXP_CHAR32	
StrReplyToQMgr	应答队列管理器名	KCXP_CHAR32	
StrSrcNodeCode	源节点编号	KCXP_CHAR20	
StrChannelName	通道名	KCXP_CHAR32	
StrDestNodeCode	目的节点编号	KCXP_CHAR20	
StrDestQm	目的队列管理器名	KCXP_CHAR32	
StrDestQ	目的队列	KCXP_CHAR32	
CbRouteType	采用的路由模式	KCXP_BYTE	
CbCompMode	采用的压缩模式	KCXP_BYTE	
CbEncryptMode	采用的加解密模式	KCXP_BYTE	
IPutAppType	放入消息的应用程序类型	KCXP_LONG	
StrPutAppName	放入消息的应用程序名	KCXP_CHAR32	
StrPutDate	消息放入日期	KCXP_CHAR16	
StrPutTime	消息放入时间	KCXP_CHAR16	
StrCorrId	报告消息相关联的消息 ID	KCXP_CHAR24	暂时没有
StrGroupId	消息的组 ID	KCXP_CHAR24	
IMsgSeqNumber	消息的传递序列	KCXP_LONG	
IOffset	消息分段之后的偏移量	KCXP_LONG	
CbMsgFlags	消息标志	KCXP_BYTE	
IOri ginalLength	消息的原始长度	KCXP_LONG	
IDataLength	消息长度	KCXP_LONG	
StrReserved	保留字段	KCXP_CHAR32	

首先必须明确一点的是：在此处介绍的消息描述符 (KCXP_MD) 主要是针对用户的应用程序调用 KCXP_Put (KCXP_Put1) 时所涉及到的一些注意事项，有些字段可能是需要用户的应用程序自己给予赋值，而有些字段则是采用 KCXP 的缺省值，还有一些字段的值是 KCXP 用户服务器根据用户先前的一些操作 (包括 KCXP_Open 等) 来给予赋值。

· **strStruclId**

标识 KCXP_MD 结构体，其值为“KCXPbbb” (b 表示一个空格)；

· **strVersion**

表示 KCXP_MD 结构体的版本号，其值为“1.0.0.0”；

· **cbReport**

该字段主要用来控制用户的 KCXP_Put 操作时，如果消息传送失败，是否需要 KCXP 产生一个报告消息，当其置为 1 时，表示需要产生报告消息，当其置为 0 时，表示不需要产生报告消息。

· **cbMsgType**

消息类型，KCXP 支持的消息类型有：

- ✧ KCXP_MT_DATAGRAM：数据报消息，该消息类型表示数据接收者不需要返回处理结果给发送者；
- ✧ KCXP_MT_REQUEST：请求消息，该消息类型表示数据发送者要求接受者返回处理结果回来；
- ✧ KCXP_MT_REPORT：报告消息，该消息类型表示用户的应用数据发送失败了，而用户又要求产生一个报告消息给它，好用于分析出错的原因；
- ✧ KCXP_MT_RESPONSE：应答消息，该消息类型表示先前的请求的应答结果；

· **iLifeTime**

消息的生命周期，它主要用于 KCXP 在传送消息的过程中，是否能在这个时间段内将其传到目的地，如果在生命周期范围内，消息未能到达目的地，KCXP 自动将其放入死信队列中，并根据需要看是否产生报告消息；

· **iInitTime**

消息进入队列管理器的时间，该时间值是用户调用 KCXP_Put (或者 KCXP_Put1) 时由用户服务器赋值给它的。

· **iFeedback**

反馈码，该值对于用户来说可能派不上用场，它最多用在 API 内部与用户服务器之间传递处理信息用，所以，大多时候用户不用去理

会该变量的值。对于用户来说，KCXP 传递消息的情况是通过 API 函数的参数返回值来传递的。

· iEncoding

用户数据加密标记，因为 KCXP 自身具有加密功能，可能有些用户需要 KCXP 将其数据进行加密，那么就需要将该字段置为宏定义 KCXP_EM_ENCRYPT，在缺省状态下，用户数据是不需要加密的。

· iCodeCharSetId

数据编码字符集，该字段暂时没用上；

· iPriority

消息的优先级，对于 KCXP 的队列来说，进入队列的消息可以有两种选择方式：一是先来后到的方式，采取这种方式的队列，其中的消息排列顺序一定是按照时间的先后顺序排列的，从而就没有理会每个消息的优先级；二是按照优先级的先来后到的方式，意思就是说，优先级高的消息尽管到达的时间稍晚，但是用户规定的优先级很高，这样该消息也有可能被放置在队列中某些消息的前面，从而也就有可能被先处理；

KCXP 能支持的消息优先级分为 10 级，0 到 9 的优先级别是依次递增的顺序，默认状态下，消息的优先级是 5；

· cbPersistence

消息的可靠性，从消息传输的可靠性来看，KCXP 的消息可以分为实时消息和可靠消息，所谓实时消息，就是要求在消息的生命周期范围内送达目的地的消息，而可靠消息，则要求消息无论是在什么样的网络环境和通信线路的质量都要把消息送达目的地。
也可以指定在一定时间范围内消息可靠。

· cbFlow

队列流量控制位，该控制位是在用户往本地队列或远端队列里存放消息的时候，当队列里的消息数达到队列定义的阈值的时候，这时，KCXP 自动产生一个报告消息，告诉消息的发送，自己已经到达队列阈值，必须考虑是否采取行动（要么就不再放消息，要么就稍慢点放）。

· strMsgId

消息的 ID，具体标示某一个消息的身份，在整个 KCXP 的应用中，它都保持唯一性。

· strReplyToQ

应答消息队列名，也就是说，当发送一条请求消息出去之后，可能

需要接收者返回处理结果，这个处理结果回来之后，应该存放的队列名是什么，这是消息接受者在回送处理结果时，必须赋值的字段之一。

· **strReplyToQMgr**

应答消息队列管理器名，也就是说，应答消息回来之后，它应该回到哪一个队列管理器上。

· **strSrcNodeCode**

源节点编号，消息发送的源节点是哪一个，这是每一条消息发送出去都必须有值的字段。

· **strChannelName**

通道名，当用户应用程序选择用**通道**来传递消息的时候，必须调用 KCXP_Open 操作打开该通道，那么以后的 KCXP_Put (或 KCXP_Put1) 操作时 KCXP 就自动为每个通过该通道的消息都要求赋上通道名，以供后面 MCA 传输消息用。

· **strDestNodeCode**

目标节点编号，当用户的应用程序选择用**动态路由**的方式来传递消息时，那么需调用 KCXP_Open 操作来打开，当其调用 KCXP_Put (KCXP_Put1) 时就自动为该字段赋上目标节点编号，以供后面 MCA 传输消息用。

· **strDestQm**

目的队列管理器名，用户应用程序在传递消息的时候，需要指明消息要求传递的目标队列管理器是那一个。

· **strDestQ**

目的队列名，当用户的应用程序调用 KCXP_Put (或 KCXP_Put1) 时，需要指明消息送达目的地之后，应该存放在哪一个本地队列上。

· **cbRouteType**

采用的路由模式，当用户的应用程序调用 KCXP_Open 操作时，KCXP 的用户服务器就已经知道了用户在稍侯的 KCXP_Put(或 KCXP_Put1) 操作时，应该在消息的这个字段赋上什么样的值，这和前面的 strChannel Name、strDestNodeCode 字段的值是相互关联的。

· **cbCompMode**

采用的压缩模式，当用户的应用程序在传递消息的时候，如果希望

KCXP 采用压缩的模式传输消息，那么该字段就应该赋上宏 KCXP_CM_COMPRESS 的值，而在缺省状态之下，是不需要压缩的。

· **cbEncryptMode**

采用的加解密模式，具体就是需要指明数据端是否需要加密，在缺省状态之下，是不需要加密的。

· **iPutAppType**

放入消息的应用程序类型，也就是用户的应用程序是在什么样的平台下编写的（windows、unix、linux、AIX 等），目前可以不予理会。

· **strPutAppName**

放入消息的应用程序名，该字段主要用在当消息传送失败时查询错误信息比较可靠一些。

· **strPutDate**

消息放入的日期，该字段的值是用户服务器自动赋值。

· **strPutTime**

消息放入的时间，该字段的值也是用户服务器自动赋值。

· **strGroupId**

消息的组 ID，当用户的应用程序欲将多个消息组合成一组消息来进行传递，具体传输的时候可能还是一个一个独立的消息体，但是每个消息的组 ID 是一致的，而每个消息的消息 ID 可能是不一样的，那么用户的应用程序在索引消息的时候就可以按照消息的组 ID 进行，并且用户还可以单独索引这一组消息中的某一个消息。

· **iMsgSeqNumber**

消息传递的序列号，该字段有两个方面的用途：1、如果用户传送的消息体大于 4K，这时，假如用户 KCXP_PMO 的操作选项中指明允许 KCXP 将其进行分段，那么 KCXP 就自动将消息分成以 4K 为单位的消息，这时每个消息的 iMsgSeqNumber 字段就表示了分段后的逻辑序号，它是到达目标节点之后消息重新组合的依据之一；2、如果用户将多个组合成一个大的逻辑消息（意思就是说，每个消息在传输的时候，还是一个一个传递），但是用户应用程序为每个消息编一个序号并存放在该字段中，当在消息的接受端如果要从这一组消息中提取其中的某一个消息时，就可以通过消息组 ID 和序列号来进行索引，从而找出需要的消息。

如果消息不分段或不分组，该字段的值为 0L；

· **iOffset**

消息分段之后的偏移量，当一个用户消息被 KCXP 分段之后，该字段就是用来记载每个消息片在原消息中的偏移量，这样便于重新组合。如果消息不分段或不分组，该字段的值为 0L；

· **cbMsgFlags**

消息标志，对于用户的应用程序而言，可以使用的值是指明是否让 KCXP 自动为其进行消息分段，如果允许，则将其赋值为宏 KCXP_MF_SEGMENTATION_ALLOWED，在将消息分段之后，为每一个消息片的 cbMsgFlags 字段又重新赋值，分段的最后一个消息的 cbMsgFlags 字段被自动置为 KCXP_MF_LAST_SEGMENT|KCXP_MF_LAST_MSG_IN_GROUP 的值，而其它消息的该字段被置为 KCXP_MF_SEGMENT|KCXP_MF_MSG_IN_GROUP；如果不允许，则将其赋值为宏 KCXP_MF_SEGMENTATION_INHIBITED；

· **iOriginalLength**

消息的原始长度，如果消息没有被分段（这里指的分段是 KCXP 内部的分段），则消息原始长度和消息的实际长度是一样的；否则，消息原始长度字段值大于消息实际长度值。

· **iDataLength**

消息的实际长度，在 KCXP 节点间进行传输的消息的实际长度，它一般不会超过 4K，用户应用程序在调用 KCXP_Put（KCXP_Put1）时，必须规定传输的消息长度。

· **strReserved**

保留字段。

● **KCXP_OD 结构**

域名	描述	数据类型	缺省值
StrStrucId	数据结构标识	KCXP_CHAR8	0Dbbbbbb
StrVersion	版本号	KCXP_CHAR8	1.0.0.0
StrObjectName	对象名字	KCXP_CHAR32	
StrObjectType	对象类型	KCXP_LONG	
StrQmgrName	该对象所在的队列管理器名	KCXP_CHAR32	

· **strStrucId**

标识 KCXP_OD 结构体，其值为“0Dbbbbbb”（b 表示一个空格）；

· **strVersion**

表示 KCXP_OD 结构体的版本号，其值为“1.0.0.0”；

· strObjectName

用户应用程序需要打开的对象名字，这是用户在调用 KCXP_Open 操作的时候必须填写的结构体 KCXP_OD 的字段之一，要向用户服务器说明具体操作的对象是什么。

· strObjectType

用户应用程序操作对象的类型，目前 KCXP 支持的对象有：

- ✧ 如果用户打开的对象是本地应用队列，那么该字段的值是 KCXP_OT_Q；
- ✧ 如果用户打开的对象是远端队列定义，那么该字段的值是 KCXP_OT_Q；
- ✧ 如果用户打开的对象是通道，那么该字段的值是 KCXP_OT_CHANNEL；
- ✧ 如果用户打开的对象是动态路由，那么该字段的值是 KCXP_OT_ROUTE；

· strQmgrName

用户操作对象所属的队列管理器名，这对于用户来说是无关紧要的一个字段。

● KCXP_PMO 结构

域名	描述	数据类型	缺省值
StrStrucId	数据结构标识	KCXP_CHAR8	PM0bbbb
StrVersion	版本号	KCXP_CHAR8	1.0.0.0
Options	操作选项	KCXP_LONG	
TimeOut	超时时长	KCXP_LONG	
Hobj	对象句柄	KCXP_LONG	

· strStrucId

标识 KCXP_PMO 结构体，其值为“PM0bbbb”（b 表示一个空格）；

· strVersion

表示 KCXP_PMO 结构体的版本号，其值为“1.0.0.0”；

· Options

控制 KCXP_Put (KCXP_Put1) 操作的选项，具体详细的控制方式请见后面相关章节的阐述。

· TimeOut

超时时长。

· Hobj

对象句柄。

4 KCXP API (C/C++)函数调用描述

◆ KCXP API 使用的通用参数：

- I. **Hconn**-连接句柄，是一个 4 字节的标志符，它是 KCXP_Conn 的返回值，在之后的其他 API 调用中都需要用它作为一个输入参数；断开连接调用 KCXP_Disconn。对于对象句柄（Hobj）一样具有这些属性。
- II. **Hobj**-对象句柄，是一个 4 字节的标志符，它是 KCXP_Open 的返回值，在之后的其他对象操作中需要它作为输入参数。
- III. **CompCode**-完成码（Completion Code）

=**KCXP_CC_OK** API 调用被成功地执行

=**KCXP_CC_WARNING** API 调用被成功地执行，但有警告

=**KCXP_CC_FAILED** API 调用失败

- IV. **Reason**-原因码（Reason Code）

=**KCXP_RC_XXXXXX** 错误原因代码

=**KCXP_RC_NONE** 没有原因

◆ KCXP_Conn-连接 KCXP 队列管理器

所有的 KCXP 操作都要求用户应用程序与队列管理器相连，这种连接是通过调用 KCXP_Conn 建立的，用户应用程序可以给出要连接的队列管理器的名称。然而用户应用程序可能不太关心是哪一个队列管理器在为自己提供服务，这种情况下，用户的应用程序可以设置队列管理器的名字为空，从而建立一个与缺省的队列管理器的连接。

KCXP_Conn 返回一个连接句柄，在随后的 KCXP API 调用中，这个句柄用来标志程序和哪一个特定的队列管理器相连。

◆ KCXP_Connx-直连 KCXP 队列管理器

该操作与 KCXP_Conn 不同之处就在于它不需要配置文件（kcxpapi.ini 和 kcxpuser.dat），在它的形式参数中需要应用程序带上用户服务器的 ip

地址、端口号、用户名、用户密码。返回值跟 KCXP_Conn 操作一样。

◆ KCXP_Disconn-断开与 KCXP 队列管理器的连接

断开与队列管理器的连接。当用户应用程序已经完成各种操作而不再需要和某个队列管理器的连接，它通过调用 KCXP_Disconn 来断开这种连接，同时队列管理器释放那些用来支持该程序的所有资源，所有那些没有被应用程序显式地释放的资源，将在此时被隐含地释放。

下面例子给出了一个 KCXP_Conn 和 KCXP_Disconn 的典型组合：

```
KCXP_Conn( QMName, &Hconn, &CompCode, &Reason );  
If( ( CompCode==KCXP_CC_OK ) || ( CompCode==KCXP_CC_WARNING ) )  
{  
    .....  
    KCXP_Disconn( &Hconn, &Options, &CompCode, &Reason );  
}
```

◆ KCXP_Open-打开一个 KCXP 对象

用户应用程序为了能操作一个 KCXP 的对象，比如一个队列，应用程序必须先要打开这个对象。这很类似于在读写一个文件时必须先打开文件。对于大多数用户应用程序而言，队列、通道是最重要的 KCXP 资源，当然还有其他类型的对象，比如动态路由、队列管理等。

对象的类型就决定了它可以进行的操作，例如应用程序只能从一个队列中取一条消息而不能从队列管理器上取一条消息。队列和队列管理器都支持查询操作，用户应用程序通过填充对象描述符来指定需要打开的对象。

打开一个对象时，用户应用程序需要定义对该对象要进行的操作，这也类似于文件的操作。这种对操作的定义是为了帮助队列管理器进行必要的锁定和共享控制从而保证对象数据的完整性，这些对操作的定义是通过参数传递给 KCXP API 的。

KCXP_Open 将返回一个对象句柄。这个对象句柄被用在随后的 KCXP API 调用中，标志特定的对象。

KCXP_Open 在缺省状态下的操作的对象类型是队列。

◆ KCXP_Close-关闭对象句柄

关闭一个对象句柄。当用户应用程序不再需要使用 KCXP 的对象时，它将调用 KCXP_Close 来关闭这个对象，队列管理器将释放所有和该对象相关联的资源，正如前面描述的那样，当用户应用程序需要断开与队列管理器的连接时，那些由程序打开的，还没有关闭的对象将被隐含地关闭。

◆ KCXP_Put-放置消息到队列中

用户应用程序放置消息到某一个队列中，除了给出构成消息的用户数据之外，用户应用程序还需要定义一个消息描述符 (KCXP_MD)，消息描述符提供一些信息用来控制 KCXP 对该消息的处理。消息描述符和消息一起被传送，这样，在消息经过的网络上任何地方，消息描述符都可以被用来控制 KCXP

对该消息的处理。例如，一条消息在网络中传递，但在最后发现不能到达目的地。消息描述符中就定义了系统在这种情况下行为，一般是向消息的原始发送者返回一条报告消息。另外，消息描述符还存在一些和消息相关的信息，比如，消息的优先级、消息的类型、是否是可靠消息等属性。

通过使用结构体 KCXP_PMO 中的选项，用户应用程序也可以控制 KCXP_Put 的行为细节，其中的细节内容请详见“KCXP API 的选项参数 - 功能扩展”。

◆ KCXP_Put1-放置一条消息到队列中

KCXP_Put1 操作与 KCXP_Put 操作的不同之处就在于：执行该操作之前不需要去打开对象，也就是不需要对象句柄，而仅仅将对象描述符（KCXP_OD）传递给它就可以了，当该操作执行完毕之后，与打开的对象句柄也及时关闭，所以它通常也用在仅放置一条消息到队列的操作上。

◆ KCXP_Get-从队列中读取/浏览一条消息

用户应用程序调用 KCXP_Get 来检索 KCXP 队列上的消息，消息所包含的用户数据和消息描述符都被返回给用户应用程序，同样，用户应用程序通过使用选项参数来控制 KCXP_Get 的行为细节。例如，当队列上没有消息的时候，调用是立即返回呢，还是等待一条消息的到达，甚至还可以确定等待的时长。

控制着 KCXP_Get 行为的结构体是 KCXP_GMO，其中的细节内容请详见“KCXP API 的选项参数 - 功能扩展”。

◆ KCXP_Inq-查询 KCXP 对象的属性

用户应用程序可以调用 KCXP_Inq 来查询 KCXP 的对象的属性设置情况，它即可以查询队列的属性，也可以查询通道、队列管理等对象的属性，例如，用户应用程序可以查询现在某个队列上有多少条消息在等待处理。KCXP_Inq 还允许一次查询多个属性。由于一些属性是数字型的，另外一些属性是字符串型，所以它包括两个整型数组和一个字符缓冲区，以及他们各自的宽度，第一个整型数组称为选择符（Selectors），数组的成员是代表对象的不同属性的整数值，同时也表明对象的哪些属性需要返回。第二个整型数组用来保存返回的整数属性的值。字符缓冲区用来保存返回的字符型属性值，它是一个字符串指针数组，所有都是按照顺序来进行存取的，意思就是说，在选择符中规定的先后顺序，无论是在整型数组中返回，还是在字符缓冲区中，都是按照这个顺序排列的，那么用户就按照这个顺序来取值。

◆ KCXP_Set-设置 KCXP 对象的属性

KCXP_Set 允许 KCXP 的对象属性值被修改，例如用户应用程序可以修改某个触发队列的到达消息的数目。KCXP_Set 也支持一次修改多个属性。同样，因为对象的属性有些是数字型，有些是字符型，KCXP_Set 的接口也包括两个数组和一个字符指针数组以及它们的长度，这和 KCXP_Inq 相类似，第一个整型数组包含选择符，每一个选择符代表需要修改的一种属性，第二个整型数组包含一些整型属性的新值，字符缓冲区则包含一些字符属性的新值。

5 KCXP API (C/C++)的选项参数 - 功能扩展

在本节我们主要来看看用户应用程序怎样通过定义不同的选项 (Options) 来控制 KCXP API 调用的操作细节。

◆ KCXP_Open 的选项 (Options)

当用户应用程序打开一个对象时，特别是打开一个队列时，选项参数允许程序定义不同种类的操作，其中包括：

- 打开供输入 (放消息)，且队列可被共享 (其他的用户应用程序也可以同时打开它)；
- 打开供输入，但队列被独占使用；
- 打开供浏览；
- 打开供输出 (取消息)；
- 打开供属性查询；
- 打开供属性修改；
- 使用该对象的缺省选项打开该对象

这些选项有些是互斥的，比如在打开一个队列时，不能同时定义“共享”和“独占”两种选项，但可以在打开一个队列时，声明它既可以用于放消息，也可以用于取消息，还可以用于查询它的属性。

◆ KCXP_Close 的选项 (Options)

KCXP_Close 的选项和动态队列有关，动态队列是在用户应用程序执行的过程中被创建的，这种队列可以是临时的。当创建临时队列的应用程序中止的时候，临时动态队列就被自动的删除。也有一种永久动态队列，需要专门的操作才能删除，这种操作依赖于 KCXP API 提供的 KCXP_Close 的选项，一种选项：当队列上不再有消息的时候就删除该队列；另外一种选项：清除队列上的所有消息并删除该队列。

◆ KCXP_Get 的选项 (Options)

KCXP_Get 的选项是由一个结构体组成，这个结构体既包含标志，也包含不同的域，其中一些选项可以被队列管理器修改，基本的选项包括：

- 如果队列中没有消息，就等待一条消息的到达；
- 如果队列中没有消息，就立即返回；
- 浏览队列上的第一条消息；
- 浏览下一条消息；
- 浏览当前光标位置的消息；
- 取走当前浏览光标所指的那条消息；
- 取走队列上的第一条消息；
- ~~锁定消息；(目前暂不支持)~~
- ~~解锁消息；(目前暂不支持)~~
- 当队列管理器正在被关闭时，此次操作失败；

浏览选项 (Browse) 允许用户应用程序扫描队列上的消息但并不取走它们。这样，应用程序就可以搜索队列上某一条满足特定标准的消息，而和浏览选项一起使用的锁定选项允许程序为自己专门保留某条消息，这样，即使有多个程序在“读”同一个队列，只要该程序决定取走该消息，它总能取到该条消息。

接受被截断的消息的选项用于当提供的缓冲空间小于消息的实际长度的情况。

该结构体中还包含一个时间域，这个域给出了 KCXP_Get 等待一条消息到达的时间，该时间的单位是毫秒。

◆ KCXP_Put 的选项 (Options)

KCXP_Put 的选项是由一个结构体组成，这个结构体既包含标志，也包含不同的域，其中一些选项可以被队列管理器修改。

在选项中有这样两个域，本次 KCXP_Put 操作完成之后，这两个域被用来保存由队列管理器返回的实际的队列管理器和目的队列的名称。

6 KCXP(C/C++)描述符

◆ 消息描述符 (Message Descriptor)

消息描述符和消息一道在网络中被传递，消息描述符的第一个组成部分是一个标志位选项集合，这个集合定义了什么样的报告消息和这条消息发生关联。

报告消息 (Report Message) 是对一条消息在网络中的生命周期内发生的各种事件的响应，其中包括如下一些种类的事件：

- 一条消息因为在生命周期内还未送到目的地而被自动删除掉；
- 消息到达了目的队列；
- 消息被程序取走；
- 发生了意外的情况，当消息送达目的队列管理器时，却发现目的队列已经满，这就是一个意外情况；

除了创建报告消息之外，消息描述符也允许用户应用程序控制报告消息中包含的一些数据，例如，用户应用程序可以要求引起报告消息的消息的消息 ID 被用作报告消息的相关联 ID。这样，接收报告消息的程序才知道报告消息是由哪一条消息引起的，如果需要的话，报告消息的消息 ID 也可以使用引起报告消息的原始消息的消息 ID。

消息描述符还标明消息的种类，这些类型可能是系统定义的，例如一条请求消息 (KCXP_MT_REQUEST) 一条应答消息 (KCXP_MT_RESPONSE) 等。消息类型也可以是用户应用程序定义，应用定义的消息类型可以用来区分同一队列上的不同消息。通过浏览，用户应用程序可以检索到特定类型的消息。

消息描述符中还包含消息的优先级，它可以被显式地被定义，也可取自一个队列的缺省的优先级。

消息的生命周期也被保存在消息的描述符中，它可以被设置为无限大，



消息描述符中有一个域标明消息是否是一条可靠消息。如果设置为可靠，队列管理器被重新启动也不会导致可靠消息的丢失，而非可靠消息则将被丢弃。和优先级一样，每个队列管理器也为队列上的消息提供一个缺省的是否可靠的属性。

在消息描述符中还给出了一个应答目的队列的地址，其中包括目的队列名和目的队列管理器名。如果没有显式地定义目的队列管理器，那么发送应答消息的队列管理器应能够确定目的队列管理器名。

◆ 对象描述符 (Object Descriptor)

在 KCXP 中支持动态创建应用队列。

7 KCXP API (C/C++)的基本语法参考

```

/*=====*/
/* KCXP_Close Call -- Close Object.                                     */
/*=====*/

KCXP_VOID KCXP_Close (
    KCXP_HCONN    Hconn,          /* input */
    KCXP_HOBJ     *Hobj,          /* input/output */
    KCXP_LONG     Options,        /* input */
    KCXP_LONG*CompCode,           /* output */
    KCXP_LONG*Reason );           /* output */

/*=====*/
/* KCXP_Conn Call -- Connect Queue Manager.                           */
/*=====*/

KCXP_VOID      KCXP_Conn (
    KCXP_CHAR    *QMgrName,       /* input */
    KCXP_HCONN   *Hconn,          /* output */
    KCXP_LONG    Options,         /* input */
    KCXP_LONG*CompCode,           /* output */
    KCXP_LONG*Reason );           /* output */

```



```
KCXP_LONG      *CompCode,      /* output */
KCXP_LONG      *Reason );      /* output */

/*=====*/
/* KCXP_Connx Call -- Connect Queue Manager with ip+port+userInfo. */
/*=====*/

KCXP_VOID      KCXP_Connx(
    KCXP_CHAR    *Ip,            /* input */
    KCXP_INT     Port,           /* input */
    KCXP_CHAR    *uName,         /* input */
    KCXP_CHAR    *uPass,         /* input */
    KCXP_HCONN   *Hconn,         /* output */
    KCXP_LONG    *CompCode,      /* output */
    KCXP_LONG    *Reason );      /* output */

/*=====*/
/* KCXP_Disconn Call -- Disconnect Queue Manager. */
/*=====*/

KCXP_VOID KCXP_Disconn (
    KCXP_HCONN   *Hconn,         /* input */
    KCXP_LONG    Options,        /* input */
    KCXP_LONG    *CompCode,      /* output */
    KCXP_LONG    *Reason );      /* output */

/*=====*/
/* KCXP_Get Call -- Get Message. */
/*=====*/

KCXP_VOID KCXP_Get (
    KCXP_HCONN   Hconn,          /* input */
    KCXP_HOBJ    Hobj,           /* input */
    KCXP_VOID    *pMsgDesc,      /* input/output */
    KCXP_VOID    *pGetMsgOpts,   /* input */
    KCXP_LONG    BuffLen,        /* input */
    KCXP_CHAR    *Buffer,        /* output */
    KCXP_LONG    *DataLen,       /* output */
    KCXP_LONG    *CompCode,      /* output */
    KCXP_LONG    *Reason );      /* output */

/*=====*/
/* KCXP_Inq Call -- Inquire Object Attributes. */
/*=====*/

KCXP_VOID KCXP_Inq (
```



```
KCXP_HCONN Hconn,          /* input */
KCXP_HOBJ Hobj,             /* input */
KCXP_LONG SelectorCount,    /* input */
KCXP_LONG *pSelectors,      /* input */
KCXP_LONG *pIntAttrCount,   /* input */
KCXP_LONG *pIntAttrs,       /* output */
KCXP_LONG *pCharAttrLen,    /* input */
KCXP_CHAR *pCharAttrs[],    /* output */
KCXP_LONG *CompCode,        /* output */
KCXP_LONG *Reason );        /* output */

/*=====*/
/* KCXP_Open Call -- Open Object. */
/*=====*/

KCXP_VOID KCXP_Open (
    KCXP_HCONN Hconn,          /* input */
    KCXP_VOID *ObjDesc,        /* input/output */
    KCXP_LONG Options,         /* input */
    KCXP_LONG *Hobj,           /* output */
    KCXP_LONG *CompCode,       /* output */
    KCXP_LONG *Reason );       /* output */

/*=====*/
/* KCXP_Put Call -- Put Message. */
/*=====*/

KCXP_VOID KCXP_Put (
    KCXP_HCONN Hconn,          /* input */
    KCXP_HOBJ Hobj,            /* input */
    KCXP_VOID *pMsgDesc,       /* input/output */
    KCXP_VOID *PutMsgOpts,     /* input/output */
    KCXP_LONG BuffLen,         /* input */
    KCXP_VOID *Buffer,          /* input =(KCXP_BYTE*BuffLen) */
    KCXP_LONG *CompCode,       /* output */
    KCXP_LONG *Reason );       /* output */

/*=====*/
/* KCXP_Put1 Call -- Put one Message. */
/*=====*/

KCXP_VOID WINAPI KCXP_Put1(
    KCXP_HCONN Hconn,          /* input */
    KCXP_VOID *pObjDesc,       /* input/output */
    KCXP_VOID *pMsgDesc,       /* input/output */
    KCXP_VOID *pPutMsgOpts,    /* input/output */

```




```
KCXP_LONG BuffLen,      /* input */
KCXP_VOID *Buffer,      /* input =(KCXP_BYTE*BuffLen) */
KCXP_LONG*CompCode,     /* output */
KCXP_LONG*Reason);      /* output */

/*=====*/
/* KCXP_Set Call -- Set Object Attributes. */
/*=====*/

KCXP_VOID KCXP_Set (
    KCXP_HCONN  Hconn,          /* input */
    KCXP_HOBJ  Hobj,           /* input */
    KCXP_LONG SelectorCount,    /* input */
    KCXP_LONG *pSelectors,      /* input */
    KCXP_LONG IntAttrCount,     /* input */
    KCXP_LONG *pIntAttrs,       /* input */
    KCXP_LONG CharAttrLen,      /* input */
    KCXP_CHAR *pCharAttrs[],    /* input */
    KCXP_LONG *CompCode,        /* output */
    KCXP_LONG *Reason );        /* output */

/*=====*/
/* KCXP_FTPControl FTP control . */
/*=====*/

KCXP_VOID  KCXP_FTPControl(KCXP_HCONN Hconn,
                           KCXP_HOBJ  Hobj,
                           KCXP_HOBJ  HGetObj, /* FTP Ans queue obj */
                           KCXP_CHAR  *strReplyQ, /* FTP Req queue name */
                           KCXP_CHAR  *strUserID, /* FTP server ID */
                           KCXP_INT   iType,      /* FTP type*/
                           KCXP_INT   iTimeOut,   /* put and get timeout */
                           KCXP_INT   iCommand,   /* FTP command */
                           KCXP_CHAR  *argv,      /* FTP command parm */
                           KCXP_INT   iDataLen,   /* RetBuff of size*/
                           KCXP_CHAR  *RetBuff,   /* ret data */
                           KCXP_INT   *pDataLen,  /* ret data len */
                           KCXP_LONG  *CompCode,
                           KCXP_LONG  *Reason);
```

8 KCXP API (C/C++)调用实例

◆ 连接与断开队列管理器



```
/*=====*/
/* Module Name: */
/* */
/* Function: This program is the connecting KCXP_MQM */
/*           and disconning KCXP_MQM example. */
/* */
/* Usage:   Invoked from command line as */
/* */
/* Where */
/* */
/* Notes: */
/*=====*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "kcxpapi.h"

KCXP_INT main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG   DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_MD     MsgDesc = {KCXP_DEFAULT_MD};

    /*=====*/
    /*test KCXP_Conn Call. */
    /* 如果是不指定队列管理器的名字，则自动连接到缺省的队列管理器上。 */
    /*=====*/

    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
        fflush ( stdout );
        exit ( 1 );
    }
    fprintf( stdout, "i have connected KCXP_MQM, Hconn=[%d].\n", Hconn );
    fflush ( stdout );
    /*=====*/
    /* test KCXP_Disconn Call. */
    /*=====*/
}
```




```
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i have disconnected KCXP_MQM.\n" );
fflush ( stdout );
}
```

◆ 打开与关闭通道

```
/*=====*/
/* Module Name: */
/*
/*
/* Function: This program is the opening channel
/*           and closing channel example.
/*
/*
/* Usage:    Invoked from command line as
/*
/*
/* Where
/*
/* Notes:
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc = { KCXP_DEFAULT_OD};
```



```
/*=====*/
/* test KCXP_Conn Call. */
/*=====*/
KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Open Call. */
/*=====*/
ObjDesc.iObjectType = KCXP_OT_CHANNEL;
strcpy ( ObjDesc.strObjectName, "SZQM2.TO.SZQM1" );
Options = KCXP_OO_FAIL_IF_QUIESCING;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i open the channel Object successfully, Hobj=[%ld].\n", Hobj );
fflush ( stdout );

/*=====*/
/* test KCXP_Close Call. */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i close the channel Object successfully.\n" );
```



```
fflush ( stdout );
```

```
/*=====*/
```

```
/* test KCXP_Disconn Call. */
```

```
/*=====*/
```

```
Options = 0L;
```

```
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
```

```
if ( CompCode != KCXP_CC_OK )
```

```
{
```

```
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
```

```
    fflush ( stdout );
```

```
    exit ( 1 );
```

```
}
```

```
}
```

◆ 打开与关闭队列(本地应用队列和远传队列定义)

```
/*=====*/
```

```
/* Module Name: */
```

```
/* */
```

```
/* Function: This program is the opening queue */
```

```
/* and closing queue example. */
```

```
/* */
```

```
/* Usage: Invoked from command line as */
```

```
/* */
```

```
/* Where */
```

```
/* */
```

```
/* Notes: */
```

```
/*=====*/
```

```
# include <stdio.h>
```

```
# include <stdlib.h>
```

```
# include <sys/errno.h>
```

```
# include <sys/types.h>
```

```
# include <sys/socket.h>
```

```
# include <netinet/in.h>
```

```
# include <netdb.h>
```

```
# include "kcxpapi.h"
```

```
KCXP_INT main ( KCXP_INT argc, KCXP_CHAR *argv[] )
```

```
{
```

```
    KCXP_LONG DataLen=0, Options , CompCode, Reason;
```

```
    KCXP_HCONN Hconn;
```

```
    KCXP_HOBJ Hobj;
```



```
KCXP_OD    ObjDesc = { KCXP_DEFAULT_OD};

/*=====*/
/* test KCXP_Conn Call.                                     */
/*=====*/

KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "TESTQ" );
Options = KCXP_OO_AS_Q_DEF/KCXP_OO_OUTPUT/KCXP_OO_INQUIRE;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i open the queue Object successfully, Hobj=[%d].\n", Hobj );
fflush ( stdout );

Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i close the queue Object successfully.\n" );
fflush ( stdout );

/*=====*/
/* test KCXP_Disconn Call.                                     */
/*=====*/
```

```
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 打开与关闭动态路由

```
/*=====*/
/* Module Name: */
/*
/*
/* Function: This program is the opening route
/*            and closing route example.
/*
/*
/* Usage:    Invoked from command line as
/*
/*
/* Where
/*
/*
/* Notes:
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc = { KCXP_DEFAULT_OD};

    /*=====*/
    /* test KCXP_Conn Call. */
}
```



```
/*=====*/
KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

ObjDesc.iObjectType = KCXP_OT_ROUTE;
strcpy ( ObjDesc.strObjectName, "00000001" );
Options = KCXP_OO_FAIL_IF_QUIESCING;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i open the route Object successfully, Hobj=[%d].\n", Hobj );
fflush ( stdout );

Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "i close the route Object successfully.\n" );
fflush ( stdout );

/*=====*/
/* test KCXP_Disconn Call. */
/*=====*/

Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
```



```
{  
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );  
    fflush ( stdout );  
    exit ( 1 );  
}  
}
```



◆ 从队列中浏览消息

```
/*=====*/
/* Module Name: */
/* */
/* Function: This program is the getting a datagram message */
/* */
/* Usage:    Invoked from command line as */
/* */
/* Where */
/* */
/* Notes: */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
    KCXP_GMO    GetMsgOpts = {KCXP_DEFAULT_GMO};
    KCXP_MD     MsgDesc = {KCXP_DEFAULT_MD};
    KCXP_INT    BuffLen;
    KCXP_CHAR    Buffer[ 1024 ];

    /*=====*/
    /* test KCXP_Conn Call. */
    /*=====*/

    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
        fflush ( stdout );
    }
}
```




```
        exit ( 1 );
    }

    /*=====*/
    /* test KCXP_Open Call.                                */
    /*=====*/

    ObjDesc.iObjectType = KCXP_OT_Q;
    strcpy ( ObjDesc.strObjectName, "TESTQ" );
    Options = KCXP_OO_AS_Q_DEF/KCXP_OO_BROWSE;

    KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
        fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
        fflush ( stdout );
        exit ( 1 );
    }

    /*=====*/
    /* test KCXP_Get Call.                                */
    /* KCXP_GMO_BROWSE_MSG_UNDER_CURSOR : 当前游标位置浏览。 */
    /* KCXP_GMO_BROWSE_FIRST : 浏览队列的第一条消息。        */
    /* KCXP_GMP_BROWSE_NEXT : 浏览队列当前位置的第一条消息。 */
    /*=====*/

    GetMsgOpts.Options =KCXP_GMO_WAIT
                        |KCXP_GMO_FAIL_IF_QUIESCING
                        |KCXP_GMO_MSG_UNDER_CURSOR;

    GetMsgOpts.WaitInterval = 10;
    memset ( Buffer, 0x00, sizeof( Buffer ) );
    while (1)
    {
        KCXP_Get( Hconn
            , Hobj
            , &MsgDesc
            , &GetMsgOpts
            , BuffLen
            , Buffer
            , &DataLen
            , &CompCode
            , &Reason );
    }
```



```
if ( CompCode == KCXP_CC_OK )
{
    fprintf( stdout, "I have recvd a message.[%s].\n", Buffer );
    fflush ( stdout );
}
}

/*=====*/
/* test KCXP_Close Call. */
/*=====*/

Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Disconn Call. */
/*=====*/

Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
}
}
```

◆ 从队列中取一个数据报消息

```
/*=====*/
/* Module Name: */
/* */
/* Function: This program is the getting a datagram message */
/* */
/* Usage:    Invoked from command line as */
/* */
/* Where */
/* */
```



```
/* Notes: */
/*=====*/
# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
    KCXP_GMO    GetMsgOpts = {KCXP_DEFAULT_GMO};
    KCXP_MD     MsgDesc = {KCXP_DEFAULT_MD};
    KCXP_INT    BuffLen;
    KCXP_CHAR   Buffer[ 1024 ];

    /*=====*/
    /* test KCXP_Conn Call. */
    /*=====*/
    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
        fflush ( stdout );
        exit ( 1 );
    }

    /*=====*/
    /* test KCXP_Open Call. */
    /*=====*/
    ObjDesc.iObjectType = KCXP_OT_Q;
    strcpy ( ObjDesc.strObjectName, "TESTQ" );
    Options = KCXP_OO_AS_Q_DEF|KCXP_OO_OUTPUT|KCXP_OO_INQUIRE;

    KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
```



```
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Get Call.                                     */
/*=====*/

GetMsgOpts.Options = KCXP_GMO_WAIT
                    | KCXP_GMO_FAIL_IF QUIESCING
                    | KCXP_GMO_MSG_UNDER_CURSOR;

GetMsgOpts.WaitInterval = 10;
memset ( Buffer, 0x00, sizeof( Buffer ) );

KCXP_Get( Hconn
        , Hobj
        , &MsgDesc
        , &GetMsgOpts
        , BuffLen
        , Buffer
        , &DataLen
        , &CompCode
        , &Reason );

if ( CompCode == KCXP_CC_OK )
{
    fprintf( stdout, "I have recvd a message.[%s].\n", Buffer );
    fflush ( stdout );
}

/*=====*/
/* test KCXP_Close Call.                                     */
/*=====*/

Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
}
```



```
        exit ( 1 );
    }

    /*=====*/
    /* test KCXP_Disconn Call.                                */
    /*=====*/

    Options = 0L;
    KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
        fflush ( stdout );
    }
}
```

◆ 从队列中取一个请求消息

```
/*=====*/
/* Module Name:                                              */
/*                                                         */
/* Function: This program is the getting a request message */
/*                                                         */
/* Usage:   Invoked from command line as                    */
/*                                                         */
/* Where                                          */
/*                                                         */
/* Notes:                                          */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj1, Hobj2;
    KCXP_OD    ObjDesc = { KCXP_DEFAULT_OD};
```



```
KCXP_PMO    PutMsgOpts = { KCXP_DEFAULT_PMO };
KCXP_GMO    GetMsgOpts = { KCXP_DEFAULT_GMO };
KCXP_MD     MsgDesc = { KCXP_DEFAULT_MD };
KCXP_INT    BuffLen;
KCXP_CHAR   Buffer[ 1024 ];

/*=====*/
/* test KCXP_Conn Call.                               */
/*=====*/

KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Open Call.                               */
/*=====*/

ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "TESTQ" );
Options = KCXP_OO_AS_Q_DEF|KCXP_OO_OUTPUT|KCXP_OO_INQUIRE;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj1, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

ObjDesc.iObjectType = KCXP_OT_CHANNEL;
strcpy ( ObjDesc.strObjectName, "szqm2.to.szqm1" );
strcpy ( ObjDesc.strQmgrName, "szqm2" );
Options = KCXP_OO_FAIL_IF_QUIESCING;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj2, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
}
```



```
fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
fflush ( stdout );
exit ( 1 );
}
fprintf( stdout, "The Channel Object Handle=[%d].\n", Hobj2 );
fflush ( stdout );

/*=====*/
/* test KCXP_Get Call. */
/*=====*/
CompCode = KCXP_CC_OK;
while ( CompCode == KCXP_CC_OK )
{
    GetMsgOpts.Options =    KCXP_GMO_WAIT
                          |KCXP_GMO_FAIL_IF QUIESCING
                          |KCXP_GMO_MSG_UNDER_CURSOR;

    GetMsgOpts.WaitInterval = 10;
    memset ( Buffer, 0x00, sizeof( Buffer ) );
    KCXP_Get( Hconn
              , Hobj1
              , &MsgDesc
              , &GetMsgOpts
              , BuffLen
              , Buffer
              , &DataLen
              , &CompCode
              , &Reason );

    if ( CompCode == KCXP_CC_FAILED )
    {
        fprintf( stdout, "There is not message to get.\n" );
        fflush ( stdout );
        break;
    }
    fprintf( stdout, "I have recvd a message.[%s].\n", Buffer );
    fflush ( stdout );

    if ( MsgDesc.cbMsgType == KCXP_MT_REQUEST )
    {
        MsgDesc.cbMsgType = KCXP_MT_RESPONSE;
        MsgDesc.iPriority = 0x7;
        MsgDesc.iLifeTime = 20000L;
    }
}
```



```
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
```

```
strcpy ( MsgDesc.strChannelName, "szqm2.to.szqm1" );
```

```
strcpy ( MsgDesc.strDestQ, "TESTQ" );
```

```
strcpy ( MsgDesc.strDestQm, "TESTMGR" );
```

```
memset ( Buffer, 0x00, sizeof ( Buffer ) );
```

```
sprintf( Buffer, "%s", "3+2-5=0" );
```

```
BuffLen = strlen ( Buffer );
```

```
KCXP_Put( Hconn
```

```
    , Hobj2
```

```
    , &MsgDesc
```

```
    , &PutMsgOpts
```

```
    , BuffLen
```

```
    , Buffer
```

```
    , &CompCode
```

```
    , &Reason );
```

```
if ( CompCode == KCXP_CC_FAILED )
```

```
    fprintf( stdout, "i put response message failed.\n" );
```

```
else
```

```
    fprintf( stdout, "I send response message=[%s]\n", Buffer );
```

```
fflush ( stdout );
```

```
CompCode = KCXP_CC_OK;
```

```
}
```

```
}
```

```
/*=====*/
```

```
/* test KCXP_Close Call. */
```

```
/*=====*/
```

```
Options = KCXP_CO_NONE;
```

```
KCXP_Close( Hconn, &Hobj1, Options, &CompCode, &Reason );
```

```
if ( CompCode != KCXP_CC_OK )
```

```
{
```

```
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
```

```
    fflush ( stdout );
```

```
}
```

```
KCXP_Close( Hconn, &Hobj2, Options, &CompCode, &Reason );
```




```
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
}

/*=====*/
/* test KCXP_Disconn Call.                                */
/*=====*/

Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
}
}
```

◆ 放一个消息到本地应用队列中

```
/*=====*/
/* Module Name:                                            */
/*                                                     */
/* Function: This program is the putting a request message into local-queue */
/*                                                     */
/* Usage:   Invoked from command line as                */
/*                                                     */
/* Where                                         */
/*                                                     */
/* Notes:                                         */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
```



```
KCXP_HCONN Hconn;
KCXP_HOBJ  Hobj;
KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
KCXP_PMO    PutMsgOpts = {KCXP_DEFAULT_PMO};
KCXP_MD     MsgDesc = {KCXP_DEFAULT_MD};
KCXP_INT    BuffLen;
KCXP_CHAR   Buffer[ 1024 ];

/*=====*/
/* test KCXP_Conn Call.                                     */
/*=====*/

KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Open Call.                                     */
/*=====*/

ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "TESTQ" );
Options = KCXP_OO_AS_Q_DEF|KCXP_OO_OUTPUT|KCXP_OO_INQUIRE;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}
fprintf( stdout, "The Client Object Handle=[%d].\n", Hobj );
fflush ( stdout );

/*=====*/
/* test KCXP_Put Call.                                     */
/*=====*/

MsgDesc.cbMsgType = KCXP_MT_DATAGRAM;
MsgDesc.iPriority = 0x7;
```



```
MsgDesc.iLifeTime = 20000L;
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
strcpy ( MsgDesc.strChannelName, "szqm2.to.szqm1" );
strcpy ( MsgDesc.strDestQ, "TESTQ" );

memset ( Buffer, 0x00, sizeof ( Buffer ) );
sprintf( Buffer, "%s", "3+2-5=?" );
BuffLen = strlen ( Buffer );

KCXP_Put( Hconn
    , Hobj
    , &MsgDesc
    , &PutMsgOpts
    , BuffLen
    , Buffer
    , &CompCode
    , &Reason );

if ( CompCode == KCXP_CC_FAILED )
    fprintf( stdout, "put message error.\n" );
else
    fprintf( stdout, "I have send a message to AppServer.[%s]\n", Buffer );
fflush ( stdout );

/*=====*/
/* test KCXP_Close Call.                                     */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Disconn Call.                                     */
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
```



```
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 放一个消息到通道中

```
/*=====*/
/* Module Name:                                     */
/*                                                     */
/* Function: This program is the putting a request message into channel */
/*                                                     */
/* Usage:      Invoked from command line as          */
/*                                                     */
/* Where                                              */
/*                                                     */
/* Notes:                                           */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
    KCXP_PMO   PutMsgOpts = { KCXP_DEFAULT_PMO};
    KCXP_MD    MsgDesc = { KCXP_DEFAULT_MD};
    KCXP_INT   BuffLen;
    KCXP_CHAR  Buffer[ 1024 ];

    /*=====*/
    /* test KCXP_Conn Call.                            */
    /*=====*/
}
```



```
KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Open Call.                                     */
/*=====*/

ObjDesc.iObjectType = KCXP_OT_CHANNEL;
strcpy ( ObjDesc.strObjectName, "szqm2.to.szqm1" );
Options = KCXP_OO_FAIL_IF_QUIESCING;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Put Call.                                     */
/*=====*/

MsgDesc.cbMsgType = KCXP_MT_DATAGRAM;
MsgDesc.iPriority = 0x7;
MsgDesc.iLifeTime = 20000L;
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
strcpy ( MsgDesc.strChannelName, "szqm2.to.szqm1" );

memset ( Buffer, 0x00, sizeof ( Buffer ) );
sprintf( Buffer, "%s", "3+2-5=?" );
BuffLen = strlen ( Buffer );

KCXP_Put( Hconn
        , Hobj
        , &MsgDesc
        , &PutMsgOpts
        , BuffLen
        , Buffer
```

```
, &CompCode
, &Reason );

if ( CompCode == KCXP_CC_FAILED )
    fprintf( stdout, "put message error.\n" );
else
    fprintf( stdout, "I have send a message to AppServer.[%s]\n", Buffer );
fflush ( stdout );

/*=====*/
/* test KCXP_Close Call.                                */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Disconn Call.                                */
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 放置一个消息到远端队列上

```
/*=====*/
/* Module Name:                                           */
/*                                                        */
/*                                                        */
/*                                                        */
/*=====*/
```



```
/* Function: This program is the putting a request message into channel */
/* */
/* Usage:   Invoked from command line as */
/* */
/* Where */
/* */
/* Notes: */
/*=====*/
# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT  main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj;
    KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
    KCXP_PMO   PutMsgOpts = { KCXP_DEFAULT_PMO};
    KCXP_MD    MsgDesc = { KCXP_DEFAULT_MD};
    KCXP_INT   BuffLen;
    KCXP_CHAR  Buffer[ 1024 ];

    /*=====*/
    /* test KCXP_Conn Call. ( 连接缺省的队列管理器 ) */
    /*=====*/
    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
        fflush ( stdout );
        exit ( 1 );
    }

    /*=====*/
    /* test KCXP_Open Call. 对象“RQ.LQ”是远端节点上的一个队列在本节点上的*/
    /* 映象定义，对于应用程序来说，就像操作本地队列一样。 */
    /*=====*/
}
```



```
/*=====*/
ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "RQ.LQ" );
Options = KCXP_OO_FAIL_IF_QUIESCING;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Put Call. */
/*=====*/

MsgDesc.cbMsgType = KCXP_MT_DATAGRAM;
MsgDesc.iPriority = 0x7;
MsgDesc.iLifeTime = 20000L;
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
strcpy ( MsgDesc.strChannelName, "szqm2.to.szqm1" );

memset ( Buffer, 0x00, sizeof ( Buffer ) );
sprintf( Buffer, "%s", "3+2-5=?" );
BuffLen = strlen ( Buffer );

KCXP_Put( Hconn
    , Hobj
    , &MsgDesc
    , &PutMsgOpts
    , BuffLen
    , Buffer
    , &CompCode
    , &Reason );

if ( CompCode == KCXP_CC_FAILED )
    fprintf( stdout, "put message error.\n" );
else
    fprintf( stdout, "I have send a message to AppServer.[%s]\n", Buffer );
fflush ( stdout );

/*=====*/
```



```
/* test KCXP_Close Call. */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/* test KCXP_Disconn Call. */
/*=====*/

Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 通过动态路由方式发送一条消息

```
/*=====*/
/* Module Name: */
/* */
/* Function: This program is the putting a request message into channel */
/* */
/* Usage:    Invoked from command line as */
/* */
/* Where */
/* */
/* Notes: */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
```



```
# include <netinet/in.h>
```

```
# include <netdb.h>
```

```
# include "kcxpapi.h"
```

```
KCXP_INT main ( KCXP_INT argc, KCXP_CHAR *argv[] )
```

```
{
```

```
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
```

```
    KCXP_HCONN Hconn;
```

```
    KCXP_HOBJ  Hobj;
```

```
    KCXP_OD    ObjDesc={ KCXP_DEFAULT_OD};
```

```
    KCXP_PMO   PutMsgOpts = { KCXP_DEFAULT_PMO};
```

```
    KCXP_MD    MsgDesc = { KCXP_DEFAULT_MD};
```

```
    KCXP_INT   BuffLen;
```

```
    KCXP_CHAR  Buffer[ 1024 ];
```

```
    /*=====*/
```

```
    /* test KCXP_Conn Call. */
```

```
    /*=====*/
```

```
    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
```

```
    if ( CompCode != KCXP_CC_OK )
```

```
    {
```

```
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
```

```
        fflush ( stdout );
```

```
        exit ( 1 );
```

```
    }
```

```
    /*=====*/
```

```
    /* test KCXP_Open Call. */
```

```
    /* 对象类型必须置为宏定义 KCXP_OT_ROUTE ; */
```

```
    /* “ 10001 ” 标志目标节点编号。 */
```

```
    /*=====*/
```

```
    ObjDesc.iObjectType = KCXP_OT_ROUTE;
```

```
    strcpy ( ObjDesc.strObjectName, "10001" );
```

```
    Options = KCXP_OO_FAIL_IF_QUIESCING;
```

```
    KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
```

```
    if ( CompCode != KCXP_CC_OK )
```

```
    {
```

```
        fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
```

```
        fflush ( stdout );
```

```
        exit ( 1 );
```

```
    }
```



```
/*=====*/
/* test KCXP_Put Call.                                     */
/*=====*/

MsgDesc.cbMsgType = KCXP_MT_DATAGRAM;
MsgDesc.iPriority = 0x7;
MsgDesc.iLifeTime = 20000L;
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
strcpy ( MsgDesc.strChannelName, "szqm2.to.szqm1" );

memset ( Buffer, 0x00, sizeof ( Buffer ) );
sprintf( Buffer, "%s", "3+2-5=?" );
BuffLen = strlen ( Buffer );

KCXP_Put( Hconn
        , Hobj
        , &MsgDesc
        , &PutMsgOpts
        , BuffLen
        , Buffer
        , &CompCode
        , &Reason );

if ( CompCode == KCXP_CC_FAILED )
    fprintf( stdout, "put message error.\n" );
else
    fprintf( stdout, "I have send a message to AppServer.[%s]\n", Buffer );
fflush ( stdout );

/*=====*/
/* test KCXP_Close Call.                                     */
/*=====*/

Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
```



```
/* test KCXP_Disconn Call. */
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 使用 KCXP_Put1 函数放入一条消息

```
#include <windows.h>
#include <process.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <stdio.h>
#include <stdlib.h>
#include "kcxpapi.h"

int main(int argc, char *argv[])
{
    KCXP_LONG  DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN Hconn;
    KCXP_HOBJ  Hobj ;
    KCXP_OD    ObjDesc = {KCXP_DEFAULT_OD};
    KCXP_PMO    MsgOpts = {KCXP_DEFAULT_PMO};
    KCXP_MD     MsgDesc = {KCXP_DEFAULT_MD};
    KCXP_INT    BuffLen = 1024;
    KCXP_CHAR   Buffer[1024];

    int i , n=0, nSuccess=0;

    /*=====*/
    /* test KCXP_Conn Call. */
    /*=====*/

    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
```



```
{
    fprintf(stdout, "connect error[%ld]. Thread index=%d\n",
Reason, index);
    fflush ( stdout );
    goto end ;
}

ObjDesc.iObjectType = KCXP_OT_Q; // queue type
strcpy ( ObjDesc.strObjectName, "TESTQ"); //put1 所用的队列名
Options = KCXP_00_AS_Q_DEF|KCXP_00_OUTPUT|KCXP_00_INPUT; //队列属性

/*=====*/
/
/* test KCXP_Put and KCXP_Get Call. */
/*=====*/
/

//put MsgDesc
MsgDesc.cbMsgType = KCXP_MT_DATAGRAM;
MsgOpts.Options = KCXP_PMO_NEW_MSG_ID; //生成 MSG ID
MsgDesc.cbMsgFlags = KCXP_MF_SEGMENTATION_ALLOWED;
MsgDesc.cbPersistence = KCXP_NOPERSISTENCE_MESSAGE;
MsgDesc.iLifetime = 20000000L; //毫秒
MsgDesc.iPriority = 0x7;
MsgDesc.iDataLength = BuffLen;
MsgOpts.Timeout = 20; //20 秒

/*=====*/
/* put message 到 TESTQ */
/*=====*/
KCXP_Put1( Hconn
    , &ObjDesc
    , &MsgDesc
    , &MsgOpts
    , BuffLen
    , Buffer
    , &CompCode
    , &Reason );
if ( CompCode == KCXP_CC_FAILED )
{
    fprintf( stdout, "put message error[%ld.\n", Reason );
    fflush ( stdout );
}
```

```
    }  
end:  
    Options = 0L;  
  
    KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );  
  
    if ( CompCode != KCXP_CC_OK )  
    {  
        fprintf(stdout, "di sconnecterror[%ld]. Thread  
index=%d\n", Reason, index );  
        fflush ( stdout );  
        //goto end ;  
    }  
    exit(0);  
}
```

◆ 查询本地应用队列的属性

```
/*=====*/  
/* Module Name:                                     */  
/*                                                     */  
/* Function: This program is the inquiring local-queue attributes */  
/*                                                     */  
/* Usage:      Invoked from command line as          */  
/*                                                     */  
/* Where                                              */  
/*                                                     */  
/* Notes:                                           */  
/*=====*/  
  
# include <stdio.h>  
# include <stdlib.h>  
# include <sys/errno.h>  
# include <sys/types.h>  
# include <sys/socket.h>  
# include <netinet/in.h>  
# include <netdb.h>  
# include "kcxpapi.h"  
  
KCXP_INT   main ( KCXP_INT argc, KCXP_CHAR *argv[] )  
{  
    KCXP_LONG   DataLen=0, Options , CompCode, Reason;  
    KCXP_HCONN Hconn;  
    KCXP_HOBJ   Hobj;
```



```
KCXP_OD      ObjDesc = { KCXP_DEFAULT_OD };
KCXP_PMO      PutMsgOpts = { KCXP_DEFAULT_PMO };
KCXP_GMO      GetMsgOpts = { KCXP_DEFAULT_GMO };
KCXP_MD       MsgDesc = { KCXP_DEFAULT_MD };
KCXP_INT      BuffLen;
KCXP_CHAR     Buffer[ 1024 ];
KCXP_LONG     SelectorCount, Selectors[10];
KCXP_LONG     IntAttrCount, IntAttrs[10], CharAttrLen;
KCXP_CHAR     CharAttrs[100];

/*=====*/
/*(A) test KCXP_Conn Call.                               */
/*=====*/
KCXP_Conn( "", &Hconn, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/*(B) test KCXP_Open Call.                               */
/*=====*/
ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "TESTQ" );
Options = KCXP_OO_AS_Q_DEF|KCXP_OO_INQUIRE;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

SelectorCount = 2;
Selectors[ 0 ] = KCXP_QA_CURDEPTH;
Selectors[ 1 ] = KCXP_QA_INITQNAME;

KCXP_Inq( Hconn
```



```
, Hobj
, SelectorCount
, &Selectors
, &IntAttrCount
, IntAttrs
, NULL
, NULL
, &CompCode
, &Reason );

if ( CompCode == KCXP_CC_FAILED )
{
    fprintf( stdout, "put message error.\n" );
    fflush ( stdout );
}

/*=====*/
/*(C) test KCXP_Close Call.                               */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/*(D) test KCXP_Disconn Call.                               */
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

◆ 设置本地应用队列的属性



```
/*=====*/
/* Module Name: */
/* */
/* Function: This program is the setting local-queue attributes */
/* */
/* Usage:   Invoked from command line as */
/* */
/* Where */
/* */
/* Notes: */
/*=====*/

# include <stdio.h>
# include <stdlib.h>
# include <sys/errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include "kcxpapi.h"

KCXP_INT   main ( KCXP_INT argc, KCXP_CHAR *argv[] )
{
    KCXP_LONG   DataLen=0, Options , CompCode, Reason;
    KCXP_HCONN  Hconn;
    KCXP_HOBJ   Hobj;
    KCXP_OD     ObjDesc = { KCXP_DEFAULT_OD};
    KCXP_PMO    PutMsgOpts = { KCXP_DEFAULT_PMO};
    KCXP_GMO    GetMsgOpts = { KCXP_DEFAULT_GMO};
    KCXP_MD     MsgDesc = { KCXP_DEFAULT_MD};
    KCXP_INT    BuffLen;
    KCXP_CHAR   Buffer[ 1024 ];
    KCXP_LONG   SelectorCount, Selectors[10];
    KCXP_LONG   IntAttrCount, IntAttrs[10], CharAttrLen;
    KCXP_CHAR   CharAttrs[100];

    /*=====*/
    /*(A) test KCXP_Conn Call. */
    /*=====*/

    KCXP_Conn( "", &Hconn, &CompCode, &Reason );
    if ( CompCode != KCXP_CC_OK )
    {
        fprintf ( stdout, "The Client connection error[%ld].\n", Reason );
    }
}
```



```
fflush ( stdout );
exit ( 1 );
}

/*=====*/
/*(B) test KCXP_Open Call. */
/*=====*/
ObjDesc.iObjectType = KCXP_OT_Q;
strcpy ( ObjDesc.strObjectName, "TESTQ" );
Options = KCXP_OO_AS_Q_DEF|KCXP_OO_SET;

KCXP_Open( Hconn, &ObjDesc, Options, &Hobj, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    KCXP_Disconn( &Hconn, 0L, &CompCode, &Reason );
    fprintf( stdout, "The Client open the object error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

SelectorCount = 1L;
Selectors[ 0 ] = KCXP_QA_VALDEPTH;
IntAttr = 1L;
IntAttrs[0] = 5000L;

KCXP_Set( Hconn
    , Hobj
    , SelectorCount
    , &Selectors
    , &IntAttrCount
    , IntAttrs
    , NULL
    , NULL
    , &CompCode
    , &Reason );

if ( CompCode == KCXP_CC_FAILED )
{
    fprintf( stdout, "put message error.\n" );
    fflush ( stdout );
}
```

```
/*=====*/
/*(C) test KCXP_Close Call.                                     */
/*=====*/
Options = KCXP_CO_NONE;
KCXP_Close( Hconn, &Hobj, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client close the object handle error[%ld].\n", Reason );
    fflush ( stdout );
    exit ( 1 );
}

/*=====*/
/*(D) test KCXP_Disconn Call.                                   */
/*=====*/
Options = 0L;
KCXP_Disconn( &Hconn, Options, &CompCode, &Reason );
if ( CompCode != KCXP_CC_OK )
{
    fprintf( stdout, "The Client disconnect the connection error[%ld].\n" );
    fflush ( stdout );
    exit ( 1 );
}
}
```

9 KCXP API (JAVA)简介

Java 版的 API 是从 C/C++版移植而来，供 Java 应用程序调用。它具有 Java 语言本身所固有的一些优点，如可移植性好，完全面向对象，良好的安全性，为 Java 应用程序访问 KCXP 节点提供了强大的支持；缺点是在效率上受到一些影响，没有 C/C++版本的 API 快。下面为 Java API 提供的主要数据类型和接口，参数与 C/C++相似。

■ KCXPAPI

这个类提供了面向应用的所有接口，包括 Conn、Disconn、Open、Close、Put、Get 等接口。

- ◆ void Conn(String QMgrName,
 KCXPPint CompCode,
 KCXPPint Reason);
- ◆ void Connx(String ipAddr,
 int port,
 String name,

```

        String password,
        KCXPPint CompCode,
        KCXPPint Reason);
◆ void Disconn(int Options,
        KCXPPint CompCode,
        KCXPPint Reason);
◆ void Open(KCXPOD pObjDesc,
        int Options,
        KCXPPint Hobj,
        KCXPPint CompCode,
        KCXPPint Reason);

◆ void Close(KCXPPint Hobj,
        int Options,
        KCXPPint CompCode,
        KCXPPint Reason);

◆ void Put(int Hobj,
        KCXPMD pMsgDesc,
        KCXPPMO pPutMsgOpts,
        int BuffLen,
        byte[] Buffer,
        KCXPPint CompCode,
        KCXPPint Reason);

◆ void Get(int Hobj,
        KCXPMD pMsgDesc,
        KCXPGMO pGetMsgOpts,
        int BuffLen,
        byte[] Buffer,
        KCXPPint DataLen,
        KCXPPint CompCode,
        KCXPPint Reason);
◆ void ErrInfo(int Reason,
        StringBuffer ErrInfo,
        boolean bEnglish);
■ KCXPMD
    该类定义了消息描述符
◆ StringBuffer strStructId;    /*struct id */
◆ StringBuffer strVersion;    /*version */
◆ byte cbReport;    /*report flag */
◆ byte cbMsgType;    /*message type */

```



```
◆ int iLifeTime; /*life time (ms) */
◆ int iInitTime; /*enter qm time, unit: ms */
◆ int iFeedback; /*feedback code */
◆ int iEncoding; /*data encode */
◆ int iCodedCharSetId; /*char set id */
◆ int iPriority; /*priority */
◆ byte cbPersistence; /*persistence */
◆ byte cbFlow; /*flow of dest queue */
◆ StringBuffer strMsgId; /*message id */
◆ StringBuffer strReplyToQ; /*reply quene name */
◆ StringBuffer strReplyToQMgr; /*reply queue manager */
◆ StringBuffer strSrcNodeCode; /*source node code */
◆ StringBuffer strChannelName; /*channel name */
◆ StringBuffer strDestNodeCode; /*destination node code */
◆ StringBuffer strDestQm; /*destination queue manager */
◆ StringBuffer strDestQ; /*destination queue */
◆ byte cbRouteType; /*route type */
◆ byte cbCompMode; //compress mode: 1-127;
◆ byte cbEncryptMode; //encrypt mode: 1-127;
◆ int iPutAppType; /*put application type */
◆ StringBuffer strPutAppName; /*put application name */
◆ StringBuffer strPutDate; /*put date */
◆ StringBuffer strPutTime; /*put time */
◆ StringBuffer strGroupId; /*group id */
◆ int iMsgSeqNumber; /*message sequence number */
◆ int iOffset; /*offset value */
◆ byte cbMsgFlags; /*message flag */
◆ int iOriginalLength; /*original message length */
◆ int iDataLength; /*data length */
◆ StringBuffer strReserved; /*reserved */
◆ int copy(KCXPMd MsgDesc);
```

■ KCXPGMO

该类定义了 Get 时 GMO 参数。

```
◆ StringBuffer strStruId;
◆ StringBuffer strVersion;
◆ int Options;
◆ int WaitInterval;
◆ int MatchOptions;
◆ byte GroupStatus;
◆ byte SegmentStatus;
◆ byte Segmentation;
```



- ◆ int MsgToken;
- ◆ int ReturnLength;

■ KCXPPMO

该类定义了 Put 时 PMO 参数。

- ◆ StringBuffer strStrucId;
- ◆ StringBuffer strVersion;
- ◆ int Options;
- ◆ int Timeout;

- ◆ int Hobj;

■ KCXPOD

该类定义了 Open 时的参数。

- ◆ StringBuffer strStrucId;
- ◆ StringBuffer strVersion;
- ◆ StringBuffer strObjectName;
- ◆ int iObjectType;
- ◆ StringBuffer strQmgrName;

■ KCXPTools

该类定义了一些数据转换函数。

- ◆ int GetValue(byte[] src,
byte[] dest,
int num, byte ch);
- ◆ String bytes2string(byte[] buf);
- ◆ void SendMsgDescToBuffer (ByteArray buffer,
KCXPMD MsgDesc,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void GetMsgDescFromBuffer (ByteArray buffer,
KCXPMD MsgDesc,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void SendObjDescToBuffer(ByteArray buffer,
KCXPOD ObjDesc,
KCXPPint CompCode,
KCXPPint Reason);



- ◆ void GetObjDescFromBuffer (ByteArray buffer,
KCXPOD pObjDesc,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void SendPMOToBuffer (ByteArray buffer,
KCXPPMO pPutMsgOpts,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void GetPMOFromBuffer (ByteArray buffer,
KCXPPMO pPutMsgOpts,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void SendGMOToBuffer (ByteArray buffer,
KCXPGMO pGetMsgOpts,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ void GetGMOFromBuffer (ByteArray buffer,
KCXPGMO pGetMsgOpts,
KCXPPint CompCode,
KCXPPint Reason);
- ◆ int CompensateL(StringBuffer s,
int len,
char ch);
- ◆ void catByte(byte[] dest,
int iPos,
byte[] src,
int len);
- ◆ void copyByte(byte[] dest,
byte[] src,
int len);
- ◆ void copyByte(byte[] dest,
byte[] src,
int iPos,
int len);



- ◆ String Bytes2String(byte[] buffer,
int len);
- ◆ int string2byte(byte[] dest,
int len,
String src);

■ KCXPC

该类含有全部的常量定义。

■ KCXPPint

该类含有一个整型成员，用于整型参数的“引用传递”

■ KCXPPbyte

该类含有一个字节成员，用于字节参数的“引用传递”

■ 编程实例

KCXPTest.java 文件中：

```
import com.szkingdom.kcxp.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public class KCXPTest
{
    public static void main(String args[])
    {
        /*初始化，在程序中只能调一次*/
        KCXPAPI.Init();

        KCXPPint CompCode = new KCXPPint();
        KCXPPint Reason = new KCXPPint();
        StringBuffer sBuf = new StringBuffer();

        /*创建一个 api 对象*/
        KCXPAPI api = new KCXPAPI();

        /*创建一个队列句柄对象*/
        KCXPPint Hobj = new KCXPPint();
```




```
KCXPPint Hobj1 = new KCXPPint();

/*创建一个 KCXPOD 句柄对象*/
KCXPOD od = new KCXPOD();

/*创建一个消息头对象*/
KCXPMD MsgDesc = new KCXPMD();

/*创建一个 KCXPPMO 对象*/
KCXPPMO pmo = new KCXPPMO();
int options;

/*建立一个连接*/
/*api.Conn("", CompCode, Reason);
*/
api.Conn("10.100.218.147", 21000, "KCXP00", "888888",
CompCode, Reason);
if (CompCode.x != KCXPC.KCXP_CC_OK)
{
    api.ErrInfo(Reason.x, sBuf, true);
    System.out.println("[Conn] Reason:" + sBuf.toString());
    return;
}

/*打开一个队列*/
od.iObjectType = KCXPC.KCXP_OT_Q;
od.strObjectName.append("TESTQ");
options = KCXPC.KCXP_00_AS_Q_DEF | KCXPC.KCXP_00_BROWSE |
KCXPC.KCXP_00_OUTPUT | KCXPC.KCXP_00_INPUT;

api.Open(od, options, Hobj, CompCode, Reason);
if (CompCode.x != KCXPC.KCXP_CC_OK)
{
    api.ErrInfo(Reason.x, sBuf, true);
    System.out.println("[Open] Reason:" + sBuf.toString());
    options = 0;
    api.Disconn(options, CompCode, Reason);
    return;
}

String Buffer = "I have a dream!!!";
byte [] data = new byte[1024];
```



```
int BuffLen = KCXPTools.string2byte(data, 1024, Buffer);

MsgDesc.iLifeTime = 20000000;
MsgDesc.cbMsgType = KCXPC.KCXP_MT_DATAGRAM;
MsgDesc.strDestQ.append("TESTQ");

pmo.Options=KCXPC.KCXP_PMO_NONE | KCXPC.KCXP_PMO_NEW_MSG_ID;
pmo.Timeout = 20;

for(int i = 0; i < 100; i++)
{
    /*放入一条消息*/
    api.Put(Hobj.x, MsgDesc, pmo, BuffLen, data, CompCode,
    Reason);
    if (CompCode.x != KCXPC.KCXP_CC_OK)
    {
        api.ErrInfo(Reason.x, sBuf, true);
        System.out.println("[Put] Reason: " + sBuf);
        return;
    }
}

api.Open(od, options, Hobj1, CompCode, Reason);
if (CompCode.x != KCXPC.KCXP_CC_OK)
{
    api.ErrInfo(Reason.x, sBuf, true);
    System.out.println("[Open] Reason: " + sBuf.toString());
    options = 0;
    api.Disconn(options, CompCode, Reason);
    return;
}

KCXPPint len = new KCXPPint();
KCXPGMO gmo = new KCXPGMO();
gmo.Options = KCXPC.KCXP_GMO_GET | KCXPC.KCXP_GMO_WAIT;
gmo.MatchOptions = KCXPC.KCXP_MO_MATCH_NONE;
gmo.WaitInterval = 10;

String retbuf = "";
int iCount = 0;
BuffLen = 1024;
```



```
while(iCount < 100)
{
    /*取一条消息*/
    api.Get(Hobj1.x, MsgDesc, gmo, BuffLen, data, len,
    CompCode, Reason);
    if (CompCode.x != KCXPC.KCXP_CC_OK)
    {
        api.ErrInfo(Reason.x, sBuf, true);
        System.out.println("[Get] Reason: " + sBuf);
        return;
    }

    retbuf = KCXPTools.bytes2string(data, len.x);
    KCXPTools.Trace("Msg ID", MsgDesc.strMsgId.toString());
    KCXPTools.Trace("Msg datalen", MsgDesc.iDataLength);
    /*输出调试信息*/
    KCXPTools.Trace("recv data", data, len.x);
    iCount++;
}

/*关闭队列句柄，并释放连接*/
options = KCXPC.KCXP_CO_NONE;
api.Close(Hobj, options, CompCode, Reason);
api.Disconn(0, CompCode, Reason);

return;
}
}
```

附录

附录 A 返回码

■ 完成码 (CompCode) 定义

完成码参数 (CompCode) 能够使得调用者很快检测到该调用是否成功完成、还是部分完成、还是调用失败。

- KCXP_CC_OK
- KCXP_CC_WARNING
- KCXP_CC_FAILED



■ 原因码 (Reason) 定义

# define	KCXP_RC_NONE	0L
# define	KCXP_RC_DEFAULT	1L
# define	KCXP_RC_ALLOC_MEMORY_FAILED	10L
# define	KCXP_RC_INVALID_PARM	100L
# define	KCXP_RC_MESSAGE_OVER_LIFETIME	101L
# define	KCXP_RC_IDENTIFY_NAME	110L
# define	KCXP_RC_IDENTIFY_PASSWORD	111L
# define	KCXP_RC_INVALID_QMGRNAME	120L
# define	KCXP_RC_INVALID_NODECODE	121L
# define	KCXP_RC_ROUTE_NONODE	200L
# define	KCXP_RC_NO_QUEUE	201L
# define	KCXP_RC_QUEUE_FILLED	301L
# define	KCXP_RC_QUEUE_INSERT	302L
# define	KCXP_RC_QUEUE_GET	303L
# define	KCXP_RC_QUEUE_MAXMESSAGE_LENGTH	304L
# define	KCXP_RC_QUEUE_PROHIBIT_PUT	305L
# define	KCXP_RC_QUEUE_MAX_DEPTH	306L
# define	KCXP_RC_RECEIVE_FAILED	400L
# define	KCXP_RC_SEND_FAILED	401L
# define	KCXP_RC_LIFETIME_ERROR	2002L
# define	KCXP_RC_HCONN_ERROR	2003L
# define	KCXP_RC_HOBJ_ERROR	2004L
# define	KCXP_RC_MAX_CONNS_LIMIT_REACHED	2005L
# define	KCXP_RC_MD_ERROR	2006L
# define	KCXP_RC_MISSING_RESPONSE_Q	2007L
# define	KCXP_RC_MSG_TYPE_ERROR	2008L
# define	KCXP_RC_MSG_TOO_BIG_FOR_Q	2009L
# define	KCXP_RC_MSG_TOO_BIG_FOR_QMGR	2010L
# define	KCXP_RC_NO_MSG_AVAILABLE	2011L
# define	KCXP_RC_NO_MSG_UNDER_CURSOR	2012L
# define	KCXP_RC_NOT_OPEN_FOR_BROWSE	2013L
# define	KCXP_RC_NOT_OPEN_FOR_GET	2014L
# define	KCXP_RC_NOT_OPEN_FOR_INQUIRE	2015L
# define	KCXP_RC_NOT_OPEN_FOR_PUT	2016L
# define	KCXP_RC_NOT_OPEN_FOR_SET	2017L
# define	KCXP_RC_OBJECT_CHANGED	2018L
# define	KCXP_RC_OBJECT_IN_USE	2019L
# define	KCXP_RC_OBJECT_TYPE_ERROR	2020L
# define	KCXP_RC_OD_ERROR	2021L
# define	KCXP_RC_OPTION_NOT_VALID_FOR_TYPE	2022L



# define	KCXP_RC_OPTION_ERROR	2023L
# define	KCXP_RC_PERSISTENCE_ERROR	2024L
# define	KCXP_RC_PERSISTENT_NOT_ALLOWED	2025L
# define	KCXP_RC_PRIORITY_EXCEEDS_MAXIMUM	2026L
# define	KCXP_RC_PRIORITY_ERROR	2027L
# define	KCXP_RC_Q_DELETED	2028L
# define	KCXP_RC_Q_FULL	2029L
# define	KCXP_RC_Q_NOT_EMPTY	2030L
# define	KCXP_RC_Q_TYPE_ERROR	2031L
# define	KCXP_RC_Q_MGR_NAME_ERROR	2032L
# define	KCXP_RC_Q_MGR_NOT_AVAILABLE	2033L
# define	KCXP_RC_REPORT_OPTIONS_ERROR	2034L
# define	KCXP_RC_TRIGGER_CONTROL_ERROR	2035L
# define	KCXP_RC_TRIGGER_DEPTH_ERROR	2036L
# define	KCXP_RC_TRIGGER_MSG_PRIORITY_ERROR	2037L
# define	KCXP_RC_INTR_ERROR	2038L
# define	KCXP_RC_AGAIN_ERROR	2039L
# define	KCXP_RC_IO_ERROR	2040L
# define	KCXP_RC_INVALID_ERROR	2041L
# define	KCXP_RC_FAULT_ERROR	2042L
# define	KCXP_RC_SOCKET_ERROR	2043L
# define	KCXP_RC_CONNECT_ERROR	2044L
# define	KCXP_RC_NO_USER_ERROR	2045L
# define	KCXP_RC_PASSWORD_ERROR	2046L
# define	KCXP_RC_LB_ERROR	2047L
# define	KCXP_RC_NODE_TOO_BUSY	2048L
# define	KCXP_RC_CHANNEL_NAME_ERROR	2049L
# define	KCXP_RC_NODE_CODE_ERROR	2050L
# define	KCXP_RC_NOT_ROUTE_ERROR	2051L
# define	KCXP_RC_QM_STOP	2052L
# define	KCXP_RC_QM_PAUSE	2063L
# define	KCXP_RC_QM_DISABLED	2064L
# define	KCXP_RC_Q_NAME_ERROR	2053L
# define	KCXP_RC_CONNECTION_BROKEN	2054L
# define	KCXP_RC_CONNECTION_STOPPING	2055L
# define	KCXP_RC_SELECTOR_COUNT_ERROR	2056L
# define	KCXP_RC_SELECTOR_ERROR	2057L
# define	KCXP_RC_SELECTOR_LIMIT_EXCEEDED	2058L
# define	KCXP_RC_CHAR_ATTRS_TOO_SHORT	2059L
# define	KCXP_RC_INT_ATTR_COUNT_TOO_SMALL	2060L
# define	KCXP_RC_SELECTOR_NOT_FOR_TYPE	2061L
# define	KCXP_RC_OPTIONS_ERROR	2062L



```
# define KCXP_RC_OPER_OBJECT_ERROR 2065L
# define KCXP_RC_GMO_ERROR 2066L
# define KCXP_RC_READ_CONF_Q_MGR_NAME_ERROR 2067L
# define KCXP_RC_READ_CONF_Q_MGR_IP_ERROR 2068L
# define KCXP_RC_READ_CONF_Q_MGR_USP_ERROR 2069L
# define KCXP_RC_Q_MGR_NAME_DEFINE_ERROR 2070L
# define KCXP_RC_Q_MGR_NAME_MATCH_ERROR 2071L
# define KCXP_RC_READ_CONF_Q_MGR_USER_NAME_ERROR 2072L
# define KCXP_RC_READ_SHADOW_ERROR 2073L
# define KCXP_RC_Q_MGR_EXIST 2074L
# define KCXP_RC_Q_MGR_UN_CREATE 2075L
# define KCXP_RC_PARAMER_ERROR 2076L
# define KCXP_RC_SOCKET_ERROR 2077L
# define KCXP_RC_OPEN_USER_INFO_FILE_ERROR 2078L
# define KCXP_RC_READ_USER_INFO_FILE_ERROR 2079L
# define KCXP_RC_OPEN_PROC_LOADAVG_ERROR 2080L
# define KCXP_RC_OPEN_PROC_MEMINFO_ERROR 2081L
# define KCXP_RC_DATA_ERROR 2082L
# define KCXP_RC_GET_FLOW_ERROR 2083L
# define KCXP_RC_ARRIVE_VAL_DEPTH 2084L
# define KCXP_RC_OPEN_FILE_ERROR 2085L
# define KCXP_RC_EVENT_LIST_FULL 2086L
# define KCXP_RC_NOT_FOUND_EVENT_DEFINE 2087L
# define KCXP_RC_ENCRYPT_ERROR 2088L
# define KCXP_RC_DECRYPT_ERROR 2089L
# define KCXP_RC_COMPRESS_ERROR 2090L
# define KCXP_RC_UNCOMPRESS_ERROR 2091L
# define KCXP_RC_READ_DATA_TIMEOUT_ERROR 2092L
# define KCXP_RC_NO_WAIT_TIME 2093L
# define KCXP_RC_LOADBALANCE_FAIL 2094L
# define KCXP_RC_MALLOC_ERROR 2095L
# define KCXP_RC_DLOPEN_COMPRESS_SO_ERROR 2096L
# define KCXP_RC_DLSYM_COMPRESS_SO_ERROR 2097L
# define KCXP_RC_DLOPEN_CRYPT_SO_ERROR 2098L
# define KCXP_RC_DLSYM_CRYPT_SO_ERROR 2099L
# define KCXP_RC_QD_NOT_FOUND 4000 L
# define KCXP_RC_QU_MALLOC_ERROR 4001 L
# define KCXP_RC_NO_WRITE_PERM 4002 L
# define KCXP_RC_REACH_MAX_DEPTH 4003 L
# define KCXP_RC_REACH_VAL_DEPTH 4004 L
# define KCXP_RC_MSG_TOO_LARGE 4005 L
# define KCXP_RC_QUEUE_EMPTY 4006 L
```

```

# define KCXP_RC_NO_READ_PERM 4007 L
# define KCXP_RC_QUEUE_TYPE_ERROR 4008 L
# define KCXP_RC_QUEUE_SHARE_ERROR 4009 L
# define KCXP_RC_QUEUE_PERM_ERROR 4010 L
# define KCXP_RC_CREATE_QUEUE_ERROR 4011 L
# define KCXP_RC_SET_ATTR_NOT_PERM 4012 L
# define KCXP_RC_ATTR_NOT_FOUND 4013 L
# define KCXP_RC_INVALID_MODE 4014 L
# define KCXP_RC_REMOVE_QS_ERROR 4015 L
# define KCXP_RC_EMPTY_Q_ERROR 4016 L
# define KCXP_RC_MSG_NOT_FOUND 4017 L
# define KCXP_RC_TRIGGER_FIRST_MSG 4018 L
# define KCXP_RC_TRIGGER_DEPTH_MSG 4019 L
# define KCXP_RC_TRIGGER_EACH_MSG 4020 L
# define KCXP_RC_CHANNEL_OPERATION_TOBEGIN 5000L
# define KCXP_RC_CHANNEL_OPERATION_NO_TOEND 5001L
# define KCXP_RC_CHANNEL_OPERATION_TOEND 5002L
# define KCXP_RC_CHANNEL_EXIST 5003L
# define KCXP_RC_CHANNEL_PATH_EXIST 5004L
# define KCXP_RC_CHANNEL_NOT_EXIST 5005L
# define KCXP_RC_CHANNEL_INVALID_CHANNEL 5006L
# define KCXP_RC_CHANNEL_INVALID_PATH 5007L
# define KCXP_RC_CHANNEL_STATUS_STOP 5008L
# define KCXP_RC_ROUTE_NOT_EXIST 5600L
# define KCXP_RC_REMOTEDDEF_EXIST 6000L
# define KCXP_RC_REMOTEDDEF_NO_EXIST 6001L
# define KCXP_RC_PLUGIN_OPEN 7000L
# define KCXP_RC_PLUGIN_GET 7001L
# define KCXP_RC_PLUGIN_CALL 7002L
# define KCXP_RC_PLUGIN_NOT_FIND 7003L
# define KCXP_RC_MESSAGE_PLUGIN 7010L
# define KCXP_RC_SYSTEM_ERROR 9999L

```

附录 B KCXP 常数定义

附录 C API 的选项确认规则