Kingstar

金仕达

KSMDB 用户手册

文档标识

文档名称	KSMDB 金仕达内存数据库用户手册
版本号	<V1.0>
状况	○ 草案 ○ 评审过的 ○ 更新过的 ◉ 定为基线的

文档修订历史

版本	日期	描述	修订者
V1.0	2020/1/14	基于 KSMDB 金仕达内存数据库 V2.3 用户手册创建	朱鸿斌

正式核准

姓名	签字	日期

分发控制

副本	接受人	机构

目 录

1. 产品概述

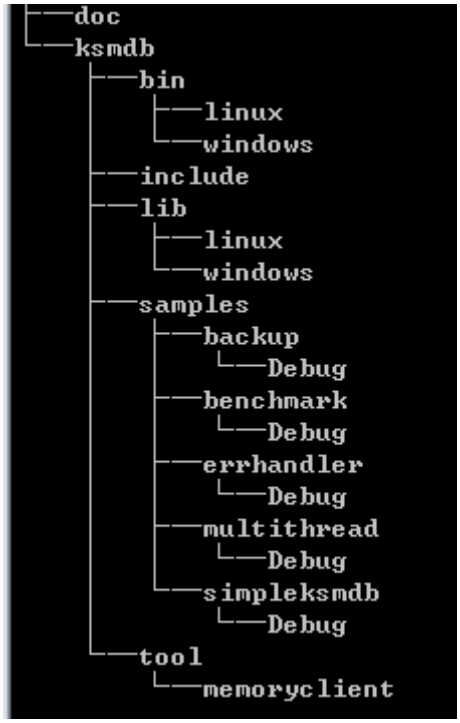
1.1. 产品概述

本手册为 KSMDB 金仕达内存数据库（以下简称 KSMDB）用户手册。KSMDB 是嵌入式内存数据库，可以为企业级服务器端应用系统和嵌入式系统提供高性能数据存取服务。KSMDB 提供符合标准 ACID 规范的事务特性，由于数据存取访问路径足够短，KSMDB 在高并发事务访问场景仍然具有极速性能表现。

1.2. 获取 KSMDB 金仕达内存数据库

KSMDB 金仕达内存数据库对外表现为一个静态链接库。为了支持 KSMDB 静态链接库的开发与使用，发布包同时包括工具集和文档。发布包的官方发布地址是：https://github.com/ksdbdev/ksmdb_eval.git。

发布包的内容说明如下：



./doc： 文档目录，主要是用户操作手册等文档。

./ksmdb/bin：KSMDB 内核库和工具，包括 linux 和 windows 两个平台。

./ksmdb/include： KSMDB 内核库头文件

./ksmdb/lib： KSMDB 内核库和依赖库

./ksmdb/samples： KSMDB 示例

./ksmdb/tool： KSMDB 内存库相关工具

2. 上手指南

KSMDB 对应用层提供的是 C++形式的接口。开发基于 KSMDB 的应用系统一般分成以下几步：

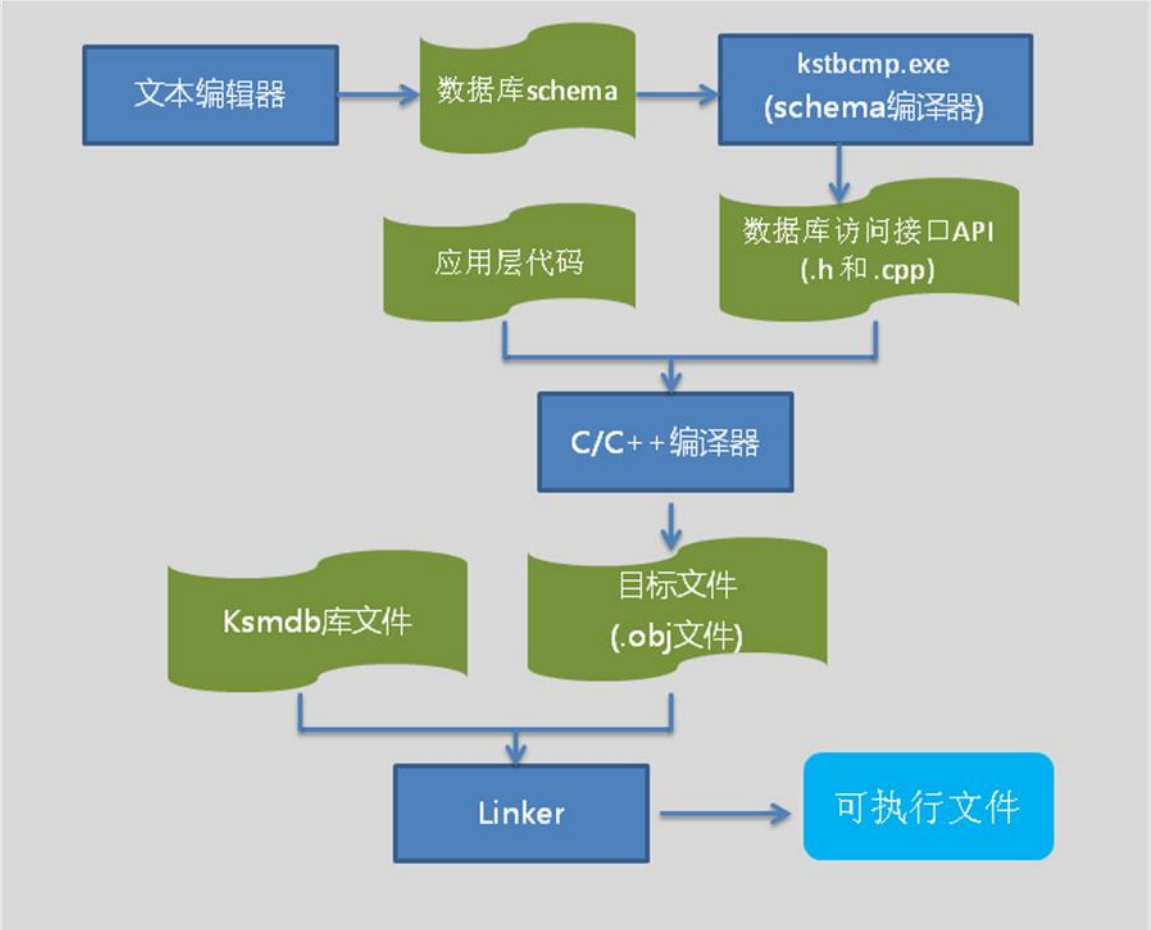
1）分析业务系统的数据模型，按 KSMDB 的标准定义数据库 schema；

2）利用 KSMDB 的模型编译工具 kstbcmp.exe 编译数据库 schema，生成 C++形式的数据库访问接口（.h 和.cpp 文件）。

3） 把数据库访问接口文件导入应用开发工程，编写基于 KSMDB 内存数据库的应用系统代码。

4）编译应用系统代码，链接 KSMDB 内存库的库文件，生成应用系统可执行程序。

这些步骤的示意图如下：



2.1. 数据建模

按一定的格式规范，定义表结构（字段、索引）。

```
declare database student;
class grade
{
    int4                id;
    string<50>          name;
    string<10>          classroom;
    double<0>           math;
    double<0>           Chinese;
    double<0>           english;
    unique tree<id>     pkey;
    tree<name>          nameindex;
    tree<classroom>     classroom;
    tree<name,classroom>    nameclassroom;
};
```

在数据建模时，数据库表字段定义可以是以下数据类型：

字段类型	参考标准变量类型 (32 位平台)	说明
char	char	字符型
uchar	unsigned char	无符号字符型
int2	short	16 位整数
uint2	unsigned short	无符号 16 位整数
int4	int	32 位整数
uint4	unsigned int	无符号 32 位整数
int8	long long	64 位整数
double<n>	double	双精度浮点数，n 表示小数点后数字位数
string<n>	char[n]	字符串，n 表示字符串长度。第 n+1 位会添加字符串截止符（'\0'）。

2.2. 编译 schema

利用工具（kstbcmp.exe）生成 C++形式的数据库访问接口。

执行命令行工具：

```
C:\>kstbcmp.exe student.ks studentdb
```

得到数据库访问接口：

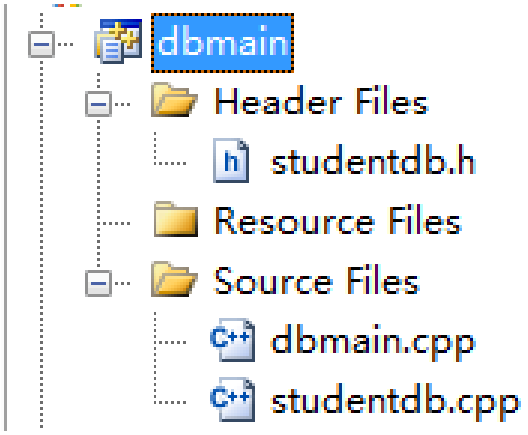
```
class DB_student
{
    static bool m_inited;
public:
    static KsDbHandle m_database;
    DB_student();
    virtual ~DB_student(){};
    static bool Create(char *path,char *name,long size);
    static bool Open(char *path,char *name,int mode,int port,bool isolate);
    static void UnLockDb();
    static void SetErrorHandle(KsErrorHandle handle);
    static int  GetCols(KsRecordHandle handle);
    static char *GetColName(KsRecordHandle handle,int col);
    static int  GetColType(KsRecordHandle handle,int col);
    static int  GetColSize(KsRecordHandle handle,int col);
    static int  GetColDecimal(KsRecordHandle handle,int col);
    static char *GetColData(KsRecordHandle handle,int col);
    static void GetMemoryInfo(long &size,long &inused);
    static void Reset();
    static bool Backup(char *filename);
    static bool Load(char *filename);
    static void BeginTransaction();
    static void RollbackTransaction();
    static void CommitTransaction();
    static void GetLastError(int *errcode,char *errmsg);
    static bool LoadLicense(char *licensefilename,char *errmsg);
    static void GetLicenseInfo(int &controlexpitedate,int &expitedate,int &
    static void Close();
    static char *LibVersion();
    static char *DbVersion();
};
```

```
class TB_grade
{
public:
    KsRecordHandle m_record;
    KsConditionHandle m_condition;
public:
    TB_grade();
    TB_grade(TB_grade &V);
    virtual ~TB_grade();
    bool newrecord();
    bool append(int id,char* name,char* classroom);
    bool erase();
    bool first();
    bool prev();
    bool next();
    bool end();
    bool dump(char *filename="grade");
    bool SaveTXT(char *filename="grade.txt");
    bool LoadTXT(char *filename="grade.txt");
    bool backup(char *filename="grade");
    bool load(char *filename="grade");
    int  get_id();
    char* get_name();
    char* get_classroom();
    bool set_math(double value);
    double get_math();
    bool set_Chinese(double value);
    double get_Chinese();
    bool set_english(double value);
    double get_english();
    bool find_by_pkey(int id,int op=EQ);
    bool find_by_nameindex(char* name,int op=EQ);
    bool find_by_classroom(char* classroom,int op=EQ);
    bool find_by_nameclassroom(char* name,char* classroom,
    bool lists();
    bool list_by_pkey();
    bool list_by_nameindex();
    bool list_by_classroom();
    bool list_by_nameclassroom();
};
```

工具同时生成 db.script 文件。在启动 ksmdbmanage 的时候，需要该文件。

2.3. 编写业务代码

导入接口文件，编写基于 KSMDB 内存库的应用系统业务代码。



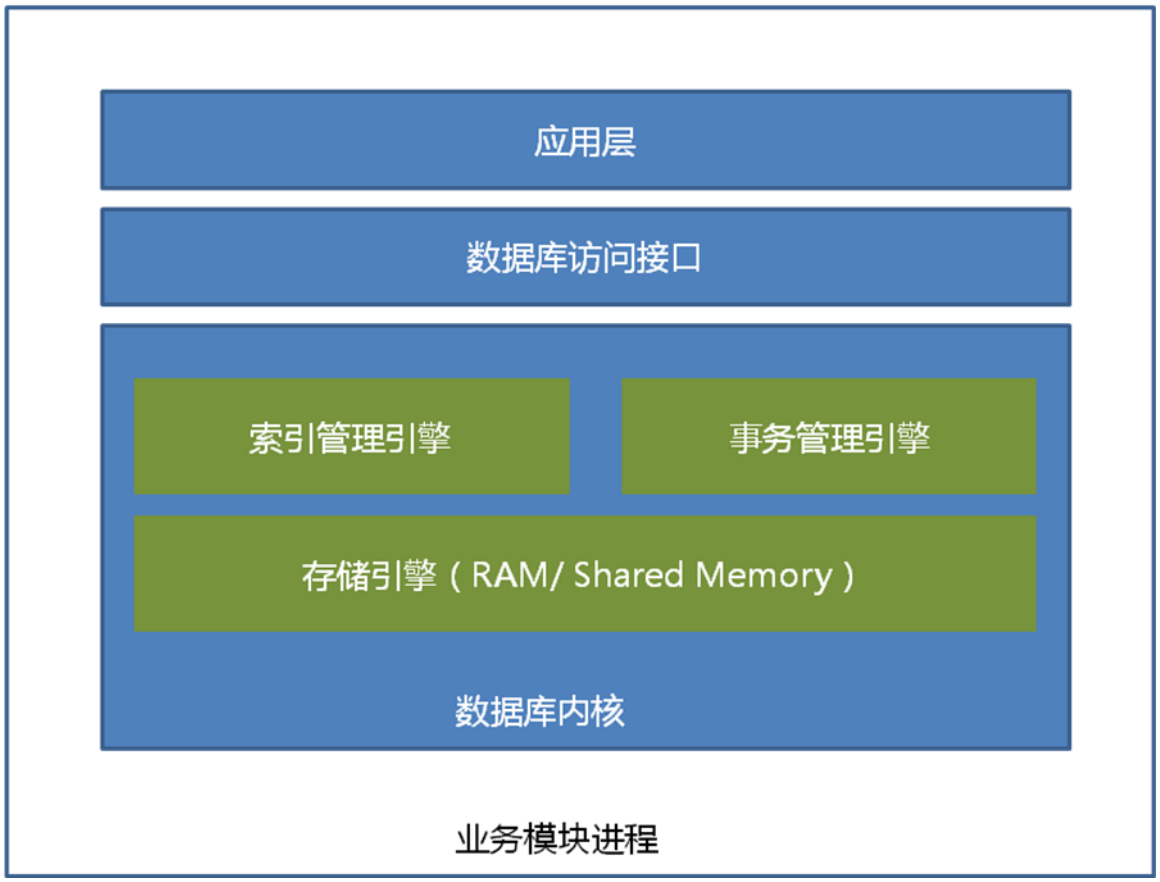
```
DB::create()
 DB_student::BeginTransaction
 grade.newrecord();
 grade.set_field();
 grade.append();
 DB_student::CommitTransaction();
 DB_student::BeginTransaction();
 if (grade.find_by_nameindex("12",LARGE_EQ)){
    grade.end();
    do{
       grade.get_id()
     }while(grade.prev());
 }
 DB_student::CommitTransaction();
 DB_student::Close();
```

2.4. 链接

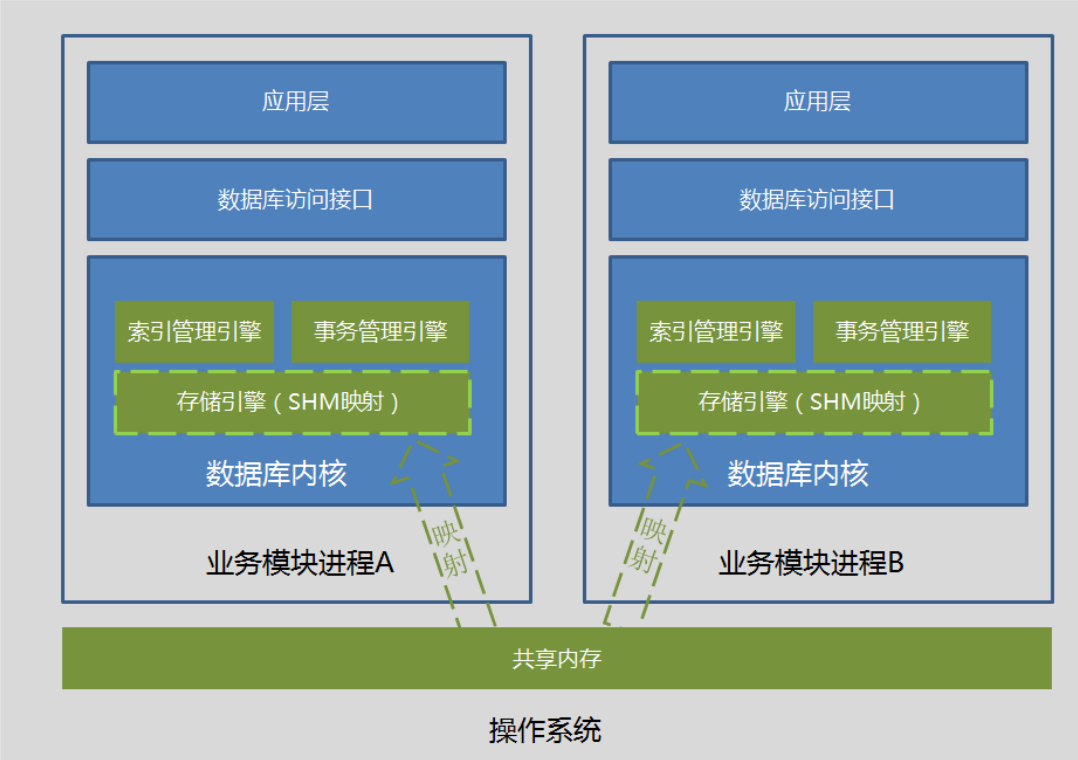
链接 KSMDB 库文件，生成应用系统可执行文件。

3. 体系结构

KSMDB 是嵌入式内存数据库（IMDS），应用层通过 C/C++ API 接口访问数据库。数据库寄生在业务模块的进程里，为应用层提供数据存取服务。数据库本身不生成线程或进程。系统结构参见图：



KSMDB 数据存储引擎基于共享内存实现，共享内存的实现方案在 Windows 平台是基于内存映射文件，在 Unix-like 平台是基于 Syste V 规范。由于基于共享内存实现存储引擎，KSMDB 可以用于多进程数据共享。多进程结构的系统结构图如图：



4. 性能测试

KSMDB 的目标是提供极速数据存储服务。为了实现这个目标，基于 KSMDB 的数据库访问代码路径尽可能短。此外，KSMDB 在数据存取和数据管理的实现过程中，充分考虑 CPU 的缓存利用率，有效减少了 CPU 周期数。通过这些技术策略和手段，KSMDB 在内存数据库领域的性能表现稳居前列。具体性能数据可以参考以下例子：

测试环境：

1）操作系统：CentOS release 6.6 (Final)

2）系统配置：20　Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz

3）内存：128 G

测试场景：1 百万条数据新增、更新、遍历，计算每条记录的操作延时。

测试方法：

$./perf A

测试结果：

insert 1000000 rows,spend 2959 ms,one row cost=2.96 us

find 1000000 times,spend 571 ms,one row cost=0.57 us

update 1000000 rows,spend 674 ms,one row cost=0.67 us

size=1048576000,inused=249260448

结果分析：

perf 是一个测试 KSMDB 读写性能的工具，主要测试单进程单线程场景的单次数据读写性能。每次数据的读写都是一个事务。从数据上看，例子所示的测试环境下，单次数据 insert，平均耗时是 2.96us；单次数据 find，平均耗时是 0.57us；单次数据 update，平均耗时是 0.67us。

5. 事务和并发

KSMDB 支持事务，实现的事务特性符合标准的事务 ACID 规范。事务机制是 KSMDB 内存数据库区别于其他基于内存的数据表或数据结构的特性。事务特性确保对内存数据库的数据存取操作要么完成，要么回滚到操作前的状态，数据库内核确保存储是数据具有完整性和一致性。有了事务特性的支持，基于 KSMDB 内存数据库的业务系统可以实现复杂的业务逻辑。

KSMDB 内存数据库所有的数据库存取操作都必须有显示的开启事务和提交/回滚事务的操作。

KSMDB 内存数据库的并发控制是基于数据库的事务实现的。这里说的并发操作都是指事务的并发。每个事务都带有并发属性，可以是只读事务，也可以是读写事务。事务的并发属性继承自数据库访问句柄，在数据库创建或者打开的时候就已经设置好，在数据库打开后，该属性不能修改。

对于数据库读写事务并发控制，SQL 标准定义了 4 种事务隔离级别： Read Uncommitted、Read Committed、 Repeatable Read 、Serializable。常用的是 Read Committed 和 Repeatable Read。如果采用 Read Committed 模式，读事务总是读到最新提交的数据，但是不会读到写事务未提交的数据。如果采用 Repeatable Read 模式，读事务只能读到本事务启动之前提交的数据，读事务进行过程中，写事务可能会更新数据并提交，但是这部分更新结果不会被读事务访问到，读事务多次查询一定条件范围的数据的话，结果集是一致的。

KSMDB 内存数据库事务隔离模式是 Read Committed。为了实现 Read Committed，读写事务并发控制策略如下：

1）如果已有读写事务在进行中，新开启只读事务的时候，只读事务能正常启动。读写事务会锁定更新过程涉及到的表（Add 操作）或者记录（Update 操作）。只读事务访问被锁定的资源时，会去申请只读事务锁，直到读写事务提交。只读事务马上就能看到读写事务提交的最新结果。只读事务可以正常访问未被锁定的数据。

2）如果已有读写事务在进行中，新开启读写事务的时候，读写事务会申请事务读写锁，从而进入阻塞状态，等待前一个读写事务提交。

3）如果已有只读事务在进行中，新开启只读事务的时候，只读事务启动和数据访问都能正常进行。

4）如果已有只读事务在进行中，新开启读写事务的时候，读写事务能正常启动，数据更新动作能正常进行。读写事务会锁定更新过程涉及到的表或者记录。只读事务访问被锁定的资源时，会去申请只读事务锁，直到读写事务提交。只读事务马上就能看到读写事务提交的最新结果。只读事务可以正常访问未被锁定的数据。

6. 多进程和多线程

6.1. 多进程

KSMDB 数据存储引擎基于共享内存实现，这个技术基础给多进程模型的应用系统架构带来很多优势。多进程模型的应用系统如果要相互共享数据，普通的解决方案包括 SOCKET 网络通信方案、文件数据交换方案等。SOCKET 网络通信方案速度受限于网络 IO，文件数据交换方案很难实现高效的数据交互。而基于 KSMDB 数据库来实现多进程模型的应用系统，由于没有 IO 操作，性能表现具有绝对优势，同时还可以使用丰富的进程间同步机制，支持复杂的模块交互逻辑。总的来说，KSMDB 为单机多进程应用系统提供了非常好的系统架构解决方案。

KSMDB 提供了 ksmdbmanage 组件。在多进程场景中，ksmdbmanag 可以用于管理数据库，主要提供数据库的创建和关闭功能。同时 ksmdbmanag 还提供了 tcp 接口，客户端程序可以通过 tcp 接口查看数据库的数据。这些特性和组件丰富了 KSMDB 的运维功能，大大提高了 KSMDB 的可用性。

6.2. 多线程

KSMDB 给应用层提供的接口是 C++形式的接口，具体表现为一个数据库类和多个数据表类。应用层通过这些数据库类和数据表类，完成对业务数据的存取访问。数据库类管理一个数据库句柄，这个句柄不是线程安全，原则上应该每个线程应该拥有自己的数据库句柄。如果 C++类的数据存取方法只有一个线程调用，那么这些数据存取方法依赖的数据库句柄可以由该 C++的成员持有；如果数据存取方法有多个线程调用，那么这些数据存取方法依赖的数据库句柄建议保存在线程本地变量。

7. 数据模型

在数据库开发过程中，提到数据模型，一般涉及到两方面的事情：一是业务模型的设计；二是开发接口的形式。基于 KSMDB 内存数据库的开发过程中，业务模型的设计本质上接近关系型模型，业务数据组织成二维表，不支持面向对象的继承、组合、嵌套等概念。而业务模型设计时的形式类似 C 语言的结构体。

8. 管理、部署和运维

8.1. ksmdbmanage

ksmdbmanage 可以看作一个数据库管理服务模块。如果基于 KSMDB 构建的系统是多进程模型，推荐引入 ksmdbmanage。ksmdbmanage 可以承担数据库创建和关闭的功能，其他业务进程打开数据库，完成数据存取操作。ksmdbmanage 对外提供了 tcp 接口，可以用于查看数据库数据，也可以更新数据库数据。通过 ksmdbmanage 的 tcp 接口更新数据库数据，更新动作的结果的会实时反应到所有访问本数据库的进程，所以对线上系统而言，这类操作必须慎重。

引入 ksmdbmanage 模块之后，一个有意义的架构解决方案是基于 ksmdbmanage 完成对系统数据的导入导出。对于基于内存数据库的实时系统而言，数据实时导入导出是一个常见的需求，比如金融交易系统的数据上场和下场。如果系统引入 ksmdbmanage 模块，不需要其他相关业务模块启动，也能完成数据的导入或者导出，简单高效的实现了大型实时系统的数据准备或者备份。

如果是多线程模型，一般是业务进程自己负责数据库的创建和关闭。ksmdbmanage 可以以“只读”模式打开数据库。ksmdbmanage 打开数据库之后，对外提供 tcp 接口，可以用于查看数据库数据。

ksmdbmanage 的配置文件是 ksmdb.ini，配置项参考示例配置文件的注释说明。

ksmdbmanage 的运行方式：

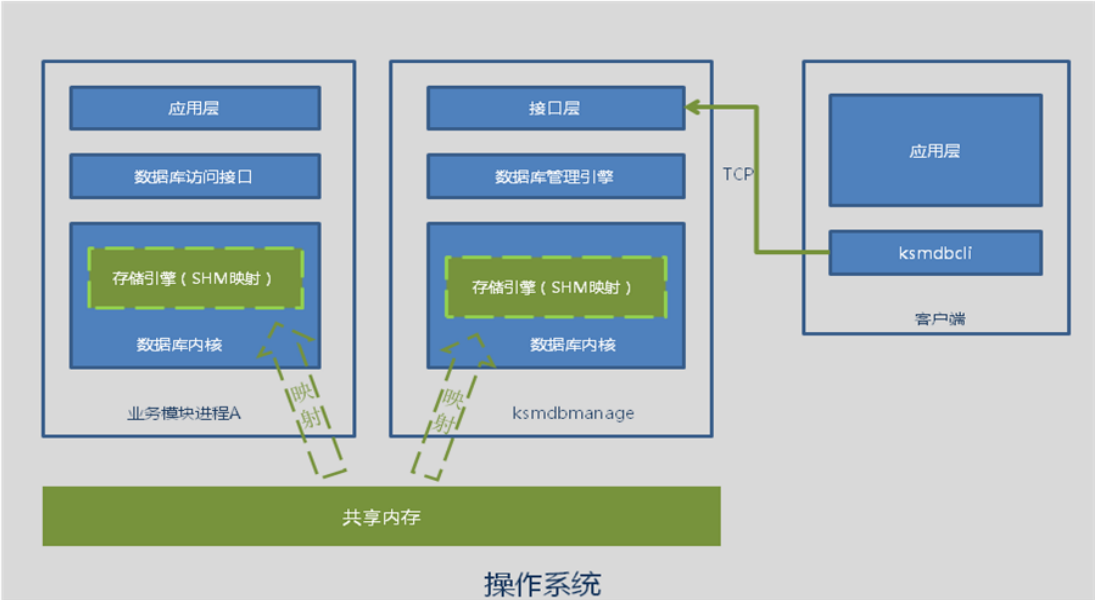
1） Windows 平台：c:\ksmdb\bin\ksmdbmanage –run

2） Linux 平台： /home/user/ksmdb/bin/./ksmdbmanage

8.2. ksmdbcli

ksmdbcli 是一个 api 库，用于对接 ksmdbmanage 的 tcp 接口。通过 tcp 通道，ksmdbcli 可以与 ksmdbmanage 通信，完成对 ksmdbmanage 管理的数据库的访问。ksmdbcli 支持的功能包括连接数据库、查看数据库、查看数据、更新数据等。具体可以参见 ksmdbclient.h。ksmdbcli 和 ksmdbmanage

的关系参见架构示意图。



8.3. MemoryClient.exe

MemoryClient.exe 是一个利用 ksmdbcli.dll 实现的简单的 GUI 工具，可以查看或者维护 ksmdbmanage 管理的数据库

8.4. 部署

KSMDB 的运行环境对 IPC 资源有要求。对于类 Unix 平台的实现的基于 KSMDB 的服务器端应用系统，在部署时，建议修改服务器 IPC 配置如下：

$cat /proc/sys/kernel/sem:

250 3200000 32 12800

$ipcs –l:

------ Shared Memory Limits --------

max number of segments = 4096

max seg size (kbytes) = 67108864

max total shared memory (kbytes) = 17179869184

min seg size (bytes) = 1

------ Semaphore Limits --------

max number of arrays = 12800

max semaphores per array = 250

max semaphores system wide = 3200000

max ops per semop call = 32

semaphore max value = 32767

8.5. IPC 资源清理

对于类 Unix 平台的开发调试环境或者生产环境中，如果基于内存库的应用程序异常退出，可能导致系统的 IPC 资源不能正常释放。所以需要及时清理系统的 IPC 资源。发布包提供 IPC 资源清理脚本 clripc.sh，可以用于清理本 KSDMB 没有正常释放的 IPC 资源。使用方法参考脚本注释。

9. 开发接口

9.1. 数据库级操作接口

*以下内容中 ,“DB”代表自定义的数据库名。DB 是数据库类 ,数据库级的操作接口都是 static 方法，通过类名直接使用。

9.1.1. DB::Create 方法

创建 KSMDB 数据库。数据库句柄保存在 DB 类是实例，通过使用 DB 类的实例来操作数据库和该数据库的表。

函数原型：

*static bool DB::Create(char *path,char *name,long size);*

参数：

path：*数据库路径。*

name：*数据库名称。*

size：*数据库大小，单位：byte。*

9.1.2. DB::Open 方法

打开 KSMDB 数据库。数据库句柄保存在 DB 类是实例，通过使用 DB 类的实例来操作数据库和该数据库的表。

函数原型：

*static bool DB:: Open(char *path,char *name,int mode,int port,bool isolate, int reuse_shmmap, int shmrcset_mode=0);*

参数：

path：*数据库路径。*

name：*数据库名称。*

mode：*打开模式，READWRITE：读写模式；READONLY：只读模式。*

port：*打开端口号，当* ***isolate=true*** *的时候本参数有效，表示连接目标端口号。*

isolate：*隔离模式，true：以隔离模式打开数据库；false：以非隔离模式打开数据库。*

reuse_shmmap：*线程的共享内存映射模式，0：每个线程自己映射共享内存；1：复用本进程共享内存映射空间。*

shmrcset_mode：*缓存介质，0：进程堆内存存储缓存；1：共享内存存储缓存，默认值是 0；*

9.1.3. DB:: SetErrorHandle 方法

设置内存库错误处理函数。

函数原型：

static void DB::SetErrorHandle(KsErrorHandle handle);

参数：

handle：*错误处理函数指针。设置好错误处理函数后，内存库调用该函数输出错误信息。错误处理函数指针的类型是：*

typedef bool (KsErrorHandle)(int errcode,char *errmsg);*

9.1.4. DB:: BeginTransaction 方法

开启事务。

函数原型：

static void DB::BeginTransaction();

9.1.5. DB:: RollbackTransaction 方法

回滚事务。

函数原型：

static void DB:: RollbackTransaction ();

9.1.6. DB:: CommitTransaction 方法

提交事务。

函数原型：

static void DB:: CommitTransaction ();

9.2. 数据表级操作接口

*以下内容中 ,“TB”代表自定义的数据表类。数据表级操作接口的方法通过数据表实例调用。

9.2.1. TB:: newrecord 方法

新增一条数据表的游标，用于新增记录。

函数原型：

bool TB::newrecord();

参数：

无。

9.2.2. TB::set_filed1 方法

给 TB 表的 filed1 字段赋值。

函数原型：

bool TB::set_math(datatype data);

参数：

***data：** 字段新值。*

9.2.3. TB::append()方法

给 TB 表新增游标的索引字段赋值，同时把该游标对应的记录新增到数据库。

函数原型：

bool TB::append(datatype data);

参数：

***data：** 新的记录的索引字段的值。*

9.2.4. TB::get_field1()方法

读取 TB 表的 filed1 值。

函数原型：

datatype TB::get_filed1();

参数：

无。

9.2.5. TB:: find_by_pkey ()方法

根据 pkey 索引查找记录。pkey 是索引名。

函数原型：

bool TB::find_by_pkey(datetype data,int op=EQ);

参数：

data：*查找目标字段值。*

op：*查找条件。*

9.2.6. TB:: lists ()方法

遍历 TB 表的所有记录。

函数原型：

bool TB:: lists ();

参数：

无。

9.2.7. TB:: first ()方法

移动 TB 表的检索游标到第一条记录。

函数原型：

bool TB:: first ();

参数：

无。

9.2.8. TB:: end ()方法

移动 TB 表的检索游标到最后一条记录。

函数原型：

bool TB:: end ();

参数：

无。

9.2.9. TB:: next ()方法

移动 TB 表的检索游标到下一条记录。

函数原型：

bool TB:: next ();

参数：

无。

9.2.10. TB:: prev ()方法

移动 TB 表的检索游标到上一条记录。

函数原型：

bool TB:: prev ();

参数：

无。

9.2.11. TB:: SaveTXT ()方法

把数据库表导出为 txt 格式的文件。

函数原型：

*bool TB:: SaveTXT (char *filename="tb.txt");*

参数：

filename： *导出目标文件名。默认值为“表名.txt”。*

9.2.12. TB:: LoadTXT ()方法

把数据库表从 txt 格式文件导入。

函数原型：

*bool TB:: LoadTXT (char *filename="tb.txt");*

参数：

filename： *导入源文件名。默认值为“表名.txt”。*

9.2.13. TB:: backup ()方法

把数据库表导出为二进制文件。

函数原型：

*bool TB:: backup (char *filename="tb ");*

参数：

filename： *导出目标文件名。默认值为“表名”。*

9.2.14. TB:: load ()方法

把数据库表导出为二进制文件。

函数原型：

*bool TB:: load (char *filename="tb ");*

参数：

filename： *导入源文件名。默认值为“表名”。*

10.技术社区和问题反馈

1）技术支持 QQ 群：1048263371

2）产品发布渠道：https://github.com/ksdbdev/ksmdb_eval

3）技术支持邮箱：hongbin.zhu@kingstarfintech.com

4）公司联系方式：

名称：上海金仕达软件科技有限公司

地址：上海市浦东新区亮景路 210 号

邮编：201203