本地执行器

- 修订

更新内容	时间	编写人	版本
虚拟机3.0使用接口	2019-05-29	李宇哲	v3.0
添加6.6接口调用流程	2020-04-22	黄绿游	v3.1

一、虚拟机sdk说明

用户语言虚拟机具有跨平台特性，基于c99规范进行开发，不直接依赖于系统相关的接口；目前提供如下版本：

- liunx环境的虚拟机动态库：libuvm3.so
- window环境的虚拟机动态库：uvm3.dll
- 单片机的虚拟机静态库：libuvm3.a

二、SDK使用说明

1. 下载对应版本的sdk
2. 将虚拟机动态库和头文件添加到项目中
3. 直接接口调用的方式调用接口

三、 虚拟机提供的接口

3.1 common_reg_init

- 完成功能

  注册虚拟机

- 函数定义

```
lbool common_reg_init(CommonGlobalModule g_module, UTableCB* tableCB, ULogCB logC
B, luint8 isKeyAndValueSave,luint32 bitFlag);
```

- 参数说明

输入参数说明

参数名称	参数说明	备注
g_module	应用必须实现的回调函数	必填
tableCB	外部数据存储的回调函数	如果不用，可以填NULL

参数名称	参数说明	备注
logCB	日志的回调函数	如果不用，可以填NULL
isKeyAndValueSave	是否保存虚拟机运行数据	填0不保存，填1保存
bitFlag	是否进行表的数据代理标识	VARTABLEBIT (1<<0) VARIDTABLEBIT (1<<1) FUNCTABLEBIT (1<<2) LOGICTABLEBIT (1<<3) INDEXSTABLEBIT (1<<4) INFOTABLEBIT (1<<5) TRANSTABLEBIT (1<<6) KEYANDVALUETABLEBIT (1<<7) LANGUAGETABLEBIT (1<<8)

- 返回值

失败返回 0，成功返回1

CommonGlobalModule类型定义如下：

```
typedef struct _CommonGlobalModule

{

    UAppFunc        appFunc;        /*< 应用系统函数入口 */

    UNoticeVar      noticeVar;      /*< 通知应用变量值更新 */

    UGetVMVar       getVMVar;       /**< 获取时间变量 */

    UGetNoticeVar   getNoticeVar;   /**< 数据外部代理用的回调，通知外部该数据将获取，外部是否需要提前更新变量 */

    UAppFrustrum    appFrustrum;    /**< 应用系统并行函数入口 */

    UReadBin        readBin;        /**< 读取bin文件数据 */

    UTaskNew        taskNew;        /**< 创建任务 */

    UMutexNew       mutexNew;       /**< 创建互斥锁 */

    UMutexLock      mutexLock;      /**< 加锁互斥锁 */
```

UTableCB类型定义如下：

```
typedef struct _UTableCB{
    TableCBExist tableCBExist;        /**< 外部数据表是否存在 */
    TableCBGet tableCBGet;            /**< 获取外部数据 */
    TableCBSet tableCBSet;            /**< 设置外部数据*/
    TableCBFree tableCBFree;          /**< 释放外部数据缓存*/
}UTableCB;
```

ULogCB类型定义如下：

```
typedef char(*ULogCB)(const char* datasourceid, const char* pStr, luint32 iLen, luin
t8* eLogPriority);
```

字段名称	字段说明	备注
appFunc	应用系统函数入口	请参考 4.1
noticeVar	通知应用更新变量	请参考 4.2
readbin	读取bin文件数据块	请参考 4.3
…		

3.2 create_vm

- 完成功能

  创建虚拟机

- 函数定义

  ```
  uvm* create_vm(const char* datasourceid, UVMErrorCode* pErrCode);
  ```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	数据源和bin文件是绑定关系

- 输出参数说明

参数名称	参数说明	备注

参数名称	参数说明	备注

pErrCode	错误码	NONE = 0x0, // 无错误
		ERR_VARTABLEINIT, // 变量表初始化失败
		ERR_FUNCTABLEINIT, //函数表初始化失败
		ERR_LOGICTABLEINIT, //逻辑表初始化失败
		ERR_INDESXTABLEINIT, //索引表初始化失败
		ERR_KEYANVALUETABLEINIT, // 键值对表初始化失败
		ERR_TRANSTABLEINIT, // 状态机迁移表初始化失败
		ERR_INFOTABLEINIT, // 状态机信息表初始化失败
		ERR_INIT_GLOBAL, // 全局回调注册出错
		ERR_GET_DATA, // 获取逻辑数据出错
		ERR_INVALID_LOGIC_TYPE,// 无效的逻辑类型
		ERR_GET_SYS_RESC, // 获取系统资源出错
		ERR_INVALID_PARAM, // 无效的入参
		ERR_MALLOC, // 内存申请失败
		ERR_LOAD_DATA, // 加载数据出错
		ERR_INVLAID_SM_NO, // 找不到指定的状态机 ERR_OTHER // 其他错误

- 返回值

  返回虚拟机的地址

3.3 common_vm_run

- 完成功能

  虚拟机运行

- 函数定义

```
lbool common_vm_run(const uvm* vm, var* Varlist, luint32 count,lbool isSave, char
* pErrCode);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	
Varlist	变量列表	
count	列表个数	
isSave	是否保存Varlist传入的变量值	

```
typedef struct _VarPtr{
    luint8 *path;    //产品属性路径
    lint8 type;    //数据类型 -1 应用变量 -2虚拟机变量
    lint32 varid;   //变量id,默认填-1，需要代理去查找
}VarPtr;


typedef struct _var{
    VarPtr* vmVarPtr;
    object* value;
}var;
```

- 输出参数说明

参数名称	参数说明	备注
pErrCode	错误码	0, // 无错误

- 返回值

  成功返回1，失败返回0

3.4 common_vm_free

- 完成功能

  虚拟机释放内存

- 函数定义

```
void common_vm_free(uvm* vm);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	

3.5 getVar

- 完成功能

  获取变量数据

- 函数定义

```
object* getVar(const uvm* vm, VarPtr* vmVarPtr);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	
vmVarPtr	变量信息	

```
typedef struct _VarPtr{
    lchar *path;    //产品路径
    lint8 type;    //数据类型 -1 应用变量 -2虚拟机变量
    lint32 varid;   //变量id,默认填-1，需要代理去查找
}VarPtr;
```

- 返回值

  Object的变量数据

3.6 setVar

- 完成功能

  更新变量数据

- 函数定义

```
    lbool setVar(const uvm* vm, var* data);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	
var	变量数据	

```
typedef struct _VarPtr{
    lchar *path;    //产品路径
    lint8 type;    //数据类型 -1 应用变量 -2虚拟机变量
    lint32 varid;   //变量id,默认填-1，需要代理去查找
}VarPtr;


typedef struct _var{
     VarPtr* vmVarPtr;
    object* value;
}var;
```

- 返回值

  Object的变量数据

3.7 common_vm_tick

- 完成功能

用于捕获定时器信息，主机每iMSecInterval毫秒调用一次

- 函数定义

```
`void common_vm_tick(uvm* vm,luint32 iMSecInterval);`
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	
iMSecInterval	毫秒数	

- 返回值

无

- 注解

该函数仅在语句中使用到了公共延时函数时，才必须调用，其他情况下可以不用理会。

3.8 getAppinfoOffset

- 完成功能

  根据虚拟机来获取应用扩展区的偏移量

- 函数定义

```
luint32 getAppinfoOffset(const uvm* vm, luint32* size);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机的地址	

- 输出参数说明

参数名称	参数说明	备注
size	偏移大小	

- 返回值

  返回应用扩展区的偏移地址

3.9 getAppinfoOffset_ex

- 完成功能

  根据应用数据源来获取应用扩展区的偏移量

- 函数定义

```
luint32 getAppinfoOffset_ex(const char* datasourceid, luint32* size);
```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用数据源	

- 输出参数说明

参数名称	参数说明	备注
size	偏移大小	

- 返回值

  返回应用扩展区的偏移地址

3.10 getVersion

- 完成功能

  获取虚拟机版本

- 函数定义

```
const char* getVersion();
```

- 返回值

  返回版本号

3.11 u_coreglobal_version

- 完成功能

  获取虚拟机core层的版本

- 函数定义

```
const char* u_coreglobal_version();
```

- 返回值

  返回版本号

四、虚拟机提供的通用数据转换接口

4.1 getObjectType

- 完成功能

  获取通用数据类型

- 函数定义

```
OTpye getObjectType(const object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 返回值

  执行结果；数据类型

```
DataType_CHAR,
DataType_SHORT,
DataType_INT,
DataType_FLOAT,
DataType_STRING
```

4.2 objectToChar

- 完成功能

  通用数据类型object转换成char数据类型

- 函数定义

```
char objectToChar(const object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 返回值

  执行结果；转换的数据结果

4.3 charToObject

- 完成功能

  char数据类型转换成通用数据类型object

- 函数定义

```
object* charToObject (int data);
```

- 输入参数说明

参数名称	参数说明	备注
data	char数据	

- 返回值

  执行结果；转换的数据结果

4.4 objectToShort

- 完成功能

  通用数据类型object转换成short数据类型

- 函数定义

```
short objectToShort(const object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 返回值

  执行结果；转换的数据结果

4.5 shortToObject

- 完成功能

  short数据类型转换成通用数据类型object

- 函数定义

```
object* shortToObject (int data);
```

l 输入参数说明

参数名称	参数说明	备注
data	short数据	

- 返回值

  执行结果；转换的数据结果

4.6 objectToInt

- 完成功能

  通用数据类型object转换成int数据类型

- 函数定义

```
int objectToInt(const object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 返回值

  执行结果；转换的数据结果

4.7 intToobject

- 完成功能

  Int数据类型转换成通用数据类型object

- 函数定义

```
object* intToObject (int data);
```

- 输入参数说明

参数名称	参数说明	备注
data	int数据	

- 返回值

  执行结果；转换的数据结果

4.8 objectToString

- 完成功能

  通用数据类型object转换成string数据类型

- 函数定义

```
Const char* objectToString (const object* data,int* pSize);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 输出参数说明

参数名称	参数说明	备注
pSize	数据长度	

- 返回值

  执行结果；转换的数据结果

4.9 stringToObject

- 完成功能

  string数据类型转换成通用数据类型object

- 函数定义

```
object* stringToObject (const char* data , unsigned int size);
```

- 输入参数说明

参数名称	参数说明	备注
data	char*	
Size	数据长度	

- 返回值

执行结果；转换的数据结果

4.10 objectToFloat

- 完成功能

  通用数据类型object转换成float数据类型

- 函数定义

```
float objectToFloat (const object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	object数据	

- 返回值

  执行结果；转换的数据结果

4.11 floatToObject

- 完成功能

  float数据类型转换成通用数据类型object

- 函数定义

```
object* floatToObject (float data);
```

- 输入参数说明

参数名称	参数说明	备注
data	float数据	

- 返回值

  执行结果；转换的数据结果

4.12 objectFree

- 完成功能

  数据的内存释放

- 函数定义

```
void objectFree (object* data);
```

- 输入参数说明

参数名称	参数说明	备注
data	要释放的数据	

五、 应用系统必须实现的接口

5.1 ComAppFunc

- 完成功能

  应用系统函数入口

- 函数定义

```
typedef char(*UAppFunc)(const char* datasourceid, const uvm* vm, const FunctionIn
fo* data, object** pRetData);
```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
vm	虚拟机的地址	
data	函数的数据	

- 输出参数说明

参数名称	参数说明	备注
pRetData	函数执行的返回值	

- 返回值

  执行结果；成功返回1，失败返回0;

注：

```
typedef struct _FunctionInfo{
    lint32      functionId;           //函数ID
    lint32      functionType;        //函数类型
    luint8        parasNum;           //入参数量
    object**    parasArray;           //形参类型/和数据的列表
    lchar*        functionKey;        //函数路径
    luint8      isSynchronize;        //是否异步
    luint8      returnDataType;     //返回类型

}FunctionInfo;
```

5.2 noticeVar

- 完成功能

  通知应用设置变量

- 函数定义

```
typedef char(*UNoticeVar)(const char* datasourceid, const uvm* vm, char type, var
* data);
```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
vm	虚拟机的地址	
type	事件类型	0：通知应用，虚拟机将开始更新变量； 1：通知应用， 虚拟机已经更新完变量
data	虚拟机的变量	包含变量的值，大小，类型

- 返回值

  执行结果；成功返回1，失败返回0; 如果失败，则变量数据进行数据回滚，回退到更新前。

5.3 UGetVMVar

- 完成功能

获取时间点变量的数据

- 函数定义

```
typedef lchar**(*UGetVMVar)(const uvm* vm, OType type, luint32* count);
```

- 输入参数说明

参数名称	参数说明	备注
vm	虚拟机指针	
type	变量类型	暂时只有DataType_VMTIME的数据会被要求查询

- 输出参数说明

参数名称	参数说明	备注
count	变量个数	返回业务方查到的变量个数

- 返回值

变量的数组，数组内容是变量的路径，例如返回arr[2]={“1”,”2”};

```
typedef enum _OType{
    DataType_CHAR = 0,          /**< 字符型 */
    DataType_SHORT = 32,      /**< 短整形 */
    DataType_INT = 64,          /**< 整形 */
    DataType_FLOAT = 96,      /**< 浮点型 */
    DataType_STRING = 160,    /**< 字符串类型 */
    DataType_DEFAULT = 255,
    DataType_VOID = 6,         //void类型
    DataType_VMTIME = 7,     //时间类型
}OType;//数据类型
```

5.4 getNoticeVar

- 完成功能

  在使用数据外部代理的情况下，通知应用是否需要更新变量data。

- 函数定义

```
typedef char(*UGetNoticeVar)(const char* datasourceid, const uvm* vm, var* data);
```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
vm	虚拟机的地址	
data	虚拟机的变量	包含变量的值，大小，类型

- 返回值

执行结果；成功返回1，失败返回0。

5.5 UAppFrustrum

- 完成功能

执行并行运行的回调

- 函数定义

```
typedef lbool(*UAppFrustrum)(void* pVm, const FunctionInfo** ppArray, lint32 coun
t);
```

- 输入参数说明

参数名称	参数说明	备注
pVm	虚拟机指针	
ppArray	函数的数据数组	
count	函数的数据数组个数	

- 返回值
  返回1成功，0失败

5.6 readBin

- 完成功能

  从bin文件指定的地址读取一定数量的字节,如果bin文件是加密的，虚拟机会在内部进行文件缓存，缓存解密后的bin数据进行处理。

- 函数定义

```
typedef char (*UReadBin)( const char* datasourceid, unsigned int iOffset, unsigne
d int iLen , char* pRetStr);
```

- 输入参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	对应一个bin文件,
iOffset	偏移量	
iLen	长度	

- 输出参数说明

参数名称	参数说明	备注
pRetStr	返回对应数量的字节	

- 返回值

  函数的执行结果，成功返回1，失败返回0;

  l 注解

  1) pRetStr :应用系统只需要将对应的值填充到pRetStr指定的内存中。

5.7 任务相关服务接口

5.7.1 UTaskNew

- 完成功能

  创建任务

- 函数定义

```
typedef char （* UTaskNew）(void *(* vm_task)(void*), void *pArg);
```

- 输入参数说明

参数名称	参数说明	备注
vm_task	任务运行函数	
pArg	运行函数的参数	

- 返回值

  执行结果；成功返回1，失败返回0;

5.8 互斥锁相关服务接口

5.8.1 UMutexNew

- 完成功能

  创建互斥锁

- 函数定义

```
typedef char (*UMutexNew)(Void** pMutex);
```

- 输入参数说明

无

- 返回参数说明

参数名称	参数说明	备注
pMutex	互斥锁	

- 返回值

  执行结果；成功返回1，失败返回0;

5.8.2 UMutexLock

- 完成功能

  加锁互斥锁

- 函数定义

```
typedef char (*UMutexLock)( Void* pMutex);
```

- 参数说明

参数名称	参数说明	备注
pMutex	互斥锁	

- 返回值

  执行结果；成功返回1，失败返回0;

5.8.3 UMutexUnlock

- 完成功能

  解锁互斥锁

- 函数定义

```
typedef char (*UMutexUnlock)( Void* pMutex);
```

- 参数说明

参数名称	参数说明	备注
pMutex	互斥锁	

- 返回值

  执行结果；成功返回1，失败返回0;

5.8.4 UMutexFree

- 完成功能

  释放互斥锁

- 函数定义

```
typedef char (*UMutexFree)( Void** ppMutex);
```

- 参数说明

参数名称	参数说明	备注
ppMutex	互斥锁	

- 返回值

执行结果；成功返回1，失败返回0;

5.9 其他服务接口

5.9.1 UGetHostTime

- 完成功能

  获取主机当前时间

- 函数定义

```
typedef char (*UGetHostTime)(UHostTime* pTime);
```

- 输出参数说明

参数名称	参数说明	备注
pTime	主机当前时间	

UHostTime类型定义如下：

```
typedef struct _UHostTime

{

    char cYear;

    char cMonth;

    char cDay;

    char cHour;

    char cMin;

    char cSec;

    char cWeek;

}UHostTime;
```

字段名称	字段说明	备注
cYear	年份，其值从2000年开始，0表示2000年，以此类推	
cMonth	月份，其值从一月开始，1表示一月，取值范围[1,12]	
cDay	一个月中的日期，取值范围为[1,31]	
cHour	时，取值范围为[0,23]	
cMin	分，取值范围为[0,59]	
cSec	秒，取值范围为[0,59]	
cWeek	星期，取值范围为[1,7]，其中1表示星期一，2表示星期二，以此类推	

- 返回值

  执行结果；成功返回1，失败返回0;

5.9.2 UMalloc

- 完成功能

  动态内存申请

- 函数定义

```
typedef void* (*UMalloc)(unsigned int iSize);
```

- 参数说明

参数名称	参数说明	备注
iSize	需动态申请的空间的大小	

- 返回值

  返回动态内存的指针，失败返回0;

5.9.3 UFree

- 完成功能

  释放动态内存

- 函数定义

```
typedef void (*UFree)(void* pMemory);
```

- 参数说明

参数名称	参数说明	备注
pMemory	待释放的内存指针	

- 返回值

  无

六、应用系统可选实现的服务接口

6.1 tableCBExist

- 完成功能

  外部数据表是否存在

- 函数定义

```
typedef lbool(*TableCBExist)(const char* datasourceid, const char* tablename);
```

- 参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
tablename	数据表名	见5.3

- 返回值

l 执行结果；成功返回1，失败返回0;

6.2 tableCBGet

- 完成功能

  通过外部数据表获取数据

- 函数定义

```
typedef lbool(*TableCBGet)(void* data,const char* datasourceid, const char* table
name, const char* key, char** retvalue, luint32* retvaluelen, luint32 maxlen);
```

- 参数说明

参数名称	参数说明	备注
data	内核传出的数据	
datasourceid	应用的数据源	
tablename	数据表名	见5.3
key	key	
maxlen	最大字符长度	

- 输出参数说明

参数名称	参数说明	备注
retvalue	返回数据内容	
retvaluelen	返回数据大小	

- 返回值

  l 执行结果；成功返回1，失败返回0;

6.3 tableCBSet

- 完成功能

  通过外部数据表更新数据

- 函数定义

```
typedef lbool(*TableCBSet)(const char* datasourceid, const char* tablename, const
 char* key, const char* value, luint32 valuelen);
```

- 参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
tablename	数据表名	见5.3
key	Key	
value	数据内容	
valuelen	数据大小	

- 返回值

  执行结果；成功返回1，失败返回0;

6.4 tableCBFree

- 完成功能

  释放外部数据表的数据缓存

- 函数定义

```
typedef lbool(*TableCBFree)(const char* datasourceid, const char* tablename, char
* value);
```

- 参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
tablename	数据表名	见5.3
value	数据内容	

- 返回值

  l 执行结果；成功返回1，失败返回0;

6.5 logCB

- 完成功能

  日志输出

- 函数定义

```
typedef char(*ULogCB)(const char* datasourceid, const char* pStr, luint32 iLen, l
uint8* eLogPriority);
```

- 参数说明

参数名称	参数说明	备注
datasourceid	应用的数据源	
pStr	日志内容	
iLen	日志字节长度	

- 输出参数说明

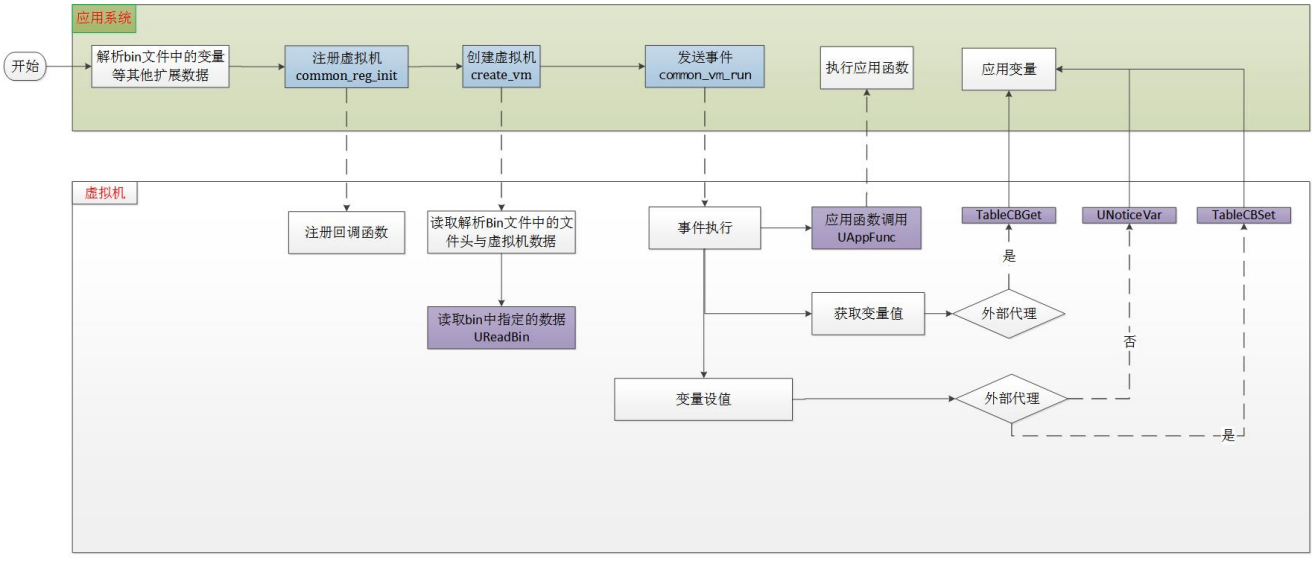
参数名称	参数说明	备注

eLogPriority	日志等级	U_LOG_PRI_TEST = 0,
		U_LOG_PRI_INFO = 1, //调试
		U_LOG_PRI_WARN = 254, // 警告
		U_LOG_PRI_ERROR = 255 // 错误

- 返回值

  无意义

6.6 接口调用流程



备注：蓝色背景的接口为虚拟机提供的主要接口
紫色背景的接口为应用系统需实现的主要回调函数

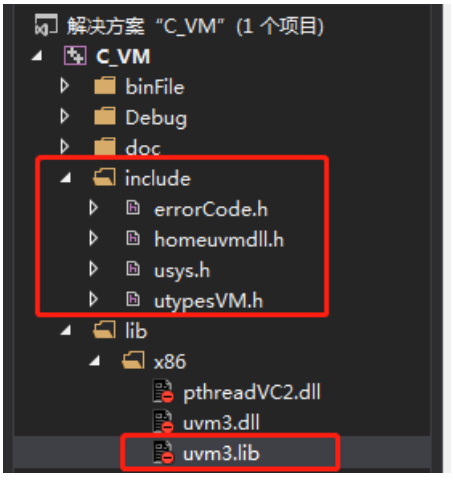
七、附录一

```
//变量表代理

//变量关联表代理

//函数表代理

//逻辑数据表代理（TokenInfo）

//事件索引表

//状态机信息表代理（状态机ID和名称关联表）

//状态机迁移表代理（StatusElement 真假次态，逻辑，真假结果，真假动作，序列号等状态信息）

//键值对表代理(虚拟机全局变量数据)

\#define VARTABLE "VARTABLE"

\#define VARIDTABLE "VARIDTABLE"

\#define FUNCTABLE "FUNCTABLE"
```

八、实践Demo

更新内容	时间	编写人	版本
虚拟机最佳实践	2019年11月8日	李宇哲	基于虚拟机v3.0

1、以window为例，获取到虚拟机的uvm库后，添加到VS工程中，其中虚拟机包括一个include文件和uvm库。



2、根据需要执行的bin文件语句：
(varInt8=-9),(varUint8=0)@varInt16
(https://github.com/varInt16)=1;varInt64=funInt8(varInt8);funInt8(1):varInt16=1;varInt64=funInt8(2)
关联语句如下
(varFloat>1.612340),(varUint8=0)@varDouble
(https://github.com/varDouble)=-4096.99;funFloat(varFloat);funFloat(-1.2221):varDouble=-4096.99;funFloat(-1.3333)

(varDouble<-4096.98),(varUint8=0)@varString (https://github.com/varString)="请把个人设置中的 Full name 设置成中文名";funDouble(-9.0249);funDouble(varDouble):varString="请把个人设置中的 Full name 设置成中文名";funDouble(9.9)

3、虚拟机执行用户语言语句效果

```
[uvm信息][1151]**SETVAR****
[uvm信息][1152]变量KEY:-2.varInt8,变量类型:-2,变量id:0
[uvm信息][1155]数据类型:CHAR,数据值:-9,字节大小:1
[uvm信息][1174]**
[uvm信息][1151]**SETVAR****
[uvm信息][1152]变量KEY:-2.varUint8,变量类型:-2,变量id:9
[uvm信息][1159]数据类型:SHORT,数据值:0,字节大小:2
[uvm信息][1174]**
[uvm信息][1151]**SETVAR****
[uvm信息][1152]变量KEY:-2.varInt16,变量类型:-2,变量id:10
[uvm信息][1159]数据类型:SHORT,数据值:1,字节大小:2
[uvm信息][1174]**
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varInt8,变量ID:0
[uvm信息][1422]数据类型:CHAR,数据值:-9,字节大小:1
[uvm信息][1429]**
[uvm信息][1774]**APPFUNC**
[uvm信息][1778]函数类型:应用函数,函数ID:0,函数KEY:-1.函数funInt8,函数是否异步:同步,函数返参类型:0,实参个数:1
[uvm信息][1783]实参数据类型:CHAR，实参数据值:-9,实参占用字节：1
[uvm信息][2014]函数返回值类型:CHAR,函数返回值:0
[uvm信息][2053]****
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varFloat,变量ID:29
[uvm信息][1398]数据类型:FLOAT,数据值:1.612341,字节大小:4
[uvm信息][1429]**
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varUint8,变量ID:9
[uvm信息][1416]数据类型:SHORT,数据值:0,字节大小:2
[uvm信息][1429]**
[uvm信息][1151]**SETVAR****
[uvm信息][1152]变量KEY:-2.varDouble,变量类型:-2,变量id:30
[uvm信息][1167]数据类型:FLOAT,数据值:-4096.990234,字节大小:4
[uvm信息][1174]**
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varFloat,变量ID:29
[uvm信息][1398]数据类型:FLOAT,数据值:1.612341,字节大小:4
[uvm信息][1429]**
[uvm信息][1774]**APPFUNC**
[uvm信息][1778]函数类型:应用函数,函数ID:8,函数KEY:-1.funFloat,函数是否异步:同步,函数返参类型:96,实参个数:1
[uvm信息][1795]实参数据类型:FLOAT，实参数据值:1.612341,实参占用字节：4
[uvm信息][1981]函数返回值类型:FLOAT,函数返回值:0.000000
```

[uvm信息][2053]****
[uvm信息][1774]**APPFUNC**
[uvm信息][1778]函数类型:应用函数,函数ID:8,函数KEY:-1.funFloat,函数是否异步:同步,函数返参类型:96,实参个数:1
[uvm信息][1795]实参数据类型:FLOAT，实参数据值:-1.222100,实参占用字节：4
[uvm信息][1981]函数返回值类型:FLOAT,函数返回值:0.000000
[uvm信息][2053]****
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varDouble,变量ID:30
[uvm信息][1398]数据类型:FLOAT,数据值:-4096.990234,字节大小:4
[uvm信息][1429]**
[uvm信息][1376]**GETVAR****
[uvm信息][1385]变量KEY:-2.varUint8,变量ID:9
[uvm信息][1416]数据类型:SHORT,数据值:0,字节大小:2
[uvm信息][1429]**
[uvm信息][1151]**SETVAR****
[uvm信息][1152]变量KEY:-2.varString,变量类型:-2,变量id:31
[uvm信息][1172]数据类型:STRING,数据值:请把个人设置中的 Full name 设置成中文名,字节大小:54
[uvm信息][1174]**
[uvm信息][1774]**APPFUNC**
[uvm信息][1778]函数类型:应用函数,函数ID:9,函数KEY:-1.funDouble,函数是否异步:同步,函数返参类型:96,实参个数:1
[uvm信息][1795]实参数据类型:FLOAT，实参数据值:-9.024900,实参占用字节：4
[uvm信息][1981]函数返回值类型:FLOAT,函数返回值:0.000000
[uvm信息][2053]****

4、示例代码重点描述

1>注册应用回调时flag标识要注意，它描述了是否需要使用数据的外部代理，即数据是否虚拟机来保存还是外部应用提供一个区域来保存这些数据。

```
    luint32 flag = 0x0000 | VARTABLEBIT | VARIDTABLEBIT | FUNCTABLEBIT | LOGICTABLEBIT | INDEXSTABLEBIT
        | INFOTABLEBIT | TRANSTABLEBIT | KEYANDVALUETABLEBIT;
//  lbool ret = common_reg_init(g_CommonGlobalModule, &g_tableCB, logout, 1, flag);
    lbool ret = common_reg_init(g_CommonGlobalModule, NULL, logout, 1, 0x00);
```

如果flag不为0x00,那就是会使用外部代理接口，这时候要实现如图所示的接口回调，虚拟机的数据都会通过外部接口去访问和获取。

```
static UTableCB g_tableCB = {
    tableCBExist,
    tableCBGet,
    tableCBSet,
    tableCBFree
};
```

如果flag为0，那就会虚拟机内部会进行数据的存储管理。

2>如果应用方有应用数据是保存在bin文件里面，那么虚拟机会提供接口，来让应用获取保存在bin文件里的数据区。会告诉这片数据区域的首地址和区域大小。通过getAppinfoOffset_ex和getAppData的使用来得到。

```
luint32 size = 0;
luint32 offset = getAppinfoOffset_ex("1", &size);
lchar* buffer = (lchar*)malloc(size);
lbool  retData = getAppData("1", buffer, size);
```

3>创建虚拟机的时候，虚拟机需要和应用提供的标识来绑定，因为这个标识是用来区分应用方在readbin的时候给正确的bin文件或者数据。例如下图的"1"，来标识虚拟机。

```
g_vm = create_vm("1", &errcode);
```

那么在readbin中，通过"1"和down.bin文件就会有个绑定关系。防止一个应用同时执行多个bin文件导致的虚拟机读取数据源错误。

```
char readBin(const char* datasourceid, unsigned int iOffset, unsigned int iLen,
    char* pRetStr){
    if( strcmp(datasourceid,"1") == 0){
        //读取bin文件

        char mapdata[255] = "E:/work/vm3.0/uvm3_binfile/down.bin";
        if (pBinBuf == NULL)
            pBinBuf = platform_read_file_content(mapdata, &psize);

        if ((iOffset + iLen) > psize){
            u_log_ex2(datasourceid, U_LOG_PRI_ERROR, "偏移超出文件范围( %d > %d)", iOffset + iLen, psize);
        }
        //获取偏移数据
        memcpy(pRetStr, pBinBuf + iOffset, iLen);

    }
    return 1;
}
```

3>在执行虚拟机的时候，会提供common_vm_run的接口，入参的数据是强类型关联，所以如果是char数据就调用charToObject，short类型就调用shortToObject来构造入参数据。下图的数据就是(varInt8=-9),(varUint8=0)@varInt16(https://github.com/varInt16)=1;varInt64=funInt8(varInt8);funInt8(1):varInt16=1;varInt64=funInt8(2)中依赖触发的条件(varInt8=-9),(varUint8=0)

```
变量ID:0
变量名称:varInt8
变量全名称:varInt8
变量key:-2.varInt8
对象ID:-2
类型ID:-2
数据类型:int8
值描述:
默认值:-128
事件类型:2
---------------------------
---------------------------
变量ID:9
变量名称:varUint8
变量全名称:varUint8
变量key:-2.varUint8
对象ID:-2
类型ID:-2
数据类型:uint8
值描述:[0,200]
默认值:0
事件类型:2
```

```
var varlist[2] = { 0, 0 };
varlist[0].vmVarPtr = (VarPtr*)malloc(sizeof(VarPtr));
varlist[0].vmVarPtr->path = (lchar*)"-2.varInt8";
varlist[0].vmVarPtr->varid = -1;
varlist[0].vmVarPtr->type = -1;
lint8 data1 = -9;
varlist[0].value = charToObject(data1);

varlist[1].vmVarPtr = (VarPtr*)malloc(sizeof(VarPtr));
varlist[1].vmVarPtr->path = (lchar*)"-2.varUint8";
varlist[1].vmVarPtr->varid = -1;
varlist[1].vmVarPtr->type = -1;
data1 = 0;
varlist[1].value = shortToObject(data1);

lbool retrun1 = common_vm_run(g_vm, varlist, 2, 1, &errcode);
if (varlist[0].vmVarPtr != NULL){
    free(varlist[0].vmVarPtr);
}
objectFree(varlist[0].value);
if (varlist[1].vmVarPtr != NULL){
    free(varlist[1].vmVarPtr);
}
objectFree(varlist[1].value);
```

4>应用回调的实现注意

4.1>appFunc调用应用函数，返回值代表该函数是否执行成功，而应用函数执行后的结果，需要通过object** pRetData传出，这点要注意到。

。

4.2>noticeVar通知应用变量更新，如果type是0,就是虚拟机在更新变量前通知应用方是否准备好更新变量，如果type是1，则表明虚拟机已经更新完变量。如果该回调返回0，则虚拟机会把变量回滚到改变前，不会让变量进行更新。

4.3>getVMVar这个接口是获取时间变量的回调。如果应用方使用了外部数据代理，这个参数需要应用方提供。如果不用外部数据代理，即注册应用回调时flag标识为0，该回调可以为NULL，不实现。

4.4>getNoticeVar这个接口是获取数据时，通知应用方。因为应用方当前的该数据也不一定是最新，有可能应用方要通过这个接口去更新数据，然后才能让虚拟机获取。该回调也是需要用到外部代理的时候才会实现和调用，如果不用外部数据代理，即注册应用回调时flag标识为0，该回调可以为NULL，不实现。

5>代码附录，仅供参考

文档更新时间: 2020-05-28 14:43 作者：赵婧