# ZCRT

平台介绍

AlexCheng

# 功能

- OS封装
- 工具库
- IPC

# OS-BASETYPE

```
typedef  char bool_t;
typedef unsigned char  uint8_t;
typedef unsigned short uint16_t;
typedef unsigned  uint32_t;
typedef long long uint64_t;

typedef signed char  int8_t;
typedef short int16_t;
typedef int   int32_t;
typedef long long int64_t;
typedef char* zstr_t;

typedef void* ZHANDLE_t;

/** callback definine  */
typedef void(*ZCRT_CB)(void* p1, void* p2, void*, void*, void*, void*);
```

# OS-BASE API

- `#include "zcrt/zcrt_os.h"`
- `int32_t  zcrt_atom_inc(int32_t * cntPtr);`
- `int32_t  zcrt_atom_dec(int32_t * cntPtr);`
- `void*    zcrt_malloc(uint32_t size, uint32_t tag);`
- `void     zcrt_free(void* p);`
- `void*    zcrt_realloc(void* p, uint32_t size, uint32_t tag);`
- `uint32_t zcrt_unique_id(void);`

# MUTES/SEM

- `#include "zcrt/zcrt_os.h"`
- `ZHANDLE_t  zcrt_sem_create( char * name, uint16_t initial);`
- `void  zcrt_sem_delete(ZHANDLE_t h);`
- `EZCRTErr zcrt_sem_take(ZHANDLE_t h, uint32_t timeout);`
- `void  zcrt_sem_give(ZHANDLE_t h);`
- `ZHANDLE_t  zcrt_mutex_create( char * name);`
- `void  zcrt_mutex_delete(ZHANDLE_t h);`
- `void  zcrt_mutex_lock(ZHANDLE_t h);`
- `void  zcrt_mutex_unlock(ZHANDLE_t h);`

# THREAD

- ```
  #include "zcrt/zcrt_os.h"
  ```
- ```
  ZHANDLE_t zcrt_thread_create(char *name,
  int32_t ss, int32_t prio, int (f)(void
  *), void *arg);
  ```
- ```
  bool_t zcrt_thread_should_stop(ZHANDLE_t
  h);
  ```
  ```

# TIME API

- `#include "zcrt/zcrt_os.h"`
- `void zcrt_sleep(int32_t msec);`
- `uint32_t zcrt_unique_id(void);`
- `uint32_t zcrt_get_time_freq(void);`
- `void zcrt_get_time64(int64_t* ts);`
- `uint32_t zcrt_start_time(void);`
- `uint32_t zcrt_current_time(void);`
- `uint32_t zcrt_running_clock(void);`
- `uint32_t zcrt_cpu_clock(void);`

# SCHEDULE

- `#include "zcrt/zschedule.h"`

- `ZHANDLE_t zcrt_schedule_create(ZModule_t module, const char* name, EZCRTSchType type, EZCRTPriority pri, uint16_t maxsize);`

- `void zcrt_schedule_delete(ZHANDLE_t sch);`

- `EZCRTErr zcrt_schedule_sendjob(ZHANDLE_t sch, ZCRT_CB job, void* p1, void* p2);`

- `EZCRTErr zcrt_schedule_sendjob_argn(ZHANDLE_t sch, ZCRT_CB job, uint32_t argn, ...);`

# EVENT

- `#include "zcrt/zevent.h"`
- `typedef void(*ZCRT_EVT_CB)(uint32_t evtid, void* evtdata, void* p1, void* p2);`

- `EZCRTErr zcrt_event_register( ZModule_t module, uint8_t priority, uint32_t evtid, ZCRT_EVT_CB cb, void* p1, void* p2 );`
- `void zcrt_event_unregister( ZModule_t module, uint32_t evtid, ZCRT_EVT_CB cb, void* p1, void* p2 );`
- `void zcrt_event_send( ZModule_t module, uint32_t evtid, void* evtdata );`

# TIMER

- `#include "zcrt/ztimer.h"`

- `EZCRTErr zcrt_timer_add(ZModule_t module, uint32_t period, ZCRT_CB cb, void* p1, void* p2 );`

- `EZCRTErr zcrt_timer_add_argn(ZModule_t module, uint32_t period, ZCRT_CB cb, uint32_t argn, ...);`

- `EZCRTErr zcrt_timer_delete(ZModule_t module, ZCRT_CB cb, void* p1, void* p2 );`

- 一次性定时器，如果需要达到循环效果，请在每个定时器结束时，再次调用 **zcrt_timer_add**

# 功能

- OS封装
- 工具库
- IPC

# ARRAY

- `#include "zcrt/zarraylist.h"`

- `ZArrayList zarray_new(uint32_t size, uint32_t unitsize, uint32_t step, uint32_t memtag);`

- `void zarray_delete(ZArrayList v);`
- `void zarray_setlength(ZArrayList v, uint32_t len);`
- `uint32_t zarray_getlength(ZArrayList v);`
- `void* zarray_get(ZArrayList v, uint32_t idx);`
- `#define ZARRAY_APPEND(arr, T, val)`

# HASH

- `#include "zcrt/zhash.h"`

- `ZHash_t zcrt_hash_new(uint32_t size, ZHASH_KEY hash_key, ZHASH_CMP hash_cmp);`
- `void zcrt_hash_delete(ZHash_t hashtable);`
- `void* zcrt_hash_add(ZHash_t hashtable, void* data);`
- `void zcrt_hash_remove(ZHash_t hashtable, void* data);`
- `void* zcrt_hash_lookup(ZHash_t hashtable, void* data);`

  `typedef void (*ZIPC_HASH_CB)(void* data);`

- `void  zcrt_hash_enumerate(ZHash_t hashtable, ZIPC_HASH_CB cb);`
-

# 功能

- OS封装
- 工具库
- IPC

# 定义格式

- require 'zipcpkg.zipctype'
- zipctype.C_SVR_HEAD-->将内容放入到生成文件头部，一般用于头文件的包含等用途
- zipctype.STRUCT_DEF-->定义结构体
- zipctype.API_DEF-->定义接口

# C_SVR_HEAD

```
C_SVR_HEAD([[
#include "test_api_def.h"
]])
```

# 基本类型

- int8_t
- uint8_t
- int16_t
- uint16_t
- int32_t
- uint32_t
- bool_t
- zstr_t
- int64_t
- **不支持uint64_t**

# 扩展类型

- 以E开始的表示枚举值，一般要求在ptenum.lua文件中定义，比如ECardType
- 以T开始的表示结构体，比如TCardStatus
- array表示数组，必须用关键字item注明单元类型：

```
STRUCT_DEF('TSlotInfo'){
    doc='TSlotInfo',
    {'int32_t', 'slot'},
    {'ECardType', 'cardType'},
}


API_DEF('cardmng','QueryCards'){
    doc='mock card',
    {'void*', 'info', args=1},
    {'array', 'slotinfo', item='TSlotInfo', out=true, doc='slotinfo'},
}
```

# 结构体

- format:

```
STRUCT_DEF('结构体名称'){
    doc='描述',
    {'类型', '变量名称'},
    {'类型', '变量名称',default=默认值},
    {'array', '变量名称',item=单元类型},
    decode=true
}
```

- 类型可以是基本类型，结构体，枚举值或者数组
- 基本类型支持default关键字
- 如果为array，必须用item指明类型
- decode为true表示已经存在解析/封装接口，不需要自动生成

# API

- format

```
API_DEF('包名','接口名称'){
    doc='描述',
    ret={'返回值'},
    hide=true/false, --如果为true，表示该API接口只针对内部使用，不需要进行IPC解析，此时下面参数无任何限制，生成时原样输出
    {'参数类型', '参数名', args=n},
    {'参数类型*', '参数名', args=n},
    {'参数类型', '参数名'},
    {'参数类型', '参数名', out=true},
}
```

- 返回值可以为void或基本类型，不建议返回复杂类型
- 参数类型可以是基本类型，枚举，结构体或array
- args表示该接口此参数为注册时的第几个注册参数，平台调用时会自动将该参数传入进来，此时参数类型可以为指针类型
- out表示该参数为输出参数

# EXAMPLE

```
local zipctype=require 'zipcpkg.zipctype'

local STRUCT_DEF=zipctype.STRUCT_DEF
local API_DEF=zipctype.API_DEF
local C_SVR_HEAD=zipctype.C_SVR_HEAD
local API_CB_MOD=zipctype.API_CB_MOD

C_SVR_HEAD([[
#include "ptpinc/ptp_dc_enum.h"
]])

STRUCT_DEF('TSlotInfo'){
    doc='TSlotInfo',
    {'int32_t', 'slot'},
    {'ECardType', 'cardType'},
}


API_DEF('cardmng','QueryCards'){
    doc='mock card',
    {'void*', 'info', args=1},
    {'array', 'slotinfo', item='TSlotInfo', out=true, doc='slotinfo'},
}
```

# 服务端--单实例注册

- 此时API_MOD_CB为false（默认）

```
    ZHANDLE_t ipccomp = ipc_pkg_cardmng_create(module,"cardmng", NULL, (void*)info,
NULL);
    zipc_server_register(module, ipccomp);
```

  - ipc_pkg_cardmng_create为根据文件和包名自动生成，第二个参数表示模块名

# 服务端--多实例规格

- 当需要多个对象提供同一接口时，建议使用该方式注册，比如说每个单板的业务模块，告警模块等。
- example

```
API_CB_MOD('cardsvc',true)
```

```
C_SVR_HEAD([[
#include "ptpinc/ptp_dc_enum.h"
#include "ptmk/ptmk_type.h"
]])


API_DEF('cardsvc','set_hxc_matrix'){
    doc='set whole hxc matrix',
    hide=true,
    ret={'ERet', doc='ret'},
    {'void', 'info', arg=1},
    {'stMatrixLink', 'hxc', doc='hxc'},
    {'stMatrixLink', 'hxc2lxc', doc='hxc2lxc'},
}
```

# 服务端--多实例注册

- example

```
stIPC_cardsvc_cbs cbs;
sprintf(modulename, "s%d.svc", slot);
cbs.set_hxc_matrix = oxc_set_hxc_matrix;

ipccomp = ipc_pkg_cardsvc_create(module,modulename, &cbs, (void*)info, NULL);
zipc_server_register(module, info->ipccomp);
```

- stIPC_cardsvc_cbs为自动生成的回调函数结构体

# 客户端--调用

- ```
  local r,r1,r2=
  ZKIPC:Call(0,'svc','change_svc',v.level,v.src,v.dst,v.prt1,v.prt2,v.prt3,v.prt4)
  ```

- 第一个参数，固定为0

- 第二个参数表示模块名

- 第三个参数表示接口名

- 后面紧跟参数

- 返回值r表示是否调用成功

- r1,r2表示返回值，有多少个返回值就返回多少个
  ```

# 客户端--注册事件

```
local function _msp_ring_evt(evtid,evt)
    local mspnode=dal:GetNode('ne/msp/ring/'..evt.idx)
    if mspnode then
        _SyncMspStatus(mspnode,evt)
    end
end

ZKIPC:reg_event(EEventId.mMspRingStatus,_msp_ring_evt)
```

注销:

```
ZKIPC:unreg_event(EEventId.mMspRingStatus,_msp_ring_evt)
```

# LUA<-->IPC API

- 发送事件

```
ZKIPC::ACall(nil,nil,0,'ne',' send event', EEventId.mCardStatus, {slotid=iu, sub=0,
cardtype=cardtype, status=ECardStatus.online})
```

- 命令行调用IPC接口
  - Call ne _CallIPC {'cardmng','QueryCards'}
  -

# THANKS

AlexCheng