



CV180X & CV181X Framework 通用组件使用手册

Version: 1.0.0

Release date: 2024-05-11

©2022 北京晶视智能科技有限公司
本文件所含信息归北京晶视智能科技有限公司所有。
未经授权，严禁全部或部分复制或披露该等信息。

目录

1	声明	2
2	概述	3
2.1	阅读说明	3
2.2	目录结构	4
3	HAL	5
3.1	概述	5
3.2	功能	6
3.2.1	按键	6
3.2.1.1	概述	6
3.2.1.2	功能开关	7
3.2.1.3	API 参考	7
3.2.1.3.1	CVI_HAL_KEY_Init	8
3.2.1.3.2	CVI_HAL_KEY_Deinit	8
3.2.1.3.3	CVI_HAL_KEY_GetState	9
3.2.1.4	数据类型	10
3.2.1.4.1	CVI_HAL_KEY_IDX_E	10
3.2.1.4.2	CVI_HAL_KEY_STATE_E	11
3.2.1.5	客制化场景修改指南	11
3.2.2	WIFI	13
3.2.2.1	概述	13
3.2.2.2	功能开关	14
3.2.2.3	API 参考	14
3.2.2.3.1	CVI_HAL_WIFI_Init	14
3.2.2.3.2	CVI_HAL_WIFI_Deinit	15
3.2.2.3.3	CVI_HAL_WIFI_Start	15
3.2.2.3.4	CVI_HAL_WIFI_Stop	16
3.2.2.3.5	CVI_HAL_WIFI_GetStartedStatus	17
3.2.2.3.6	CVI_HAL_WIFI_CheckCfgValid	17
3.2.2.4	数据类型	18
3.2.2.4.1	CVI_HAL_WIFI_MODE_E	18
3.2.2.4.2	CVI_HAL_WIFI_STA_MODE_E	19
3.2.2.4.3	CVI_HAL_WIFI_COMMON_CFG_S	20
3.2.2.4.4	CVI_HAL_WIFI_APMODE_CFG_S	20
3.2.2.4.5	CVI_HAL_WIFI_STAMODE_COMMON_CFG_S	21
3.2.2.4.6	CVI_HAL_WIFI_STAMODE_SENIOR_CFG_S	21
3.2.2.4.7	CVI_HAL_WIFI_STAMODE_CFG_S	22
3.2.2.4.8	CVI_HAL_WIFI_CFG_S	22
3.2.2.4.9	CVI_HAL_WIFI_SCAN_RESULT_S	23
3.2.2.4.10	CVI_HAL_WIFI_SCAN_RESULTSET_S	23

3.2.2.5	客制化场景修改指南	24
3.2.3	触摸屏	25
3.2.3.1	概述	25
3.2.3.2	功能开关	26
3.2.3.3	API 参考	27
3.2.3.3.1	CVI_HAL_TOUCHPAD_Init	28
3.2.3.3.2	CVI_HAL_TOUCHPAD_Deinit	28
3.2.3.3.3	CVI_HAL_TOUCHPAD_Suspend	29
3.2.3.3.4	CVI_HAL_TOUCHPAD_Resume	30
3.2.3.3.5	CVI_HAL_TOUCHPAD_Start	30
3.2.3.3.6	CVI_HAL_TOUCHPAD_Stop	31
3.2.3.3.7	CVI_HAL_TOUCHPAD_ReadInputEvent	32
3.2.3.4	数据类型	32
3.2.3.4.1	CVI_HAL_TOUCHPAD_INPUTINFO_S	32
3.2.3.5	客制化场景修改指南	33
3.2.4	显示屏幕	34
3.2.4.1	概述	34
3.2.4.2	功能开关	35
3.2.4.3	API 参考	35
3.2.4.3.1	CVI_HAL_SCREEN_Init	36
3.2.4.3.2	CVI_HAL_SCREEN_Deinit	37
3.2.4.3.3	CVI_HAL_SCREEN_Register	37
3.2.4.3.4	CVI_HAL_SCREEN_GetAttr	38
3.2.4.3.5	CVI_HAL_SCREEN_GetDisplayState	39
3.2.4.3.6	CVI_HAL_SCREEN_SetDisplayState	39
3.2.4.3.7	CVI_HAL_SCREEN_GetBackLightState	40
3.2.4.3.8	CVI_HAL_SCREEN_SetBackLightState	41
3.2.4.3.9	CVI_HAL_SCREEN_GetLuma	42
3.2.4.3.10	CVI_HAL_SCREEN_SetLuma	42
3.2.4.3.11	CVI_HAL_SCREEN_GetSaturation	43
3.2.4.3.12	CVI_HAL_SCREEN_SetSaturation	44
3.2.4.3.13	CVI_HAL_SCREEN_GetContrast	44
3.2.4.3.14	CVI_HAL_SCREEN_SetContrast	45
3.2.4.4	数据类型	46
3.2.4.4.1	CVI_HAL_SCREEN_IDX_E	47
3.2.4.4.2	CVI_HAL_SCREEN_TYPE_E	47
3.2.4.4.3	CVI_HAL_SCREEN_LCD_INTF_TYPE_E	48
3.2.4.4.4	CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_E	48
3.2.4.4.5	CVI_HAL_SCREEN_MIPI_VIDEO_MODE_E	49
3.2.4.4.6	CVI_HAL_SCREEN_MIPI_VIDEO_FORMAT_E	50
3.2.4.4.7	CVI_HAL_SCREEN_SYNC_ATTR_S	50
3.2.4.4.8	CVI_HAL_SCREEN_CLK_TYPE_E	51
3.2.4.4.9	CVI_HAL_SCREEN_CLK_PLL_S	52
3.2.4.4.10	CVI_HAL_SCREEN_CLK_ATTR_S	52
3.2.4.4.11	CVI_HAL_SCREEN_COMMON_ATTR_S	53
3.2.4.4.12	CVI_HAL_SCREEN_MIPI_ATTR_S	54
3.2.4.4.13	CVI_HAL_SCREEN_LCD_ATTR_S	55
3.2.4.4.14	CVI_HAL_SCREEN_ATTR_S	55
3.2.4.4.15	CVI_HAL_SCREEN_PWM_S	56
3.2.4.4.16	CVI_HAL_SCREEN_STATE_E	56

3.2.4.4.17	CVI_HAL_SCREEN_OBJ_S	57
3.2.4.5	客制化场景修改指南	58
3.2.5	重力加速度计	59
3.2.5.1	概述	59
3.2.5.2	功能开关	60
3.2.5.3	API 参考	61
3.2.5.3.1	CVI_HAL_GSENSOR_Init	62
3.2.5.3.2	CVI_HAL_GSENSOR_DeInit	62
3.2.5.3.3	CVI_HAL_GSENSOR_SetSensitivity	63
3.2.5.3.4	CVI_HAL_GSENSOR_SetAttr	64
3.2.5.3.5	CVI_HAL_GSENSOR_GetCurValue	64
3.2.5.3.6	CVI_HAL_GSENSOR_GetCollisionStatus	65
3.2.5.3.7	CVI_HAL_GSENSOR_Register	66
3.2.5.3.8	CVI_HAL_GSENSOR_ReadInterrupt	66
3.2.5.3.9	CVI_HAL_GSENSOR_OpenInterrupt	67
3.2.5.4	数据类型	68
3.2.5.4.1	CVI_HAL_GSENSOR_SENSITIVITY_E	68
3.2.5.4.2	CVI_HAL_GSENSOR_VALUE_S	69
3.2.5.4.3	CVI_HAL_GSENSOR_ATTR_S	69
3.2.5.4.4	CVI_HAL_GSENSOR_CFG_S	70
3.2.5.4.5	CVI_HAL_GSENSOR_OBJ_S	70
3.2.5.5	客制化场景修改指南	71
3.2.6	GPS	72
3.2.6.1	概述	72
3.2.6.2	功能开关	72
3.2.6.3	API 参考	72
3.2.6.3.1	CVI_HAL_GPS_Init	73
3.2.6.3.2	CVI_HAL_GPS_Deinit	73
3.2.6.3.3	CVI_HAL_GPS_GetRawData	74
3.2.6.4	数据类型	75
3.2.6.4.1	CVI_GPSDATA_S	75
3.2.6.5	客制化场景修改指南	75
3.2.7	LED	75
3.2.7.1	概述	75
3.2.7.2	功能开关	76
3.2.7.3	API 参考	77
3.2.7.3.1	CVI_HAL_LED_Init	78
3.2.7.3.2	CVI_HAL_LED_GetState	78
3.2.7.3.3	CVI_HAL_LED_Deinit	79
3.2.7.3.4	CVI_HAL_LED_SetValue	80
3.2.7.4	数据类型	80
3.2.7.4.1	CVI_HAL_LED_STATE_E	80
3.2.7.4.2	CVI_HAL_LED_IDX_E	81
3.2.7.4.3	CVI_LED_GPIO_ID_E	82
3.2.7.5	客制化场景修改指南	82
3.2.8	看门狗	83
3.2.8.1	概述	83
3.2.8.2	功能开关	83
3.2.8.3	API 参考	84
3.2.8.3.1	CVI_HAL_WATCHDOG_Init	85

3.2.8.3.2	CVI_HAL_WATCHDOG_Deinit	85
3.2.8.3.3	CVI_HAL_WATCHDOG_Feed	86
3.2.8.4	数据类型	87
3.2.8.5	客制化场景修改指南	87
3.2.9	电量计	87
3.2.9.1	概述	87
3.2.9.2	功能开关	88
3.2.9.3	API 参考	88
3.2.9.3.1	CVI_HAL_ADC_Init	89
3.2.9.3.2	CVI_HAL_ADC_Deinit	89
3.2.9.3.3	CVI_HAL_ADC_GetValue	90
3.2.9.4	数据类型	91
3.2.9.5	客制化场景修改指南	91
3.2.10	GPIO && PWM	91
3.2.10.1	概述	91
3.2.10.2	功能开关	91
3.2.10.3	API 参考	91
3.2.10.3.1	CVI_GPIO_Export	91
3.2.10.3.2	CVI_GPIO_Unexport	92
3.2.10.3.3	CVI_GPIO_Direction_Input	93
3.2.10.3.4	CVI_GPIO_Direction_Output	93
3.2.10.3.5	CVI_GPIO_Set_Value	94
3.2.10.3.6	CVI_GPIO_Get_Value	95
3.2.10.3.7	CVI_GPIO_Poll	95
3.2.10.3.8	CVI_PWM_Init	96
3.2.10.3.9	CVI_PWM_Deinit	97
3.2.10.3.10	CVI_PWM_Get_Percent	98
3.2.10.3.11	CVI_PWM_Set_Percent	98
3.2.10.3.12	CVI_PWM_Set_Param	99
3.2.10.4	数据类型	100
3.2.10.4.1	CVI_GPIO_NUM_E	100
3.2.10.4.2	CVI_GPIO_EDGE_E	101
3.2.10.4.3	CVI_GPIO_VALUE_E	102
3.2.10.4.4	CVI_HAL_PWM_S	102
3.2.10.5	客制化场景修改指南	103
4	OSAL	104
4.1	概述	104
4.2	API 参考	104
4.2.1	cvi_osal_get_boot_time_us	105
4.2.2	cvi_osal_get_boot_time_ms	105
4.2.3	cvi_osal_get_boot_time_ns	106
4.2.4	cvi_osal_mutex_create	107
4.2.5	cvi_osal_mutex_destroy	107
4.2.6	cvi_osal_mutex_lock	108
4.2.7	cvi_osal_mutex_unlock	109
4.2.8	cvi_osal_task_create	110
4.2.9	cvi_osal_task_destroy	110
4.2.10	cvi_osal_task_join	111
4.2.11	cvi_osal_task_sleep	112

4.2.12	cvi_osal_task_resched	112
4.2.13	cvi_osal_cond_create	113
4.2.14	cvi_osal_cond_destroy	114
4.2.15	cvi_osal_cond_signal	114
4.2.16	cvi_osal_cond_wait	115
4.2.17	cvi_osal_cond_timedwait	116
4.2.18	cvi_osal_sem_create	116
4.2.19	cvi_osal_sem_destroy	117
4.2.20	cvi_osal_sem_wait	118
4.2.21	cvi_osal_sem_post	118
4.3	数据类型	119
4.3.1	cvi_osal_mutex_attr_t	119
4.3.2	cvi_osal_mutex_t / *cvi_osal_mutex_handle_t	120
4.3.3	cvi_osal_task_attr_t	120
4.3.4	cvi_osal_task_t / *cvi_osal_task_handle_t	121
4.3.5	cvi_osal_task_entry_t	122
4.3.6	cvi_osal_cond_attr_t	122
4.3.7	cvi_osal_cond_t / *cvi_osal_cond_handle_t	123
4.3.8	cvi_osal_sem_attr_t	123
4.3.9	cvi_osal_sem_t / *cvi_osal_sem_handle_t	124
5	COMMON	125
5.1	HFSM	125
5.1.1	概述	125
5.1.2	架构原理	125
5.1.2.1	状态机模型	127
5.1.2.2	状态机切换规则	127
5.1.3	API 参考	127
5.1.3.1	CVI_HFSM_Create	128
5.1.3.2	CVI_HFSM_Destroy	128
5.1.3.3	CVI_HFSM_AddState	129
5.1.3.4	CVI_HFSM_SetInitialState	130
5.1.3.5	CVI_HFSM_GetCurrentState	130
5.1.3.6	CVI_HFSM_Start	131
5.1.3.7	CVI_HFSM_Stop	132
5.1.3.8	CVI_HFSM_SendAsyncMessage	132
5.1.4	数据类型	133
5.1.4.1	CVI_STATE_NAME_LEN	133
5.1.4.2	CVI_STATE_MAX_AMOUNT	134
5.1.4.3	CVI_PROCESS_MSG_RESULTE_OK	134
5.1.4.4	CVI_PROCESS_MSG_UNHANDLER	134
5.1.4.5	CVI_STATE_S	135
5.1.4.6	CVI_HFSM_EVENT_E	136
5.1.4.7	CVI_HFSM_EVENT_INFO_S	136
5.1.4.8	CVI_HFSM_EVENT_CALLBACK	137
5.1.4.9	CVI_HFSM_ATTR_S	137
5.1.4.10	CVI_MESSAGE_S	138
5.2	EVENTHUB	138
5.2.1	概述	138
5.2.2	架构原理	139

5.2.3	API 参考	140
5.2.3.1	CVI_EVTHUB_Init	140
5.2.3.2	CVI_EVTHUB_Deinit	141
5.2.3.3	CVI_EVTHUB_Register	141
5.2.3.4	CVI_EVTHUB_UnRegister	142
5.2.3.5	CVI_EVTHUB_Publish	143
5.2.3.6	CVI_EVTHUB_CreateSubscriber	143
5.2.3.7	CVI_EVTHUB_DestroySubscriber	144
5.2.3.8	CVI_EVTHUB_Subscribe	145
5.2.3.9	CVI_EVTHUB_UnSubscribe	145
5.2.3.10	CVI_EVTHUB_GetEventHistory	146
5.2.4	数据类型	147
5.2.4.1	MSG_PAYLOAD_LEN	147
5.2.4.2	CVI_EVENTHUB_SUBSCRIBE_NAME_LEN	147
5.2.4.3	CVI_TOPIC_ID	148
5.2.4.4	CVI_EVENT_S	148
5.2.4.5	CVI_EVENTHUB_SUBSCRIBER_S	149
5.3	MQ	149
5.3.1	概述	149
5.3.2	API 参考	149
5.3.2.1	CVI_MQ_CreateEndpoint	150
5.3.2.2	CVI_MQ_DestroyEndpoint	150
5.3.2.3	CVI_MQ_SendRAW	151
5.3.2.4	CVI_MQ_Send	152
5.3.2.5	CVI_MQ_SendAck	153
5.3.3	数据类型	153
5.3.3.1	CVI_MQ_MSG_S	154
5.3.3.2	CVI_MQ_ENDPOINT_S	154
5.3.3.3	CVI_MQ_ENDPOINT_CONFIG_S	155
5.4	DTCF	156
5.4.1	概述	156
5.4.2	架构原理	156
5.4.3	API 参考	158
5.4.3.1	CVI_DTCF_Init	158
5.4.3.2	CVI_DTCF_GetDirNames	159
5.4.3.3	CVI_DTCF_DeInit	160
5.4.3.4	CVI_DTCF_Scan	160
5.4.3.5	CVI_DTCF_GetFileByIndex	161
5.4.3.6	CVI_DTCF_DelFileByIndex	162
5.4.3.7	CVI_DTCF_AddFile	162
5.4.3.8	CVI_DTCF_GetOldestFileIndex	163
5.4.3.9	CVI_DTCF_GetFileAmount	164
5.4.3.10	CVI_DTCF_GetOldestFilePath	164
5.4.3.11	CVI_DTCF_CreateFilePath	165
5.4.3.12	CVI_DTCF_GetRelatedFilePath	166
5.4.3.13	CVI_DTCF_GetEmrFilePath	167
5.4.3.14	CVI_DTCF_GetFileDirType	167
5.4.4	数据类型	168
5.4.4.1	CVI_FILE_PATH_LEN_MAX	168
5.4.4.2	CVI_DIR_LEN_MAX	169

	5.4.4.3	CVI_DTCF_DIR_E	169
	5.4.4.4	CVI_DTCF_FILE_TYPE_E	170
5.5	EXIF		171
5.5.1	概述		171
5.5.2	API 参考		171
5.5.2.1	CVI_EXIF_MakeExifFile		171
5.5.2.2	CVI_EXIF_MakeExifParam		172
5.5.2.3	CVI_EXIF_MakeNewSatJpgFromBuf		173
5.5.3	数据类型		173
5.5.3.1	CVI_EXIF_FILE_INFO_S		174
5.6	LOG		176
5.6.1	概述		176
5.6.2	API 参考		176
5.6.2.1	CVI_LOG_INIT		176
5.6.2.2	CVI_LOG_SET_LEVEL		177
5.6.2.3	CVI_LOG_SET_TAG		177
5.6.3	数据类型		178
5.7	QUEUE		178
5.7.1	概述		178
5.7.2	API 参考		178
5.7.2.1	CVI_QUEUE_Create		179
5.7.2.2	CVI_QUEUE_Destroy		179
5.7.2.3	CVI_QUEUE_Clear		180
5.7.2.4	CVI_QUEUE_GetLen		180
5.7.2.5	CVI_QUEUE_Push		181
5.7.2.6	CVI_QUEUE_Pop		182
5.7.3	数据类型		182
5.7.3.1	CVI_QUEUE_HANDLE_T		183
5.8	SIGNAL_SLOT		183
5.8.1	概述		183
5.8.2	API 参考		183
5.8.2.1	CVI_SIGNAL_InitByType		184
5.8.2.2	CVI_SIGNAL_Init		184
5.8.2.3	CVI_SLOT_Init		185
5.8.2.4	CVI_SIGNAL_Connect		186
5.8.2.5	CVI_SIGNAL_Emit		187
5.8.2.6	CVI_SIGNAL_Disconnect		187
5.8.2.7	CVI_SIGNAL_Deinit		188
5.8.2.8	CVI_SLOT_Deinit		189
5.8.3	数据类型		189
5.8.3.1	CVI_SIGNAL_SLOT_TYPE_E		189
5.8.3.2	CVI_SIGNAL_S		190
5.8.3.3	CVI_SLOT_S		191
5.9	STORAGE		191
5.9.1	概述		191
5.9.2	API 参考		192
5.9.2.1	CVI_STG_Init		193
5.9.2.2	CVI_STG_DeInit		193
5.9.2.3	CVI_STG_GetFsInfo		194
5.9.2.4	CVI_STG_GetSDInfo		195

5.9.2.5	CVI_STG_GetDevState	195
5.9.2.6	CVI_STG_Umount	196
5.9.2.7	CVI_STG_Mount	197
5.9.2.8	CVI_STG_Format	197
5.9.2.9	CVI_STG_FormatWithLabel	198
5.9.2.10	CVI_STG_GetInfo	199
5.9.2.11	CVI_STG_RepairFAT32	199
5.9.2.12	CVI_STG_DetectPartition	200
5.9.2.13	CVI_STG_TestPartition	201
5.9.3	数据类型	201
5.9.3.1	CVI_STG_FS_INFO_S	202
5.9.3.2	CVI_STG_DEV_STATE_E	202
5.9.3.3	CVI_STG_FS_TYPE_E	203
5.9.3.4	CVI_STG_STATE_E	203
5.9.3.5	CVI_STG_DEV_INFO_S	204
5.9.3.6	CVI_STG_TRANSMISSION_SPEED_E	205
5.9.3.7	STG_DEVINFO_S	206
5.10	SYSUTILS	207
5.10.1	概述	207
5.10.2	API 参考	207
5.10.2.1	cvi_insmode	207
5.10.2.2	cvi_rmmod	208
5.10.2.3	cvi_PathIsDirectory	209
5.10.2.4	cvi_rmdir	209
5.10.2.5	cvi_mkdir	210
5.10.2.6	cvi_system	211
5.10.2.7	cvi_usleep	211
5.10.2.8	cvi_du	212
5.10.2.9	cvi_async	213
5.10.3	数据类型	213
5.10.3.1	CVI_MAX_PATH_LEN	213
5.11	TIMER	214
5.11.1	概述	214
5.11.2	API 参考	214
5.11.2.1	CVI_Timer_Init	214
5.11.2.2	CVI_Timer_DeInit	215
5.11.2.3	CVI_Timer_Create	216
5.11.2.4	CVI_Timer_Reset	216
5.11.2.5	CVI_Timer_SetTickValue	217
5.11.2.6	CVI_Timer_Destroy	218
5.11.2.7	CVI_Timer_CleanUp	219
5.11.2.8	CVI_Timer_SetPeriodicAttr	219
5.11.2.9	CVI_Timer_GetPastTime	220
5.11.3	数据类型	221
5.11.3.1	CVI_TIMER_S	221
5.11.3.2	CVI_TIMER_HANDLE_T	222
5.11.3.3	CVI_TIMER_PROC_CALLBACK	222
5.12	UPGRADE	222
5.12.1	概述	222
5.12.2	API 参考	223

5.12.2.1	CVI_UPGRADE_Init	223
5.12.2.2	CVI_UPGRADE_Deinit	224
5.12.2.3	CVI_UPGRADE_CheckPkg	224
5.12.2.4	CVI_UPGRADE_DoUpgrade	225
5.12.2.5	CVI_UPGRADE_DoUpgradeViaSD	226
5.12.2.6	CVI_UPGRADE_RegisterEvent	226
5.12.3	数据类型	227
5.12.3.1	CVI_UPGRADE_PKG_HEAD_S	227
5.12.3.2	CVI_UPGRADE_EVENT_S	228
5.12.3.3	CVI_UPGRADE_DEV_INFO_S	229
6	录像模块	230
6.1	概述	230
6.2	基本概念	231
6.3	组件 API	231
6.3.1	Mem_alloc	232
6.3.1.1	API 参考	232
6.3.1.1.1	CVI_MEM_Allocate	232
6.3.1.1.2	CVI_MEM_Free	233
6.3.1.1.3	CVI_MEM_AllocateVb	233
6.3.1.1.4	CVI_MEM_VbFree	234
6.3.2	Muxer	235
6.3.2.1	API 参考	235
6.3.2.1.1	CVI_MUXER_Create	235
6.3.2.1.2	CVI_MUXER_Start	236
6.3.2.1.3	CVI_MUXER_WritePacket	237
6.3.2.1.4	CVI_MUXER_Stop	237
6.3.2.1.5	CVI_MUXER_Destroy	238
6.3.2.1.6	CVI_MUXER_FlushPackets	239
6.3.2.2	数据类型	239
6.3.2.2.1	CVI_MUXER_TRACK_VIDEO_CODEC_E	240
6.3.2.2.2	CVI_MUXER_TRACK_AUDIO_CODEC_E	240
6.3.2.2.3	CVI_MUXER_CODEC_VIDEO_S	241
6.3.2.2.4	CVI_MUXER_CODEC_AUDIO_S	242
6.3.2.2.5	CVI_MUXER_CODEC_SUBTITLE_S	242
6.3.2.2.6	CVI_MUXER_CODEC_THUMBNAIL_S	243
6.3.2.2.7	CVI_MUXER_EVENT_E	243
6.3.2.2.8	CVI_MUXER_FRAME_TYPE_E	244
6.3.2.2.9	CVI_MUXER_FRAME_INFO_S	245
6.3.2.2.10	CVI_MUXER_EVENT_CALLBACK	245
6.3.2.2.11	CVI_MUXER_ATTR_S	246
6.3.3	Recorder	247
6.3.3.1	API 参考	247
6.3.3.1.1	CVI_RECORDER_SendFrame	247
6.3.3.1.2	CVI_RECORDER_Start_NormalRec	248
6.3.3.1.3	CVI_RECORDER_Stop_NormalRec	249
6.3.3.1.4	CVI_RECORDER_Start_EventRec	249
6.3.3.1.5	CVI_RECORDER_Stop_EventRecPost	250
6.3.3.1.6	CVI_RECORDER_ForceStop_EventRec	251
6.3.3.1.7	CVI_RECORDER_Start_LapseRec	252

6.3.3.1.8	CVI_RECORDER_Stop_LapseRec	252
6.3.3.1.9	CVI_RECORDER_Destroy	253
6.3.3.1.10	CVI_RECORDER_Create	254
6.3.3.1.11	CVI_RECORDER_Split	254
6.3.3.1.12	CVI_RECORDER_Timelapse_Is_SendVenc	255
6.3.3.1.13	CVI_RECORDER_GetUs	256
6.3.3.2	数据类型	257
6.3.3.2.1	CVI_FRAME_STREAM_SEGMENT_MAX_NUM	258
6.3.3.2.2	CVI_RECORDER_TRACK_MAX_CNT	258
6.3.3.2.3	CVI_RECORDER_EVENT_E	258
6.3.3.2.4	CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT_S	259
6.3.3.2.5	CVI_RECORDER_GET_FILENAME_CALLBACK	260
6.3.3.2.6	CVI_RECORDER_EVENT_CALLBACK	261
6.3.3.2.7	CVI_RECORDER_GET_SUBTITLE_CALLBACK	261
6.3.3.2.8	CVI_RECORDER_GET_MEM_BUFFER_STOP_CALLBACK	262
6.3.3.2.9	CVI_RECORDER_REQUEST_IDR_CALLBACK	262
6.3.3.2.10	CVI_RECORDER_STOP_CALLBACK	263
6.3.3.2.11	CVI_RECORDER_GET_DIR_TYPE_CALLBACK	263
6.3.3.2.12	CVI_RECORDER_TYPE_INDEX_E	264
6.3.3.2.13	CVI_RECORDER_CALLBACK_TYPE_E	264
6.3.3.2.14	CVI_RECORDER_CB_HANDLES_S	265
6.3.3.2.15	CVI_RECORDER_RBUF_TYPE_E	266
6.3.3.2.16	CVI_RECORDER_RBUF_ATTR_S	266
6.3.3.2.17	CVI_RECORDER_TRACK_SOURCE_TYPE_E	267
6.3.3.2.18	CVI_RECORDER_TYPE_E	267
6.3.3.2.19	CVI_RECORDER_SPLIT_TYPE_E	268
6.3.3.2.20	CVI_RECORDER_SPLIT_ATTR_S	268
6.3.3.2.21	CVI_RECORDER_NORMAL_ATTR_S	269
6.3.3.2.22	CVI_RECORDER_LAPSE_ATTR_S	269
6.3.3.2.23	CVI_RECORDER_TRACK_VideoSourceInfo_S	270
6.3.3.2.24	CVI_RECORDER_TRACK_AudioSourceInfo_S	271
6.3.3.2.25	CVI_RECORDER_TRACK_PrivateSourceInfo_S	271
6.3.3.2.26	CVI_RECORDER_TRACK_SOURCE_S	272
6.3.3.2.27	CVI_RECORDER_STREAM_ATTR_S	273
6.3.3.2.28	CVI_RECORDER_ATTR_S	273
6.3.3.2.29	CVI_RECORDER_FRAME_STREAM_S	274
6.3.4	Ringbuffer	275
6.3.4.1	API 参考	275
6.3.4.1.1	CVI_RBUF_Init	276
6.3.4.1.2	CVI_RBUF_DeInit	276
6.3.4.1.3	CVI_RBUF_DataCnt	277
6.3.4.1.4	CVI_RBUF_Unused	278
6.3.4.1.5	CVI_RBUF_Copy_In	279
6.3.4.1.6	CVI_RBUF_Copy_Out	279
6.3.4.1.7	CVI_RBUF_Copy_OutTmp	280
6.3.4.1.8	CVI_RBUF_Refresh_In	281
6.3.4.1.9	CVI_RBUF_Refresh_Out	282
6.3.4.1.10	CVI_RBUF_Refresh_OutTmp	282
6.3.4.1.11	CVI_RBUF_ShowLog	283
6.3.4.1.12	CVI_RBUF_Reset	284

6.3.4.1.13	CVI_RBUF_Get_InSize	284
6.3.4.1.14	CVI_RBUF_Get_Totalsize	285
6.3.4.2	数据类型	286
6.3.4.2.1	RINGBUF_OUTPTR_CNT	286
6.3.4.2.2	CVI_RBUF_INFO_S	286
6.3.5	Record_Service	287
6.3.5.1	API 参考	287
6.3.5.1.1	CVI_RECORD_SERVICE_Create	288
6.3.5.1.2	CVI_RECORD_SERVICE_Destroy	289
6.3.5.1.3	CVI_RECORD_SERVICE_UpdateParam	289
6.3.5.1.4	CVI_RECORD_SERVICE_StartRecord	290
6.3.5.1.5	CVI_RECORD_SERVICE_StopRecord	291
6.3.5.1.6	CVI_RECORD_SERVICE_StartTimelapseRecord	291
6.3.5.1.7	CVI_RECORD_SERVICE_StopTimelapseRecord	292
6.3.5.1.8	CVI_RECORD_SERVICE_EventRecord	293
6.3.5.1.9	CVI_RECORD_SERVICE_StopEventRecord	294
6.3.5.1.10	CVI_RECORD_SERVICE_StartMute	294
6.3.5.1.11	CVI_RECORD_SERVICE_StopMute	295
6.3.5.1.12	CVI_RECORD_SERVICE_PivCapture	296
6.3.5.1.13	CVI_RECORD_SERVICE_WaitPivFinish	297
6.3.5.2	数据类型	297
6.3.5.2.1	CVI_RECORD_SERVICE_HANDLE_T	298
6.3.5.2.2	CVI_RECORD_SERVICE_FILE_TYPE_E	298
6.3.5.2.3	CVI_RECORD_SERVICE_VIDEO_CODEC_E	299
6.3.5.2.4	CVI_RECORD_SERVICE_AUDIO_CODEC_E	299
6.3.5.2.5	CVI_RECORD_SERVICE_VENC_BIND_MODE_E	300
6.3.5.2.6	CVI_RECORD_SERVICE_PARAM_S	300
6.3.5.2.7	CVI_RECORD_SERVICE_GPS_RMCINFO_S	303
6.3.5.2.8	CVI_RECORD_SERVICE_GPS_INFO_S	304
6.3.6	Audio_Service	304
6.3.6.1	API 参考	304
6.3.6.1.1	CVI_AUDIO_SERVICR_ACAP_CallbackSet	305
6.3.6.1.2	CVI_AUDIO_SERVICR_ACAP_CallbackUnset	305
6.3.6.1.3	CVI_AUDIO_SERVICR_ACAP_AacCallbackSet	306
6.3.6.1.4	CVI_AUDIO_SERVICR_ACAP_AacCallbackUnset	307
6.3.6.1.5	CVI_AUDIO_SERVICR_ACAP_TaskStart	308
6.3.6.1.6	CVI_AUDIO_SERVICR_ACAP_TaskStop	308
6.3.6.2	数据类型	309
6.3.6.2.1	CVI_AUDIO_SERVICR_ACAP_GET_FRAME_CALLBACK	309
6.3.6.2.2	CVI_AUDIO_SERVICR_ACAP_GET_AAC_FRAME_CALLBACK	309
7	播放器模块	311
7.1	概述	311
7.2	播放器模块处理流程	312
7.2.1	获取媒体信息分析	312
7.2.2	解码分析	312
7.3	播放器状态图	314
7.4	组件 API	314
7.4.1	Demuxer	315
7.4.1.1	基类 Demuxer	315

7.4.1.2	派生类 FFmpegDemuxer	316
7.4.1.3	API 参考	317
7.4.1.3.1	CVI_DEMUXER_Create	317
7.4.1.3.2	CVI_DEMUXER_Destroy	318
7.4.1.3.3	CVI_DEMUXER_Open	319
7.4.1.3.4	CVI_DEMUXER_Close	319
7.4.1.3.5	CVI_DEMUXER_Pause	320
7.4.1.3.6	CVI_DEMUXER_Resume	321
7.4.1.3.7	CVI_DEMUXER_SetInput	322
7.4.1.3.8	CVI_DEMUXER_Read	322
7.4.1.3.9	CVI_DEMUXER_Seek	323
7.4.1.3.10	CVI_DEMUXER_GetMediaInfo	324
7.4.1.4	数据类型	325
7.4.1.4.1	CVI_DEMUXER_HANDLE_T	325
7.4.1.4.2	CVI_DEMUXER_STREAM_RESOLUTION_S	325
7.4.1.4.3	CVI_DEMUXER_STREAM_INFO_S	326
7.4.1.4.4	CVI_DEMUXER_PACKET_S	327
7.4.1.4.5	CVI_DEMUXER_MEDIA_INFO_S	327
7.4.2	Thumbnail_extractor	329
7.4.2.1	API 参考	329
7.4.2.1.1	CVI_THUMBNAI_L_EXTRACTOR_Create	329
7.4.2.1.2	CVI_THUMBNAI_L_EXTRACTOR_Destroy	330
7.4.2.1.3	CVI_THUMBNAI_L_EXTRACTOR_GetThumbnail	331
7.4.2.1.4	CVI_THUMBNAI_L_EXTRACTOR_ClearPacket	331
7.4.2.2	数据类型	332
7.4.2.2.1	CVI_THUMBNAI_L_EXTRACTOR_HANDLE_T	332
7.4.2.2.2	CVI_THUMBNAI_L_PACKET_S	333
7.4.3	Player	333
7.4.3.1	目录结构	333
7.4.3.2	API 参考	334
7.4.3.2.1	CVI_PLAYER_Init	335
7.4.3.2.2	CVI_PLAYER_Deinit	336
7.4.3.2.3	CVI_PLAYER_Create	336
7.4.3.2.4	CVI_PLAYER_Destroy	337
7.4.3.2.5	CVI_PLAYER_SetDataSource	338
7.4.3.2.6	CVI_PLAYER_GetDataSource	339
7.4.3.2.7	CVI_PLAYER_LightOpen	339
7.4.3.2.8	CVI_PLAYER_Play	340
7.4.3.2.9	CVI_PLAYER_Stop	341
7.4.3.2.10	CVI_PLAYER_Pause	341
7.4.3.2.11	CVI_PLAYER_Resume	342
7.4.3.2.12	CVI_PLAYER_Seek	343
7.4.3.2.13	CVI_PLAYER_TPlay	344
7.4.3.2.14	CVI_PLAYER_SetAudioParameters	344
7.4.3.2.15	CVI_PLAYER_SetVideoParameters	345
7.4.3.2.16	CVI_PLAYER_SetAOHandler	346
7.4.3.2.17	CVI_PLAYER_SetCustomArgAOHandler	347
7.4.3.2.18	CVI_PLAYER_SetVOHandler	347
7.4.3.2.19	CVI_PLAYER_SetCustomArgVOHandler	348
7.4.3.2.20	CVI_PLAYER_SetEventHandler	349

7.4.3.2.21	CVI_PLAYER_SetCustomArgEventHandler	350
7.4.3.2.22	CVI_PLAYER_SaveImage	350
7.4.3.2.23	CVI_PLAYER_GetMediaInfo	351
7.4.3.2.24	CVI_PLAYER_GetPlayInfo	352
7.4.3.2.25	CVI_PLAYER_GetVideoFrame	353
7.4.3.2.26	CVI_PLAYER_GetVideoPacket	353
7.4.3.2.27	CVI_PLAYER_GetVideoExtraPacket	354
7.4.3.2.28	CVI_PLAYER_SetVideoDecodeHandler	355
7.4.3.2.29	CVI_PLAYER_SetVideoCustomArgDecodeHandler	356
7.4.3.2.30	CVI_PLAYER_SetiAudioDecodeHandler	356
7.4.3.2.31	CVI_PLAYER_SetAudioCustomArgDecodeHandler	357
7.4.3.2.32	CVI_PLAYER_PacketContainSps	358
7.4.3.2.33	CVI_PLAYER_SeekPause	359
7.4.3.2.34	CVI_PLAYER_SeekFlage	359
7.4.3.2.35	CVI_PLAYER_SeekTime	360
7.4.3.2.36	CVI_PLAYER_PlayerSeep	361
7.4.3.2.37	CVI_PLAYER_GetForWardBackWardStatus	361
7.4.3.3	数据类型	362
7.4.3.3.1	CVI_PLAYER_EVENT_TYPE_E	363
7.4.3.3.2	CVI_PLAYER_EVENT_S	363
7.4.3.3.3	CVI_PLAYER_PACKET_S	364
7.4.3.3.4	CVI_PLAYER_MEDIA_INFO_S	364
7.4.3.3.5	CVI_PLAYER_HANDLE_T	364
7.4.3.3.6	CVI_PLAYER_FRAME_S	365
7.4.3.3.7	CVI_PLAYER_OUTPUT_HANDLER	366
7.4.3.3.8	CVI_PLAYER_CUSTOM_ARG_OUTPUT_HANDLER	366
7.4.3.3.9	CVI_PLAYER_EVENT_HANDLER	366
7.4.3.3.10	CVI_PLAYER_CUSTOM_ARG_EVENT_HANDLER	367
7.4.3.3.11	CVI_PLAYER_DECODE_HANDLER_S	367
7.4.3.3.12	CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER_S	368
7.4.4	Playback_Service	368
7.4.4.1	API 参考	368
7.4.4.1.1	CVI_PLAYER_SERVICE_GetDefaultParam	369
7.4.4.1.2	CVI_PLAYER_SERVICE_Create	370
7.4.4.1.3	CVI_PLAYER_SERVICE_Destroy	371
7.4.4.1.4	CVI_PLAYER_SERVICE_SetInput	371
7.4.4.1.5	CVI_PLAYER_SERVICE_GetMediaInfo	372
7.4.4.1.6	CVI_PLAYER_SERVICE_GetPlayInfo	373
7.4.4.1.7	CVI_PLAYER_SERVICE_Play	374
7.4.4.1.8	CVI_PLAYER_SERVICE_PlayerAndSeek	374
7.4.4.1.9	CVI_PLAYER_SERVICE_Stop	375
7.4.4.1.10	CVI_PLAYER_SERVICE_Pause	376
7.4.4.1.11	CVI_PLAYER_SERVICE_Seek	376
7.4.4.1.12	CVI_PLAYER_SERVICE_SetEventHandler	377
7.4.4.1.13	CVI_PLAYER_SERVICE_Resize	378
7.4.4.1.14	CVI_PLAYER_SERVICE_MoveTo	379
7.4.4.1.15	CVI_PLAYER_SERVICE_ToggleFullscreen	379
7.4.4.1.16	CVI_PLAYER_SERVICE_GetSignals	380
7.4.4.1.17	CVI_PLAYER_SERVICE_GetSlots	381
7.4.4.1.18	CVI_PLAYER_SERVICE_SeekTime	382

7.4.4.1.19	CVI_PLAYER_SERVICE_SeekFlage	382
7.4.4.1.20	CVI_PLAYER_SERVICE_SeekPause	383
7.4.4.1.21	CVI_PLAYER_SERVICE_TouchSeekPause	384
7.4.4.1.22	CVI_PLAYER_SERVICE_PlayerSeep	384
7.4.4.1.23	CVI_PLAYER_SERVICE_PlayerSeepBack	385
7.4.4.1.24	CVI_PLAYER_SERVICE_GetFileMediaInfo	386
7.4.4.2	数据类型	387
7.4.4.2.1	CVI_PLAYER_SERVICE_PARAM_S	387
7.4.4.2.2	CVI_PLAYER_SERVICE_EVENT_TYPE_E	388
7.4.4.2.3	CVI_PLAYER_SERVICE_EVENT_S	389
7.4.4.2.4	CVI_PLAYER_SERVICE_EVENT_HANDLER	390
7.4.4.2.5	CVI_PLAYER_SERVICE_SLOTS_S	390
7.4.4.2.6	CVI_PLAYER_SERVICE_SIGNALS_S	391
7.4.4.2.7	CVI_PLAYER_SERVICE_HANDLE_T	392
8	文件修复	393
8.1	概述	393
8.2	实现逻辑	394
8.2.1	解析 mdat	394
8.2.2	更新 moov	395
8.3	API 参考	396
8.3.1	目录结构	396
8.3.2	API	398
8.3.2.1	CVI_FILE_RECOVER_Create	399
8.3.2.2	CVI_FILE_RECOVER_Destroy	400
8.3.2.3	CVI_FILE_RECOVER_Open	400
8.3.2.4	CVI_FILE_RECOVER_Check	401
8.3.2.5	CVI_FILE_RECOVER_Dump	402
8.3.2.6	CVI_FILE_RECOVER_Recover	402
8.3.2.7	CVI_FILE_RECOVER_RecoverAsync	403
8.3.2.8	CVI_FILE_RECOVER_RecoverJoin	404
8.3.2.9	CVI_FILE_RECOVER_Close	405
8.3.2.10	CVI_FILE_RECOVER_SetEventHandler	405
8.3.2.11	CVI_FILE_RECOVER_SetCustomArgEventHandler	406
8.3.2.12	CVI_FILE_RECOVER_PreallocateState	407
8.3.3	数据结构	408
8.3.3.1	CVI_FILE_RECOVER_HANDLE_T	408
8.3.3.2	CVI_FILE_RECOVER_EVENT_TYPE_E	408
8.3.3.3	CVI_FILE_RECOVER_EVENT_S	409
8.3.3.4	CVI_FILE_RECOVER_EVENT_HANDLER	409
8.3.3.5	CVI_FILE_RECOVER_CUSTOM_ARG_EVENT_HANDLER	410
9	服务组件	411
9.1	Rtsp_Service	411
9.1.1	API 参考	413
9.1.1.1	CVI_RTSP_SERVICE_Create	413
9.1.1.2	CVI_RTSP_SERVICE_Destroy	414
9.1.1.3	CVI_RTSP_SERVICE_UpdateParam	415
9.1.1.4	CVI_RTSP_SERVICE_StartMute	416
9.1.1.5	CVI_RTSP_SERVICE_StopMute	416
9.1.1.6	CVI_RTSP_SERVICE_StartStop	417

9.1.2	数据类型	418
9.1.2.1	MAX_RTSP_STREAM_NAME_LEN	418
9.1.2.2	CVI_RTSP_SERVICE_PARAM_S	418
9.1.2.3	CVI_RTSP_SERVICE_VIDEO_CODEC_E	420
9.1.2.4	CVI_RTSP_SERVICE_AUDIO_CODEC_E	420
9.1.2.5	CVI_RTSP_SERVICE_CALLBACK	421
9.1.2.6	CVI_RTSP_SERVICE_HANDLE_T	421
9.2	Storage_Service	422
9.2.1	API 参考	422
9.2.1.1	CVI_STORAGE_SERVICE_Create	422
9.2.1.2	CVI_STORAGE_SERVICE_Destroy	423
9.2.1.3	CVI_STORAGE_SERVICE_GetFsInfo	424
9.2.1.4	CVI_STORAGE_SERVICE_GetDevInfo	424
9.2.1.5	CVI_STORAGE_SERVICE_Format	425
9.2.1.6	CVI_STORAGE_SERVICE_Mount	426
9.2.1.7	CVI_STORAGE_SERVICE_Umount	427
9.2.2	数据类型	427
9.2.2.1	STORAGE_LABEL_LEN	428
9.2.2.2	CVI_STORAGE_SERVICE_PARAM_S	428
9.2.2.3	CVI_STORAGE_SERVICE_EVENT_CALLBACK	429
9.2.2.4	CVI_STORAGE_SERVICE_HANDLE_T	429
9.3	Photo_Service	429
9.3.1	API 参考	430
9.3.1.1	CVI_PHOTO_SERVICE_Create	430
9.3.1.2	CVI_PHOTO_SERVICE_Destroy	431
9.3.1.3	CVI_PHOTO_SERVICE_PivCapture	431
9.3.1.4	CVI_PHOTO_SERVICE_WaitPivFinish	432
9.3.2	数据类型	433
9.3.2.1	CVI_PHOTO_SERVICE_PARAM_S	433
9.3.2.2	CVI_PHOTO_SERVICE_HANDLE_T	434
9.4	Liveview_Service	435
9.4.1	API 参考	435
9.4.1.1	CVI_LIVEVIEW_SERVICE_Create	435
9.4.1.2	CVI_LIVEVIEW_SERVICE_Destroy	436
9.4.1.3	CVI_LIVEVIEW_SERVICE_GetParam	436
9.4.2	数据类型	437
9.4.2.1	CVI_DISP_MAX_WND_NUM	438
9.4.2.2	CVI_LIVEVIEW_SERVICE_WNDATTR_S	438
9.4.2.3	CVI_LIVEVIEW_SERVICE_ATTR_S	439
9.4.2.4	CVI_LIVEVIEW_SERVICE_PARAM_S	440
9.4.2.5	CVI_LIVEVIEW_SERVICE_HANDLE_T	440
9.5	ADAS_Service	441
9.5.1	API 参考	441
9.5.1.1	CVI_ADAS_SERVICE_Create	441
9.5.1.2	CVI_ADAS_SERVICE_Destroy	442
9.5.1.3	CVI_ADAS_SERVICE_SetState	442
9.5.2	数据类型	443
9.5.2.1	CVI_ADAS_SERVICE_VOICE_CALLBACK	444
9.5.2.2	CVI_ADAS_SERVICE_LABEL_CALLBACK	444
9.5.2.3	CVI_ADAS_SERVICE_CALLBACK_S	445

9.5.2.4	CVI_ADAS_SERVICE_VPROC_ATTR_S	445
9.5.2.5	CVI_ADAS_SERVICE_MODEL_ATTR_S	446
9.5.2.6	CVI_ADAS_SERVICE_HANDLE_T	446
9.5.2.7	CVI_ADAS_SERVICE_ATTR_S	447
9.5.2.8	CVI_ADAS_SERVICE_PARAM_S	447
9.5.2.9	CVI_ADAS_SERVICE_CTX_S	448
9.5.2.10	CVI_ADAS_SERVICE_CMD_E	449
9.6	Speech_Service	450
9.6.1	API 参考	450
9.6.1.1	CVI_SPEECH_SERVICE_Create	450
9.6.1.2	CVI_SPEECH_SERVICE_Destroy	451
9.6.1.3	CVI_SPEECH_SERVICE_StartSpeech	452
9.6.1.4	CVI_SPEECH_SERVICE_StopSpeech	452
9.6.1.5	CVI_SPEECH_SERVICE_Register	453
9.6.2	数据类型	454
9.6.2.1	FUNC_SPEECH_HANDLE	454
9.6.2.2	CVI_SPEECH_HANDLE_S	454
9.6.2.3	CVI_SPEECH_SERVICE_PARAM_S	455
9.7	QRCode_Service	455
9.7.1	API 参考	455
9.7.1.1	CVI_QRCode_Service_Create	456
9.7.1.2	CVI_QRCode_Service_Destroy	456
9.7.2	数据类型	457
9.7.2.1	CVI_QRCODE_SERVICE_HANDLE_T	457
9.7.2.2	CVI_QRCODE_SERVICE_PARAM_S	458

修订记录

Revision	Date	Description
1.0.0	2024/05/11	初版

1 声明



法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

联系我们

地址 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

电话 +86-10-57590723 +86-10-57590724

邮编 100094（北京）518100（深圳）

官方网站 <https://www.sophgo.com/>

技术论坛 <https://developer.sophgo.com/forum/index.html>

2 概述

2.1 阅读说明

该文档为 Framework 通用软件开发指南，该开发指南旨在讲述 Framework 组件，如图 2-1 所示。

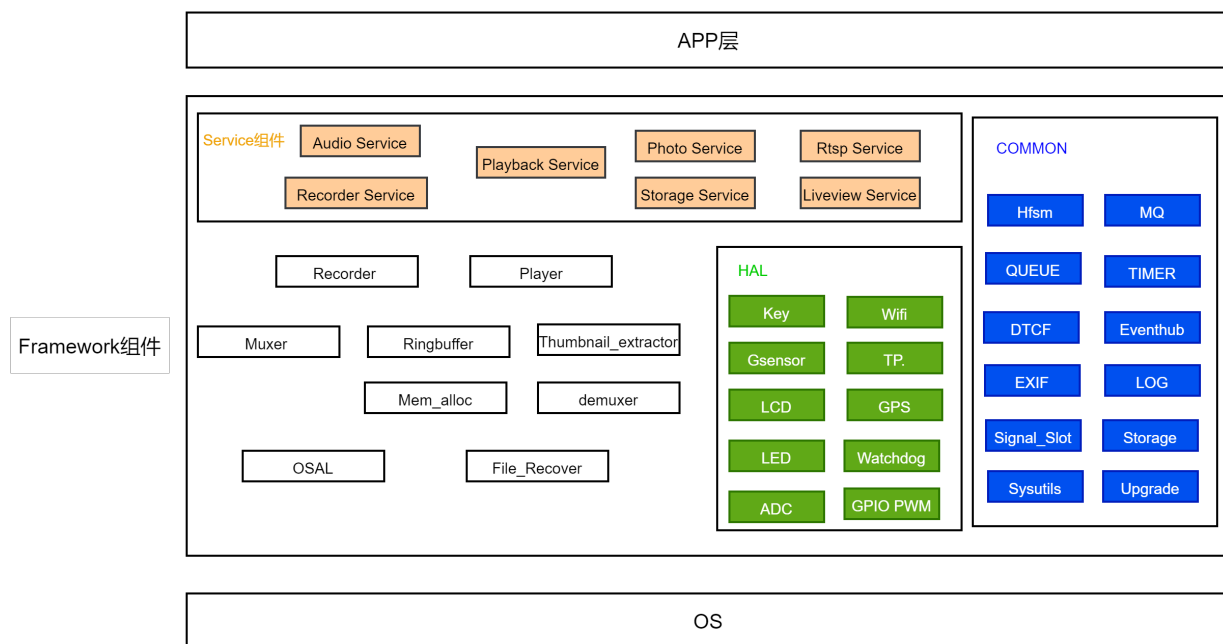


图 2.1: Framework 通用软件

为方便讲述和记忆这些组件，现将组件分为七大类：

- HAL (Hardware Adapter Layer)：是指硬件外设适配层软件模块，包含 Key, WiFi, TouchPad(TP) 等。
- OSAL: 操作系统抽象层 (Operating System Abstraction Layer)，是一种以实现多任务为核心的系统资源管理机制。
- Common: 一些普遍使用的组件，例如 timer (定时器)。
- 录像模块: 录像所使用的组件，例如 muxer (封装)。
- 播放器模块: 设备播放视频所涉及的组件，例如 demuxer (解封装)。

- 文件修复 (File_Recover): 录像文件不完整时启动。
- 服务组件: 这类组件专门对接 APP 层, 例如预览服务 (Liveview Service), 回放服务 (Playback Service)。

2.2 目录结构

这些组件分布在 framework 文件夹和 cpsl 文件夹:

```
cpsl
├── /hal
├── /common
└── /osal
framework
├── /services: 服务组件
└── /components: 其他组件
```

注意:

- 这些组件可以有选择性打开, 在编译之前执行 make menuconfig, 其中:
 - Services options 选项中提供了服务组件的开关选项。
 - Peripheral options 选项中提供了 HAL 层组件的开关选项。
 - Components options 选项中提供了 components 文件夹下组件的开关选项。

3 HAL

3.1 概述

HAL (Hardware Adapter Layer) 是指硬件外设适配层软件模块。基于此模块产品应用层通过接口访问外设，外设差异性体现在 HAL 和驱动。如果客户更换外设，只需要改动 HAL 和驱动。HAL 模块上下文关系如图 3-1 所示。

注解:

- 提炼 HAL 层，外设相关的实现放在 HAL 层，对外提供接口，程序框架清晰简单。
- 硬件功能实现集中化，便于管理和维护。

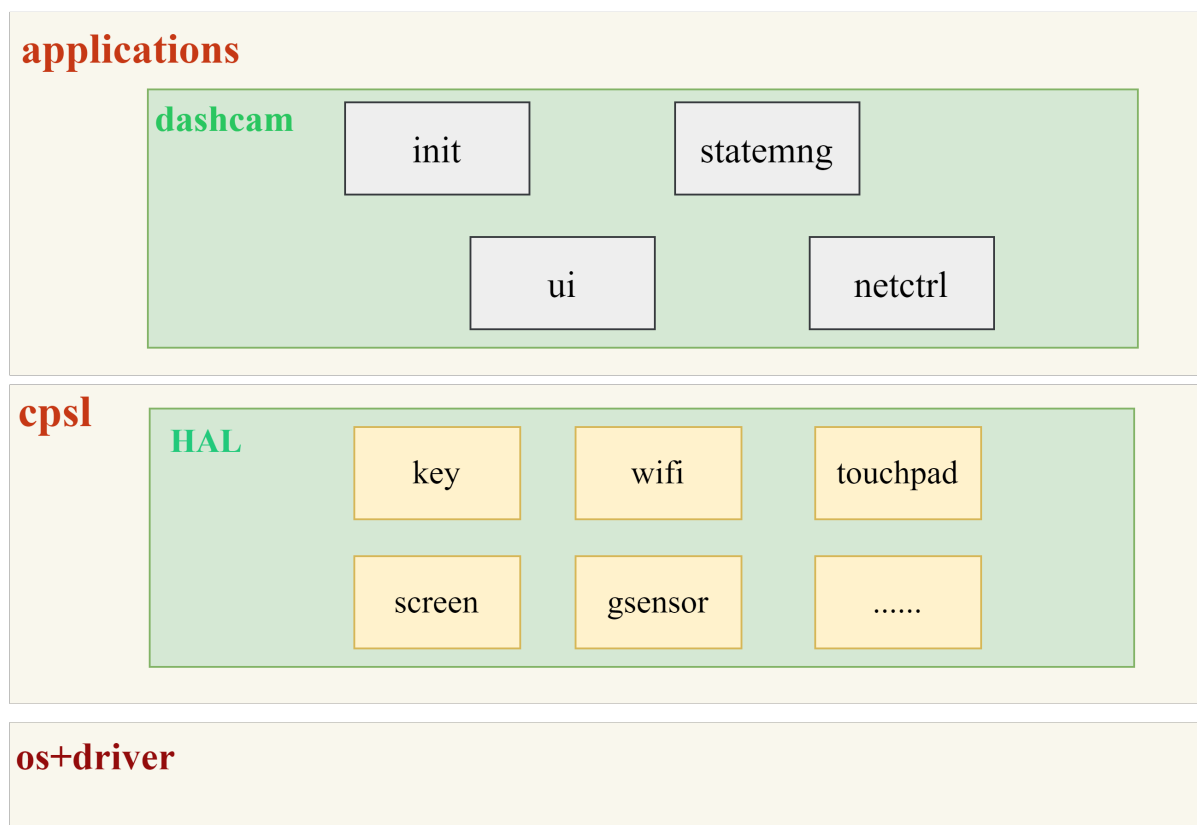


图 3.1: HAL 模块框图

3.2 功能

HAL 层包含以下功能：

- 按键 (Key)
- WiFi
- 触摸屏 (TP)
- 显示屏幕 (LCD)
- 重力加速度计 (Gsensor)
- GPS
- LED
- 看门狗 (Watchdog)
- 电量计 (ADC)
- GPIO、PWM、UART

3.2.1 按键

3.2.1.1 概述

key 按键（黄色部分）上层使用者为 keymng，下层依赖常电区寄存器或者 IO driver（查询 IO 对应状态）。当前按键模块上下文关系如图 3-2 所示。

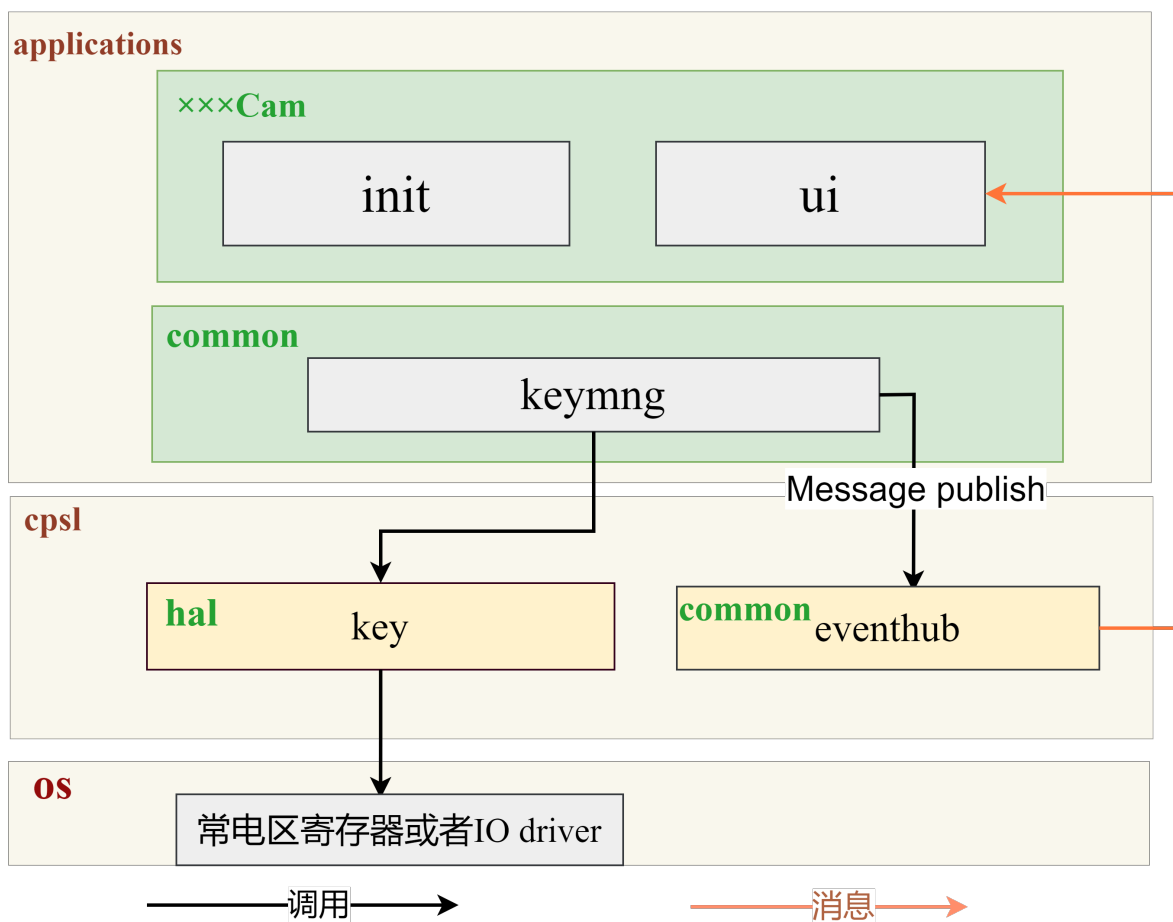


图 3.2: key 模块上下文

3.2.1.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 key 开关选项。

3.2.1.3 API 参考

key 模块主要提供按键状态获取，该功能模块为用户提供以下 API：

- `CVI_HAL_KEY_Init`：初始化 key 模块。
- `CVI_HAL_KEY_Deinit`：去初始化 key 模块。
- `CVI_HAL_KEY_GetState`：获取按键状态。

3.2.1.3.1 CVI_HAL_KEY_Init

【描述】

初始化 key 模块。

【语法】

```
int32_t CVI_HAL_KEY_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_key.h
- 库文件: libcvi_hal_key.a / libcvi_hal_key.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.1.3.2 CVI_HAL_KEY_Deinit

【描述】

去初始化 key 模块。

【语法】

```
int32_t CVI_HAL_KEY_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_key.h
- 库文件: libcvi_hal_key.a / libcvi_hal_key.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.1.3.3 CVI_HAL_KEY_GetState

【描述】

获得按键状态。

【语法】

```
int32_t CVI_HAL_KEY_GetState(CVI_HAL_KEY_IDX_E enKeyIdx,  
                             CVI_HAL_KEY_STATE_E* penKeyState);
```

【参数】

参数名称	描述	输入/输出
enKeyIdx	按键索引。 取值范围:[0,CVI_HAL_KEY_IDX_BUIT)。	输入
penKeyState	按键状态指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_key.h
- 库文件: libcvi_hal_key.a / libcvi_hal_key.so

【注意】

- penKeyState 指向内容为:
CVI_HAL_KEY_STATE_DOWN 表示按键按下。
CVI_HAL_KEY_STATE_UP 表示按键弹起。
- 按键的防抖以及长按功能由 keymng 实现。

【举例】

无。

3.2.1.4 数据类型

相关数据类型定义如下：

- CVI_HAL_KEY_IDX_E：定义按键 ID。
- CVI_HAL_KEY_STATE_E：定义按键状态。

3.2.1.4.1 CVI_HAL_KEY_IDX_E

【说明】

定义按键 ID。

【定义】

```
typedef enum cviHAL_KEY_IDX_E
{
    CVI_HAL_KEY_IDX_0 = 0, /**<key index 0*/
    CVI_HAL_KEY_IDX_1,
    CVI_HAL_KEY_IDX_2,
    CVI_HAL_KEY_IDX_3,
    CVI_HAL_KEY_IDX_4,
    CVI_HAL_KEY_IDX_5,
    CVI_HAL_KEY_IDX_6,
    CVI_HAL_KEY_IDX_BUIT
} CVI_HAL_KEY_IDX_E;
```

【成员】

成员名称	描述
CVI_HAL_KEY_IDX_0	ID 为 0 的按键
CVI_HAL_KEY_IDX_1	ID 为 1 的按键
CVI_HAL_KEY_IDX_x	ID 为 x 的按键
CVI_HAL_KEY_IDX_BUIT	按键 ID 最大值 +1，可用于按键的个数

【注意事项】

添加按键请参考 2.2.1.5 客制化场景修改指南章节。

【相关数据类型及接口】

无。

3.2.1.4.2 CVI_HAL_KEY_STATE_E

【说明】

定义按键 ID。

【定义】

```
typedef enum cviHAL_KEY_STATE_E
{
    CVI_HAL_KEY_STATE_DOWN = 0, /**<key down state*/
    CVI_HAL_KEY_STATE_UP, /**<key up state*/
    CVI_HAL_KEY_STATE_BUIT
} CVI_HAL_KEY_STATE_E;
```

【成员】

成员名称	描述
CVI_HAL_KEY_STATE_DOWN	按键按下状态
CVI_HAL_KEY_STATE_UP	按键弹起状态
CVI_HAL_KEY_STATE_BUIT	按键状态最大值 +1, 可用于按键状态个数

【注意事项】

当前按键状态，按键的防抖以及长按功能由 keymng 实现。

【相关数据类型及接口】

无。

3.2.1.5 客制化场景修改指南

注解： 下面介绍如何添加新的按键。

步骤 1 引脚配置

在 cvi_board_init.c 配置 key 引脚，请用户根据硬件原理图合理配置引脚

步骤 2 hal 层修改

1、新增按键索引

若当前使用的按键数量已大于枚举索引，需在 cpsl/hal/key/include/cvi_hal_key.h 中添加 CVI_HAL_KEY_IDX_x, x 为新按键 ID

```
typedef enum cviHAL_KEY_IDX_E
{
    CVI_HAL_KEY_IDX_0 = 0, /**<key index 0*/
    CVI_HAL_KEY_IDX_1,
    CVI_HAL_KEY_IDX_2,
    ...
    CVI_HAL_KEY_IDX_x,
    CVI_HAL_KEY_IDX_BUIT
} CVI_HAL_KEY_IDX_E;
```

2、按键定义

按键触发可以基于 ADC 或 GPIO 实现，若 GPIO 口足够可选择 GPIO，否则推荐使用 ADC。需在 cpsl/hal/key/include/cvi_hal_key.c 中为每个按键添加定义

```
// GPIO实现方式：需为每个按键添加GPIO
static CVI_GPIO_ID_E GPIOID[] =
    {{CVI_HAL_KEY_IDX_0, CVI_GPIOE_08}, // PWR_GPIO 可用于电源管理
    ...
    {CVI_HAL_KEY_IDX_x, CVI_GPIOy_y}}; // 对应GPIO
```

```
// ADC实现方式：需为每个按键添加对应ADC值
static CVI_GPIO_ID_E GPIOID[] =
    {{CVI_HAL_KEY_IDX_0, CVI_GPIOE_08}, // PWR_GPIO 可用于电源管理
    {{CVI_HAL_KEY_IDX_1, num},
    ...
    {CVI_HAL_KEY_IDX_x, num}}; // 对应ADC数值
```

ADC 数值如何确定：

- 1、确定板端 ADC 功能已经打开（Menuconfig->Peripheral options）
- 2、按对应按键后，板端读取 ADC 数值即可

```
cat /sys/bus/iio/devices/iio:device0/in_voltages_raw // 若选择ADC通道1, in_voltage1_
↪raw, 以此类推
```

3、调整 cpsl/hal/key/include/cvi_hal_key.c 的宏定义

```
#define ADC_CHANNEL1_KEY (1) // 当前默认选用 ADC 通道1
#define ADC_CH_KEY_VALUE_RANGE (200) // ADC检测范围 [num-200 , num+200]
```

步骤 3 应用层修改

在 applications/common/devmng/include/cvi_keymng.h 对 应 修 改
CVI_KEYMNG_KEY_IDX_E 结构体

步骤 4 配置修改

key 配置修改请参考 config_devmng.ini。

```
[keymng.key]
key_cnt      = 1 # 按键数量
key_type0    = 0 # 0支持click 1支持hold
key_id0      = 1 # 默认1
longkey_enable0 = 1 # 使能长按事件
longkey_time0 = 2000 # 长按触发时间
```

3.2.2 WIFI

3.2.2.1 概述

wifi 的实现逻辑集中在 wifi 模块，对外提供功能接口，向下除了依赖 wifi 芯片对应驱动程序、wifi 硬件管脚连接，内部依赖标准组件 hostapd、wpa_supplicant、dhcpcd。

上下文依赖关系如图 3-3 所示。

注解:

- 不同型号 wifi 驱动不一致，wifi 硬件管脚也可能不一致，但 hostpad、dhcpcd 保持不变。
- wifi 可修改对应配置文件，请参考 config_devmng.ini。

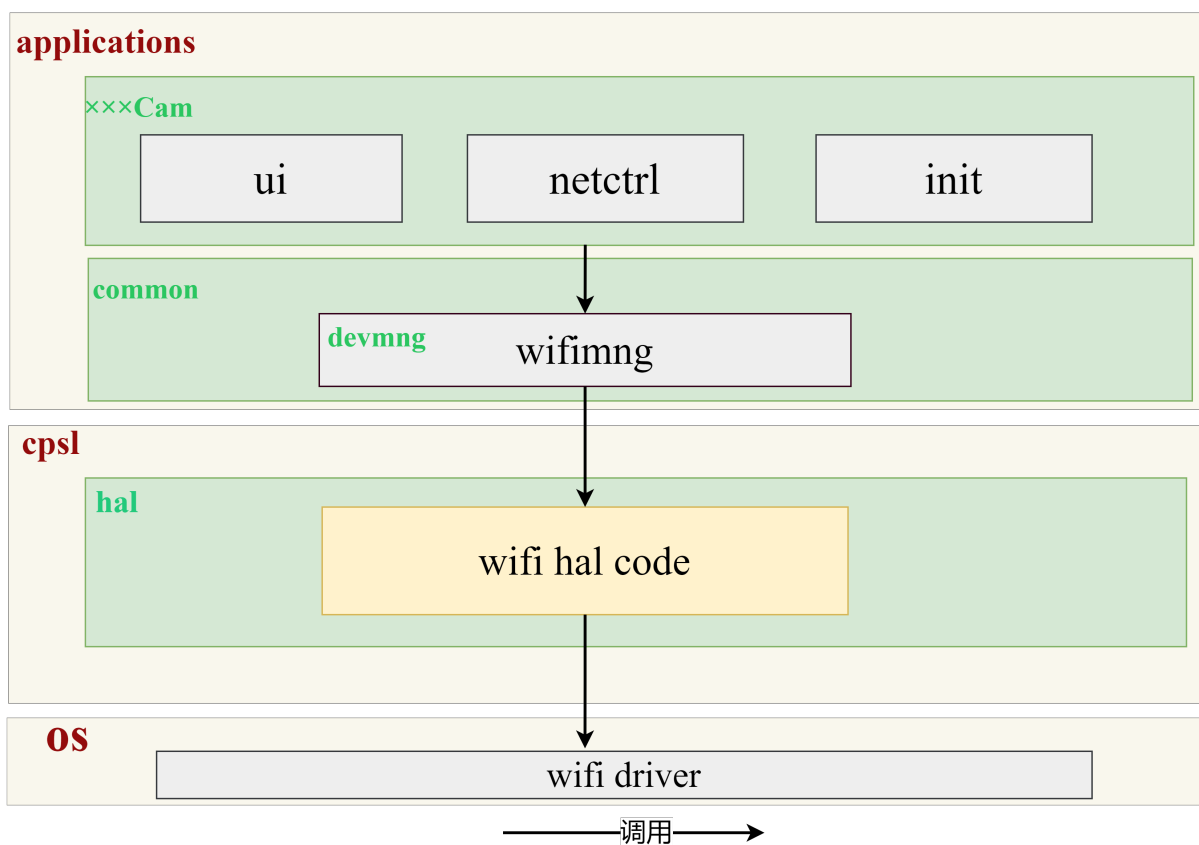


图 3.3: WIFI 模块上下文

3.2.2.2 功能开关

menuconfig 中 Peripheral options 选项中提供了开关选项和 WIFI 模组的选择。

3.2.2.3 API 参考

该功能模块为用户提供以下 API:

- CVI_HAL_WIFI_Init : 初始化 WIFI 模块。
- CVI_HAL_WIFI_Deinit : 去初始化 WIFI 模块。
- CVI_HAL_WIFI_Start : 启动 WIFI。
- CVI_HAL_WIFI_Stop : 停止 WIFI。
- CVI_HAL_WIFI_GetStartedStatus : 获取 WIFI 是否启动。
- CVI_HAL_WIFI_CheckeCfgValid : 检查 config 是否有效。

3.2.2.3.1 CVI_HAL_WIFI_Init

【描述】

初始化 WIFI 模块。

【语法】

```
int32_t CVI_HAL_WIFI_Init(CVI_HAL_WIFI_MODE_E enMode);
```

【参数】

参数名称	描述	输入/输出
enMode	WIFI 模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_wifi.h
- 库文件: libcvi_hal_wifi.a / libcvi_hal_wifi.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.2.3.2 CVI_HAL_WIFI_Deinit

【描述】

去初始化 WIFI 模块。

【语法】

```
int32_t CVI_HAL_WIFI_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_wifi.h
- 库文件: libcvi_hal_wifi.a / libcvi_hal_wifi.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.2.3.3 CVI_HAL_WIFI_Start

【描述】

启动 WIFI 模块。

【语法】

```
int32_t CVI_HAL_WIFI_Start(const CVI_HAL_WIFI_CFG_S *pstCfg);
```

【参数】

参数名称	描述	输入/输出
pstCfg	WIFI 配置结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_wifi.h`
- 库文件: `libcvi_hal_wifi.a` / `libcvi_hal_wifi.so`

【注意】

- 依赖 `CVI_HAL_WIFI_Init` 接口调用, 当 `CVI_HAL_WIFI_Init` 接口未调用或者失败, 调用该接口失败
- `pstCfg` 中成员 `enmode` 需要与 `CVI_HAL_WIFI_Init` 接口一致

【举例】

无。

3.2.2.3.4 CVI_HAL_WIFI_Stop

【描述】

启动 WIFI 模块。

【语法】

```
int32_t CVI_HAL_WIFI_Stop(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_wifi.h`
- 库文件: `libcvi_hal_wifi.a` / `libcvi_hal_wifi.so`

【注意】

该接口只是停止 WIFI 功能, WIFI 驱动不卸载。

【举例】

无。

3.2.2.3.5 CVI_HAL_WIFI_GetStartedStatus

【描述】

获得 WIFI 是否启动。

【语法】

```
int32_t CVI_HAL_WIFI_GetStartedStatus(bool *pbEnable);
```

【参数】

参数名称	描述	输入/输出
pbEnable	wifi 是否启动指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_wifi.h
- 库文件: libcvi_hal_wifi.a / libcvi_hal_wifi.so

【注意】

无。

【举例】

无。

3.2.2.3.6 CVI_HAL_WIFI_CheckeCfgValid

【描述】

检查 WIFI 参数配置是否有效。

【语法】

```
int32_t CVI_HAL_WIFI_CheckeCfgValid(const CVI_HAL_WIFI_CFG_S *pstCfg, bool  
↪ *pCfgValid);
```

【参数】

参数名称	描述	输入/输出
pstCfg	wifi 配置结构体指针	输入
pCfgValid	wifi 参数是否有效指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_wifi.h
- 库文件: libcvi_hal_wifi.a / libcvi_hal_wifi.so

【注意】

无。

【举例】

无。

3.2.2.4 数据类型

相关数据类型定义如下：

- CVI_HAL_WIFI_MODE_E：定义 WIFI 模式枚举。
- CVI_HAL_WIFI_STA_MODE_E：定义 WIFI STA 模式枚举。
- CVI_HAL_WIFI_COMMON_CFG_S：定义 WIFI 通用配置结构体有。
- CVI_HAL_WIFI_APMODE_CFG_S：定义 AP 模式结构体。
- CVI_HAL_WIFI_STAMODE_COMMON_CFG_S：定义 STA 通用模式结构体。
- CVI_HAL_WIFI_STAMODE_SENIOR_CFG_S：定义 STA 高级模式结构体。
- CVI_HAL_WIFI_STAMODE_CFG_S：定义 STA 模式结构体。
- CVI_HAL_WIFI_CFG_S：定义 WIFI 结构体。
- CVI_HAL_WIFI_SCAN_RESULT_S：定义热点消息结构体。
- CVI_HAL_WIFI_SCAN_RESULTSET_S：定义搜索 WIFI 热点结构体。

3.2.2.4.1 CVI_HAL_WIFI_MODE_E

【说明】

定义 WIFI 模式枚举。

【定义】

```
typedef enum cviHAL_WIFI_MODE_E {  
    CVI_HAL_WIFI_MODE_STA = 0, /**<STA mode*/  
    CVI_HAL_WIFI_MODE_AP, /**<AP mode*/  
    CVI_HAL_WIFI_MODE_BUIT  
} CVI_HAL_WIFI_MODE_E;
```

【成员】

成员名称	描述
CVI_HAL_WIFI_MODE_STA	STA 模式
CVI_HAL_WIFI_MODE_AP	AP 模式
CVI_HAL_WIFI_MODE_BUIT	模式最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.2 CVI_HAL_WIFI_STA_MODE_E**【说明】**

定义 WIFI STA 模式枚举。

【定义】

```
typedef enum cviHAL_WIFI_MODE_E {  
    CVI_HAL_WIFI_STA_MODE_COMMON = 0, /**<ap broadcast ssid*/  
    CVI_HAL_WIFI_STA_MODE_SENIOR, /**<ap hide ssid*/  
    CVI_HAL_WIFI_STA_MODE_BUIT  
} CVI_HAL_WIFI_STA_MODE_E;
```

【成员】

成员名称	描述
CVI_HAL_WIFI_STA_MODE_COMMON	STA 普通模式
CVI_HAL_WIFI_STA_MODE_SENIOR	STA 高级模式
CVI_HAL_WIFI_STA_MODE_BUIT	STA 模式最大值 +1

【注意事项】

- STA 普通模式用于连接开启了 SSID 广播功能的 AP 热点。
- STA 高级模式用于连接 SSID 隐藏的 AP 热点。

【相关数据类型及接口】

无。

3.2.2.4.3 CVI_HAL_WIFI_COMMON_CFG_S

【说明】

定义 WIFI 通用配置结构体。

【定义】

```
typedef struct cviHAL_WIFI_COMMON_CFG_S {  
    /**<wifi ssid,aszWiFiSSID len decided by strlen(aszWiFiSSID)*/  
    char szWiFiSSID[CVI_HAL_WIFI_SSID_LEN];  
    /**<wifi password,szWiFiPassWord len decided by strlen(szWiFiPassWord)*/  
    char szWiFiPassWord[CVI_HAL_WIFI_PASSWORD_LEN];  
} CVI_HAL_WIFI_COMMON_CFG_S;
```

【成员】

成员名称	描述
szWiFiSSID	wifi ssid(热点名称)
szWiFiPassWord	wifi 密码

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.4 CVI_HAL_WIFI_APMODE_CFG_S

【说明】

定义 wifi ap 模式配置结构体。

【定义】

```
typedef struct cviHAL_WIFI_APMODE_CFG_S {  
    bool bHideSSID;/**<true:ssid broadcast,false:hide*/  
    int32_t s32Channel; /**<wifi channel */  
    CVI_HAL_WIFI_COMMON_CFG_S stCfg;/**<wifi common cfg*/  
} CVI_HAL_WIFI_APMODE_CFG_S;
```

【成员】

成员名称	描述
bHideSSID	是否隐藏 wifi ssid(wifi 热点名称)
s32Channel	wifi 通道
stCfg	通用配置结构体

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.5 CVI_HAL_WIFI_STAMODE_COMMON_CFG_S**【说明】**

定义 STA 通用模式结构体。

【定义】

```
typedef struct cviHAL_WIFI_STAMODE_COMMON_CFG_S {  
    CVI_HAL_WIFI_COMMON_CFG_S stCfg;  
} CVI_HAL_WIFI_STAMODE_COMMON_CFG_S;
```

【成员】

成员名称	描述
stCfg	通用配置结构体

【注意事项】

sta 模式下普通搜索热点使用。

【相关数据类型及接口】

无。

3.2.2.4.6 CVI_HAL_WIFI_STAMODE_SENIOR_CFG_S**【说明】**

定义 STA 高级模式结构体。

【定义】

```
typedef struct cviHAL_WIFI_STAMODE_SENIOR_CFG_S {  
    CVI_HAL_WIFI_COMMON_CFG_S stCfg;  
} CVI_HAL_WIFI_STAMODE_SENIOR_CFG_S;
```

【成员】

成员名称	描述
stCfg	通用配置结构体

【注意事项】

用于连接 ssid 被隐藏 AP 热点。

【相关数据类型及接口】

无。

3.2.2.4.7 CVI_HAL_WIFI_STAMODE_CFG_S

【说明】

定义 STA 模式结构体。

【定义】

```
typedef struct cviHAL_WIFI_STAMODE_CFG_S {  
    CVI_HAL_WIFI_STA_MODE_E enStaMode; /**<STA mode*/  
    union tagHAL_WIFI_STAMODE_CFG_U {  
        /**<cfg for common connect ap mode*/  
        CVI_HAL_WIFI_STAMODE_COMMON_CFG_S stCommonCfg;  
        /**<cfg for senior connect ap mode*/  
        CVI_HAL_WIFI_STAMODE_SENIOR_CFG_S stSeniorCfg;  
    } unCfg;  
} CVI_HAL_WIFI_STAMODE_CFG_S;
```

【成员】

成员名称	描述
enStaMode	sta 模式
unCfg	选定模式对应配置

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.8 CVI_HAL_WIFI_CFG_S

【说明】

定义 WIFI 模式结构体。

【定义】

```
typedef struct cviHAL_WIFI_CFG_S {  
    CVI_HAL_WIFI_MODE_E enMode; /**<wifi mode*/  
    union tagHAL_WIFI_CFG_U {  
        CVI_HAL_WIFI_APMODE_CFG_S stApCfg; /**<wifi AP cfg*/  
        CVI_HAL_WIFI_STAMODE_CFG_S stStaCfg; /**<wifi STA cfg*/  
    } unCfg;  
} CVI_HAL_WIFI_CFG_S;
```

【成员】

成员名称	描述
enMode	WIFI 模式
unCfg	选定模式对应配置

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.9 CVI_HAL_WIFI_SCAN_RESULT_S**【说明】**

定义热点信息结构体。

【定义】

```
typedef struct cviHAL_WIFI_SCAN_RESULT_S {  
    char szSSID[CVI_HAL_WIFI_SSID_LEN];/**<Hotspot SSID */  
    char szBssid[CVI_HAL_WIFI_STR_LEN];/**<Hotspot BSSID */  
    uint32_t u32Channel;/**<Hotspot channel*/  
    uint32_t u32Signal;/**<Hotspot signal strength,value[0,100]*/  
} CVI_HAL_WIFI_SCAN_RESULT_S;
```

【成员】

成员名称	描述
szSSID	wifi 热点标识
szBssid	wifi 热点网卡标识
u32Channel	wifi 热点通道
u32Signal	wifi 热点信号强度，范围 [0,100]

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.4.10 CVI_HAL_WIFI_SCAN_RESULTSET_S**【说明】**

定义搜索 WIFI 热点结构体。

【定义】

```
typedef struct cviHAL_WIFI_SCAN_RESULTSET_S {  
    uint32_t u32Num;/**<Hotspot num*/  
    CVI_HAL_WIFI_SCAN_RESULT_S astResult[CVI_HAL_WIFI_RESULT_SIZE];  
} CVI_HAL_WIFI_SCAN_RESULTSET_S;
```

【成员】

成员名称	描述
u32Num	搜索 wifi 热点个数
astResult	搜索 wifi 热点结构体数组

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.2.5 客制化场景修改指南

注解： 下面介绍如何添加新的 WIFI 模块，注意当前车载 turnkey 已经默认配置 wifi 模块，用户只需参考以下步骤即可（确保硬件已经妥当）。

以 rtl8189fs WIFI 型号为例，添加具体步骤如下：

步骤 1 引脚配置

在 `cvi_board_init.c` 配置 WIFI 引脚，请用户根据硬件原理图合理配置引脚

```
pinmux_config(PINMUX_SDIO1); // 默认SDIO 1
PINMUX_CONFIG(VIVO_D10, XGPIOB_11); // pinmux , function
```

步骤 2 驱动安装

- 1、在 `platform/osdrv/wifi` 添加新 WIFI 模块目录，并添加驱动文件
- 2、修改 `platform/osdrv/wifi/Makefile`

```
all:
    ifeq ($(CONFIG_WIFI_RTL8189),y)
        @if [ -d sdio_rtl8189fs ];then cd sdio_rtl8189fs; make;fi

clean:
    @if [ -d sdio_rtl8189fs ];then cd sdio_rtl8189fs; make clean;fi
```

步骤 3 hal 层修改

- 1、修改 `cpsl/hal/wifi/` 下的 `kbuild`、`kconfig` 和 `makefile`

`kbuild`

```
else ifeq (${CONFIG_WIFI_RTL8188}, y)
    KBUILD_DEFINES += -DCONFIG_WIFI_RTL8188
```

`kconfig`

```
config WIFI_RTL8188
bool "Enable wifi RTL8188"
help
Say Y if you want to enable wifi RTL8188.
```

makefile

```
wifi-$(CONFIG_WIFI_RTL8189) += src/rtl8189fs
```

2、在 cpsl/hal/wifi/src/添加新 WIFI 模块目录，并仿照添加 cvi_wifi_hal.c

步骤 4 配置修改

wifi 配置修改请参考 config_devmng.ini

```
; wifi
[wifi_config]
enable      = 1 ; 0:disabled 1:enable
mode        = 1 ; 0:STA mode 1: AP mode
ssid        = "cvi_cardv_ap" ; 32 char max
password     = "12345678" ; 32 char max
ssid_hide   = 0 ; 0:hide 1:show
channel     = 6
```

3.2.3 触摸屏

3.2.3.1 概述

触摸屏（TP）提供触摸信息获取，使用者为 ui，向下依赖触摸屏驱动以及触摸屏硬件布局，当前触摸屏上下文依赖关系如图 3-4 所示。

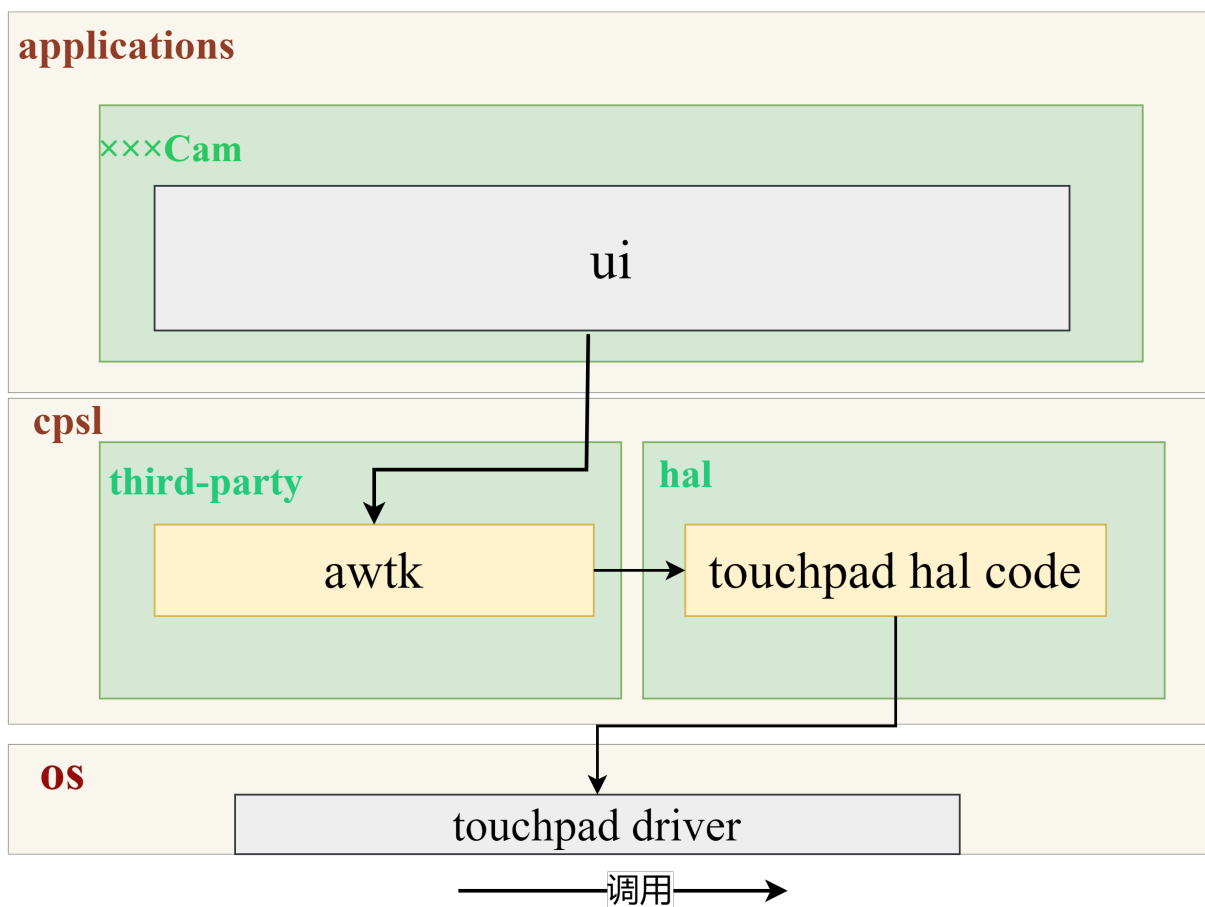


图 3.4: TP 模块上下文

3.2.3.2 功能开关

menuconfig 中 Peripheral options 选项中提供了开关选项和触摸屏的选择。

触摸屏功能开关打开后提供获取触摸信息和暂停/恢复触摸功能。其中 UI 获取触摸信息的流程如图 3-5 所示。

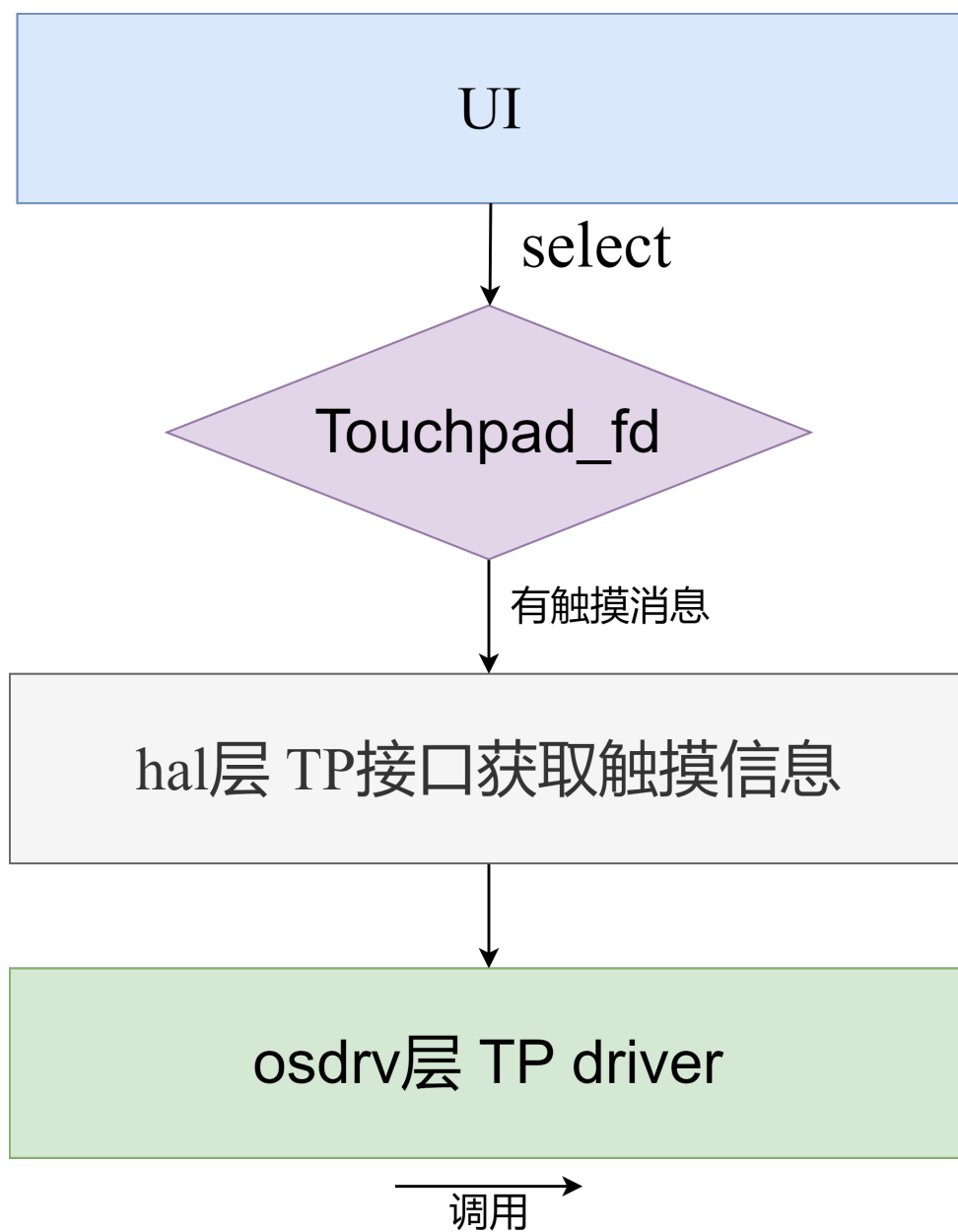


图 3.5: UI 获取触摸信息流程图

3.2.3.3 API 参考

该功能模块为用户提供以下 API:

- `CVI_HAL_TOUCHPAD_Init`: 初始化 touchpad 管脚复用, 加载触摸屏驱动。
- `CVI_HAL_TOUCHPAD_Deinit`: 去初始化 touchpad 模块, 卸载触摸屏驱动。
- `CVI_HAL_TOUCHPAD_Suspend`: 用于暂停获取触摸信息。
- `CVI_HAL_TOUCHPAD_Resume`: 用户恢复获取触摸信息。
- `CVI_HAL_TOUCHPAD_Start`: 打开触摸屏文件描述, 并返回给上层使用 (上层用于 select 监听)。

- `CVI_HAL_TOUCHPAD_Stop`：关闭触摸屏文件描述。
- `CVI_HAL_TOUCHPAD_ReadInputEvent`：当检测到有触摸事件，调用该接口获取触摸信息。

3.2.3.3.1 CVI_HAL_TOUCHPAD_Init

【描述】

初始化 touchpad 管脚复用，加载触摸屏驱动。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_touchpad.h
- 库文件：libcvi_hal_touchpad.a/libcvi_hal_touchpad.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.3.3.2 CVI_HAL_TOUCHPAD_Deinit

【描述】

去初始化 touchpad 模块，卸载触摸屏驱动。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_touchpad.h
- 库文件: libcvi_hal_touchpad.a/libcvi_hal_touchpad.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.3.3.3 CVI_HAL_TOUCHPAD_Suspend

【描述】

用于暂停获取触摸信息。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Suspend(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_touchpad.h
- 库文件: libcvi_hal_touchpad.a/libcvi_hal_touchpad.so

【注意】

- 屏幕熄屏或者待机场景，调用该接口暂停获取触摸信息。
- 与 CVI_HAL_TOUCHPAD_Resume 配对使用。

【举例】

无。

3.2.3.3.4 CVI_HAL_TOUCHPAD_Resume

【描述】

用户恢复获取触摸信息。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Resume(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_touchpad.h
- 库文件: libcvi_hal_touchpad.a/libcvi_hal_touchpad.so

【注意】

- 与 CVI_HAL_TOUCHPAD_Suspend 配对使用。

【举例】

无。

3.2.3.3.5 CVI_HAL_TOUCHPAD_Start

【描述】

打开触摸屏文件描述，并返回给上层使用（上层用于 select 监听）。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Start(int32_t* ps32Fd);
```

【参数】

参数名称	描述	输入/输出
ps32Fd	触摸屏文件描述指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_touchpad.h`
- 库文件: `libcvi_hal_touchpad.a/libcvi_hal_touchpad.so`

【注意】

- 调用 `CVI_HAL_TOUCHPAD_Stop` 接口关闭文件描述符。

【举例】

无。

3.2.3.3.6 CVI_HAL_TOUCHPAD_Stop

【描述】

关闭触摸屏文件描述。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_Stop(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_touchpad.h`
- 库文件: `libcvi_hal_touchpad.a/libcvi_hal_touchpad.so`

【注意】

无。

【举例】

无。

3.2.3.3.7 CVI_HAL_TOUCHPAD_ReadInputEvent

【描述】

当检测到有触摸事件，调用该接口获取触摸信息。

【语法】

```
int32_t CVI_HAL_TOUCHPAD_ReadInputEvent(CVI_HAL_TOUCHPAD_INPUTINFO_S*  
→pstInputData);
```

【参数】

参数名称	描述	输入/输出
pstInputData	触摸信息结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_touchpad.h
- 库文件: libcvi_hal_touchpad.a/libcvi_hal_touchpad.so

【注意】

select 监听到有触摸事件，调用该接口。

【举例】

无。

3.2.3.4 数据类型

相关数据类型定义如下：

- CVI_HAL_TOUCHPAD_INPUTINFO_S: 触摸信息结构体。

3.2.3.4.1 CVI_HAL_TOUCHPAD_INPUTINFO_S

【说明】

定义触摸信息结构体。

【定义】

```
typedef struct cviHAL_TOUCHPAD_INPUTINFO_S
{
    int32_t s32ID;/**<input id info, one finger or two fingers*/
    int32_t s32X;/**<x coordinate absolute*/
    int32_t s32Y;/**<y coordinate absolute*/
    uint32_t u32Pressure;/**<is press on screen: 0, 1*/
    uint32_t u32TimeStamp;/**<time stamp*/
} CVI_HAL_TOUCHPAD_INPUTINFO_S;
```

【成员】

成员名称	描述
s32ID	消息 ID，单点触摸为 0；多点触摸表示多点序号
s32X	触摸 x 位置信息，单位为 pixel
s32Y	触摸 y 位置信息，单位为 pixel
u32Pressure	触摸压力
u32TimeStamp	触摸时间戳

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.3.5 客制化场景修改指南

注解： 下面介绍如何添加新的 TP 模块（确保硬件已经妥当）。

以 GT9XX TP 驱动为例，添加具体步骤如下：

步骤 1 引脚配置

在 `cvi_board_init.c` 配置 TP 引脚，请用户根据硬件原理图合理配置引脚

```
PINMUX_CONFIG(VIVO_D1, XGPIOB_20); // pinmux , function
PINMUX_CONFIG(VIVO_D0, XGPIOB_21);
PINMUX_CONFIG(IIC0_SCL, XGPIOA_28);
PINMUX_CONFIG(IIC0_SDA, XGPIOA_29);
```

步骤 2 dts 配置

以 `cv1811h_wevb_0007a_spinor` 为例，在 `cv1811h_wevb_0007a_spinor.dts` 配置

```
/ {
    i2c1gpio: i2c1-gpio {
        compatible = "i2c-gpio";
        // 设置sda使用的gpio
        sda-gpios = <&portb 21 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
        // 设置scl使用的gpio
```

(下页继续)

(续上页)

```
scl-gpios = <&portb 20 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
#size-cells = <0x0>;
#address-cells = <0x1>;
i2c-gpio,delay-us = <5>;
status = "okay";
gt9xx: gt9xx@5d {
    compatible = "goodix,gt9xx";
    reg = <0x5d>;
    status = "okay";
};
};
};
```

步骤 3 驱动安装

- 1、当前在 platform/osdrv/tp 下默认包含 GT9XX 的驱动文件
- 2、修改驱动文件，platform/osdrv/tp/gt9xx.h

```
// GPIO值可以对应cpsl/hal/gpio/include/cvi_hal_gpio.h
#define GTP_RST_PORT 508 //配置RST脚的GPIO
#define GTP_INT_PORT 509 //配置INT脚的GPIO
```

3.2.4 显示屏幕

3.2.4.1 概述

显示屏幕（LCD）模块提供屏幕相关功能接口，调用者为上层应用模块（如 init、状态机、ui），上下文依赖关系如图 3-6 所示。

当前系统框架为 Linux+Alios 双系统架构，媒体业务部署在 Alios 端：

- 用户交互业务在 Linux 端，屏幕亮度、熄屏/亮屏等屏幕动态属性在 Linux 端提供接口。
- 由于上电快速预览，在 Alios 也提供接口。
- 当前屏幕 HAL 层包含 Linux 和 Alios 两套控制 LCD 的接口。

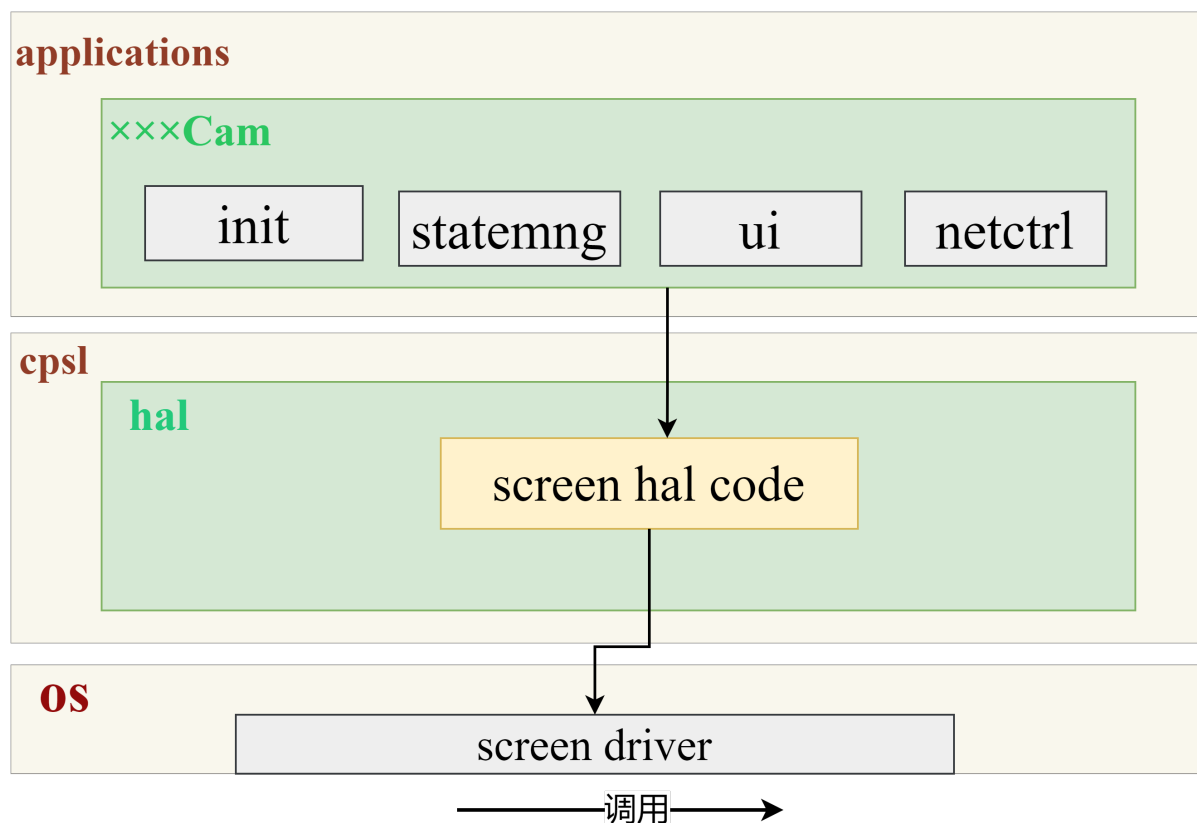


图 3.6: LCD 模块上下文

3.2.4.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 LCD 开关和型号选择。顶层目录引用 screen 目录下 kconfig。

3.2.4.3 API 参考

该功能模块为用户提供以下 API:

- **CVI_HAL_SCREEN_Init**: 初始化 LCD 管脚复用, 加载 LCD 驱动。
- **CVI_HAL_SCREEN_Deinit**: 去初始化 LCD 模块, 卸载 LCD 驱动。
- **CVI_HAL_SCREEN_Register**: LCD 注册, 注册选定型号的 LCD。
- **CVI_HAL_SCREEN_GetAttr**: 获取 LCD 属性和 VO 时序、时钟参数。
- **CVI_HAL_SCREEN_GetDisplayState**: 获取 LCD 显示状态。
- **CVI_HAL_SCREEN_SetDisplayState**: 设置 LCD 显示状态。
- **CVI_HAL_SCREEN_GetBackLightState**: 获取 LCD 背光状态。
- **CVI_HAL_SCREEN_SetBackLightState**: 设置 LCD 背光状态。
- **CVI_HAL_SCREEN_GetLuma**: 获取 LCD 亮度。
- **CVI_HAL_SCREEN_SetLuma**: 设置 LCD 亮度。

- CVI_HAL_SCREEN_GetSaturation : 获得 LCD 饱和度。
- CVI_HAL_SCREEN_SetSaturation : 设置 LCD 饱和度。
- CVI_HAL_SCREEN_GetContrast : 获得 LCD 对比度。
- CVI_HAL_SCREEN_SetContrast : 设置 LCD 对比度。

3.2.4.3.1 CVI_HAL_SCREEN_Init

【描述】

初始化 LCD 管脚复用，加载 LCD 驱动。

【语法】

```
int32_t CVI_HAL_SCREEN_Init(CVI_HAL_SCREEN_IDX_E enScreenIndex);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.4.3.2 CVI_HAL_SCREEN_Deinit

【描述】

去初始化 LCD 模块，卸载 LCD 驱动。

【语法】

```
int32_t CVI_HAL_SCREEN_Deinit(CVI_HAL_SCREEN_IDX_E enScreenIndex);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号， [0, CVI_HAL_SCREEN_IDX_BUTT)	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_screen.h
- 库文件：libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.4.3.3 CVI_HAL_SCREEN_Register

【描述】

LCD 注册，注册选定型号的 LCD。

【语法】

```
int32_t CVI_HAL_SCREEN_Register(CVI_HAL_SCREEN_IDX_E enScreenIndex, const CVI_
↪HAL_SCREEN_OBJ_S *pstScreenObj);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号， [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pstScreenObj	LCD 对象指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

初始化注册一次，不支持重复注册。

【举例】

无。

3.2.4.3.4 CVI_HAL_SCREEN_GetAttr

【描述】

获取 LCD 属性和 VO 时序、时钟参数。

【语法】

```
int32_t CVI_HAL_SCREEN_GetAttr(CVI_HAL_SCREEN_IDX_E enScreenIndex, CVI_HAL_SCREEN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pstAttr	LCD 属性结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

获取屏幕属性用于 VO 适配 LCD 屏。

【举例】

无。

3.2.4.3.5 CVI_HAL_SCREEN_GetDisplayState

【描述】

获取 LCD 显示状态。

【语法】

```
int32_t CVI_HAL_SCREEN_GetDisplayState(CVI_HAL_SCREEN_IDX_E enScreenIndex, CVI_HAL_SCREEN_STATE_E *penDisplayState);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
penDisplayState	LCD 显示状态指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

获取屏幕自身显示状态, 与屏幕背光不同。

【举例】

无。

3.2.4.3.6 CVI_HAL_SCREEN_SetDisplayState

【描述】

获取 LCD 显示状态。

【语法】

```
int32_t CVI_HAL_SCREEN_SetDisplayState(CVI_HAL_SCREEN_IDX_E enScreenIndex, CVI_HAL_SCREEN_STATE_E enDisplayState);
```


【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
penDisplayState	LCD 显示状态指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

设置屏幕自身显示状态, 与屏幕背光不同。

【举例】

无。

3.2.4.3.7 CVI_HAL_SCREEN_GetBackLightState

【描述】

获取 LCD 背光状态。

【语法】

```
int32_t CVI_HAL_SCREEN_GetBackLightState(CVI_HAL_SCREEN_IDX_E enScreenIndex,  
→CVI_HAL_SCREEN_STATE_E *penBackLightState);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
penBackLightState	LCD 背光状态状态指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.8 CVI_HAL_SCREEN_SetBackLightState

【描述】

设置 LCD 背光状态。

【语法】

```
int32_t CVI_HAL_SCREEN_SetBackLightState(CVI_HAL_SCREEN_IDX_E enScreenIndex,
↪CVI_HAL_SCREEN_STATE_E enBackLightState);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
penBackLightState	LCD 背光状态状态指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.9 CVI_HAL_SCREEN_GetLuma

【描述】

获取 LCD 亮度。

【语法】

```
CVI_HAL_SCREEN_PWM_S CVI_HAL_SCREEN_GetLuma(CVI_HAL_SCREEN_IDX_E  
→enScreenIndex);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入

【返回值】

返回值	描述
非 NULL	成功
NULL	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.10 CVI_HAL_SCREEN_SetLuma

【描述】

设置 LCD 亮度。

【语法】

```
int32_t CVI_HAL_SCREEN_SetLuma(CVI_HAL_SCREEN_IDX_E enScreenIndex, CVI_HAL_  
→SCREEN_PWM_S pwmAttr);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pwmAttr	PWM 属性结构体	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.11 CVI_HAL_SCREEN_GetSaturature

【描述】

获得 LCD 饱和度。

【语法】

```
int32_t CVI_HAL_SCREEN_GetSaturature(CVI_HAL_SCREEN_IDX_E enScreenIndex, uint32_t *pu32Saturature);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pu32Saturature	饱和度指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.12 CVI_HAL_SCREEN_SetSaturature

【描述】

设置 LCD 饱和度。

【语法】

```
int32_t CVI_HAL_SCREEN_SetSaturature(CVI_HAL_SCREEN_IDX_E enScreenIndex, uint32_t u32Saturature);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
u32Saturature	饱和度指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.13 CVI_HAL_SCREEN_GetContrast

【描述】

获得 LCD 饱和度。

【语法】

```
int32_t CVI_HAL_SCREEN_GetContrast(CVI_HAL_SCREEN_IDX_E enScreenIndex, uint32_t *pu32Contrast);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pu32Contrast	对比度指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_screen.h
- 库文件: libcvi_hal_screen.a/libcvi_hal_screen.so

【注意】

无。

【举例】

无。

3.2.4.3.14 CVI_HAL_SCREEN_SetContrast

【描述】

获得 LCD 对比度。

【语法】

```
int32_t CVI_HAL_SCREEN_SetContrast(CVI_HAL_SCREEN_IDX_E enScreenIndex, uint32_t u32Contrast);
```

【参数】

参数名称	描述	输入/输出
enScreenIndex	屏幕 ID 号, [0, CVI_HAL_SCREEN_IDX_BUTT)	输入
pu32Contrast	对比度指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_screen.h`
- 库文件: `libcvi_hal_screen.a/libcvi_hal_screen.so`

【注意】

无。

【举例】

无。

3.2.4.4 数据类型

相关数据类型定义如下：

- `CVI_HAL_SCREEN_IDX_E`: 定义屏幕 ID 枚举。
 - `CVI_HAL_SCREEN_TYPE_E`: 定义屏幕类型枚举。
 - `CVI_HAL_SCREEN_LCD_INTF_TYPE_E`: 定义 LCD 屏接口类型枚举。
 - `CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_E`: 定义 mipi 屏数据传输类型枚举。
 - `CVI_HAL_SCREEN_MIPI_VIDEO_MODE_E`: 定义 mipi 屏适配模式枚举。
 - `CVI_HAL_SCREEN_MIPI_VIDEO_FORMAT_E`: 定义 mipi 屏视频格式类型枚举。
 - `CVI_HAL_SCREEN_SYNC_ATTR_S`: 定义屏幕同步时序结构体。
 - `CVI_HAL_SCREEN_CLK_TYPE_E`: 定义 VO 设备时钟源类型枚举。
 - `CVI_HAL_SCREEN_CLK_PLL_S`: 定义 PLL 分频器配置结构体。
 - `CVI_HAL_SCREEN_CLK_ATTR_S`: 定义 VO 设备时钟配置结构体。
 - `CVI_HAL_SCREEN_COMMON_ATTR_S`: 定义屏幕通用属性结构体。
 - `CVI_HAL_SCREEN_MIPI_ATTR_S`: 定义 mipi 屏属性结构体。
 - `CVI_HAL_SCREEN_LCD_ATTR_S`: 定义 LCD 屏属性结构体。
 - `CVI_HAL_SCREEN_ATTR_S`: 定义屏幕属性结构体。
 - `CVI_HAL_SCREEN_LCD_ATTR_S`: 定义屏幕 PWM 结构体。
 - `CVI_HAL_SCREEN_STATE_E`: 定义屏幕状态枚举。
 - `CVI_HAL_SCREEN_OBJ_S`: 定义屏幕对象结构体。
-

3.2.4.4.1 CVI_HAL_SCREEN_IDX_E

【说明】

定义屏幕 ID 枚举。

【定义】

```
typedef enum cviHAL_SCREEN_IDX_E {  
    CVI_HAL_SCREEN_IDX_0 = 0,  
    CVI_HAL_SCREEN_IDX_1,  
    CVI_HAL_SCREEN_IDX_BUTT  
} CVI_HAL_SCREEN_IDX_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_IDX_0	ID 为 0 的屏幕
CVI_HAL_SCREEN_IDX_1	ID 为 1 的屏幕
CVI_HAL_SCREEN_IDX_BUTT	最大屏幕 ID+1

【注意事项】

只定义了两个屏，支持用户扩展多个屏。

【相关数据类型及接口】

无。

3.2.4.4.2 CVI_HAL_SCREEN_TYPE_E

【说明】

定义屏幕类型枚举。

【定义】

```
typedef enum cviHAL_SCREEN_TYPE_E {  
    CVI_HAL_SCREEN_INTF_TYPE_LCD = 0, /**<RGB interface type, such as lcd_6bit lcd_  
→8bit, parallel communication protocol*/  
    /**<MIPI interface type, serial communication protocol*/  
    CVI_HAL_SCREEN_INTF_TYPE_MIPI,  
    CVI_HAL_SCREEN_INTF_TYPE_I80,  
    CVI_HAL_SCREEN_INTF_TYPE_BUIT  
} CVI_HAL_SCREEN_TYPE_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_INTF_TYPE_LCD	RGB 接口类型 LCD，比如 6 bit LCD、8bit LCD、16bit LCD 等
CVI_HAL_SCREEN_INTF_TYPE_MIPI	MIPI DSI 接口类型
CVI_HAL_SCREEN_INTF_TYPE_I80	I80 接口类型
CVI_HAL_SCREEN_INTF_TYPE_BUIT	最大屏幕类型 + 1

【注意事项】

- RGB 类型接口 LCD，视频数据通过 RGB 类型接口传输，配置信息一般通过 I2C 或者 SPI 接口传输。
- MIPI DSI 接口类型 LCD 视频数据和配置报文都走 MIPI DSI 接口传输。

【相关数据类型及接口】

无。

3.2.4.4.3 CVI_HAL_SCREEN_LCD_INTF_TYPE_E

【说明】

定义 LCD 屏接口类型枚举。

【定义】

```
typedef enum cviHAL_SCREEN_LCD_INTF_TYPE_E {  
    CVI_HAL_SCREEN_LCD_INTF_6BIT = 0, /**<6bit intf type*/  
    CVI_HAL_SCREEN_LCD_INTF_8BIT,  
    CVI_HAL_SCREEN_LCD_INTF_16BIT,  
    CVI_HAL_SCREEN_LCD_INTF_24BIT,  
    CVI_HAL_SCREEN_LCD_INTF_BUIT  
} CVI_HAL_SCREEN_LCD_INTF_TYPE_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_LCD_INTF_6BIT	一个时钟传输 6bit 视频数据
CVI_HAL_SCREEN_LCD_INTF_8BIT	一个时钟传输 8bit 视频数据
CVI_HAL_SCREEN_LCD_INTF_16BIT	一个时钟传输 16bit 视频数据
CVI_HAL_SCREEN_LCD_INTF_24BIT	一个时钟传输 24bit 视频数据
CVI_HAL_SCREEN_LCD_INTF_BUIT	LCD 屏接口类型最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.4 CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_E

【说明】

定义 mipi 屏数据传输类型枚举。

【定义】

```
typedef enum cviHAL_SCREEN_MIPI_OUTPUT_TYPE_E {  
    CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_CMD = 0x0,  
    CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_VIDEO,  
    CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_BUIT  
} CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_CMD	MIPI传输类型为 CMD
CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_VIDEO	MIPI传输类型为 VIDEO
CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_BUIT	MIPI传输类型最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.5 CVI_HAL_SCREEN_MIPI_VIDEO_MODE_E

【说明】

定义 mipi 屏适配模式枚举。

【定义】

```
typedef enum cviHAL_SCREEN_MIPI_VIDEO_MODE_E {  
    CVI_HAL_SCREEN_MIPI_VIDEO_MODE_BURST = 0x0,/**<Burst Mode*/  
    CVI_HAL_SCREEN_MIPI_VIDEO_MODE_PULSES,/**<Non-Burst Mode with Sync Pulses*/  
    CVI_HAL_SCREEN_MIPI_VIDEO_MODE_EVENTS,/**<Non-Burst Mode with Sync Events*/  
    CVI_HAL_SCREEN_MIPI_VIDEO_MODE_BUIT  
} CVI_HAL_SCREEN_MIPI_VIDEO_MODE_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_MIPI_VIDEO_MODE_BURST	MIPI视频模式为 Burst Mode
CVI_HAL_SCREEN_MIPI_VIDEO_MODE_PULSES	MIPI视频模式为 Non-Burst Mode with Sync Pulses
CVI_HAL_SCREEN_MIPI_VIDEO_MODE_EVENTS	MIPI视频模式为 Non-Burst Mode with Sync Pulses
CVI_HAL_SCREEN_MIPI_VIDEO_MODE_BUIT	MIPI视频模式最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.6 CVI_HAL_SCREEN_MIPI_VIDEO_FORMAT_E

【说明】

定义 mipi 屏视频格式类型枚举。

【定义】

```
typedef enum cviHAL_SCREEN_MIPI_VIDEO_FORMAT_E {
    CVI_HAL_SCREEN_MIPI_VIDEO_RGB_16BIT = 0x0,
    CVI_HAL_SCREEN_MIPI_VIDEO_RGB_18BIT,
    CVI_HAL_SCREEN_MIPI_VIDEO_RGB_24BIT,
    CVI_HAL_SCREEN_MIPI_VIDEO_BUIT
} CVI_HAL_SCREEN_MIPI_VIDEO_FORMAT_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_MIPI_VIDEO_RGB_16BIT	16bit 视频格式为 RGB 16bit, 如 RGB565
CVI_HAL_SCREEN_MIPI_VIDEO_RGB_18BIT	18bit 视频格式为 RGB 18bit, 如 RGB666
CVI_HAL_SCREEN_MIPI_VIDEO_RGB_24BIT	24bit 视频格式为 RGB 24bit, 如 RGB888
CVI_HAL_SCREEN_MIPI_VIDEO_MODE_BUIT	视频格式最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.7 CVI_HAL_SCREEN_SYNC_ATTR_S

【说明】

定义屏幕同步时序结构体。

【定义】

```
typedef struct cviHAL_SCREEN_SYNC_ATTR_S {
    uint16_t u16Vact; /* vertical active area */
    uint16_t u16Vbb; /* vertical back blank porch */
    uint16_t u16Vfb; /* vertical front blank porch */
    uint16_t u16Hact; /* herizontal active area */
    uint16_t u16Hbb; /* herizontal back blank porch */
    uint16_t u16Hfb; /* herizontal front blank porch */

    uint16_t u16Hpw; /* horizontal pulse width */
    uint16_t u16Vpw; /* vertical pulse width */

    bool bIdv; /*< data Level polarity,0 mean high level valid,default 0,can not config*/
    bool bIhs; /*< horizon Level polarity,0 mean high level valid*/
    bool bIvs; /*< vertical Level polarity,0 mean high level valid*/
} CVI_HAL_SCREEN_SYNC_ATTR_S;
```

【成员】

成员名称	描述
u16Vact	垂直有效区，隔行输出时表示顶场垂直有效区。单位：行
u16Vbb	垂直消隐后肩，隔行输出时表示顶场垂直消隐后肩。单位：行
u16Vfb	垂直消隐前肩，隔行输出时表示顶场垂直消隐前肩。单位：行
u16Hact	水平有效区。单位：像素
u16Hbb	水平消隐后肩。单位：像素
u16Hfb	水平消隐前肩。单位：像素
u16Hpw	水平同步信号的宽度。单位：像素
u16Vpw	垂直同步信号的宽度。单位：行
bIdv	数据有效信号的极性。目前不可调，默认高有效
bIhs	水平有效信号的极性，配置 0 为高有效，配置 1 为低有效
bIvs	垂直有效信号的极性，配置 0 为高有效，配置 1 为低有效

【注意事项】

适配一款屏幕，属性参数就固定了，应用层获取属性参数配置 VO 时序，实现与屏幕适配。

【相关数据类型及接口】

无。

3.2.4.4.8 CVI_HAL_SCREEN_CLK_TYPE_E

【说明】

定义 VO 设备时钟源类型枚举。

【定义】

```
typedef enum cviHAL_SCREEN_CLK_TYPE_E {
    CVI_HAL_SCREEN_CLK_TYPE_PLL = 0x0,
    CVI_HAL_SCREEN_CLK_TYPE_LCDMCLK,
    CVI_HAL_SCREEN_CLK_TYPE_BUTT
} CVI_HAL_SCREEN_CLK_TYPE_E;
```

【成员】

成员名称	描述
CVI_HAL_SCREEN_CLK_TYPE_PLL	PLL 类型时钟源
CVI_HAL_SCREEN_CLK_TYPE_LCDMCLK	LCD 分频器时钟源
CVI_HAL_SCREEN_CLK_TYPE_BUTT	类型 +1

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_SCREEN_CLK_ATTR_S。

3.2.4.4.9 CVI_HAL_SCREEN_CLK_PLL_S

【说明】

定义 PLL 分频器配置结构体。

【定义】

```
typedef struct cviHAL_SCREEN_CLK_PLL_S {  
    uint32_t u32Fbdiv;  
    uint32_t u32Frac;  
    uint32_t u32Refdiv;  
    uint32_t u32Postdiv1;  
    uint32_t u32Postdiv2;  
} CVI_HAL_SCREEN_CLK_PLL_S;
```

【成员】

成员名称	描述
u32Fbdiv	PLL 整数倍频系数
u32Frac	PLL 小数分频系数
u32Refdiv	PLL 参考时钟分频系数
u32Postdiv1	PLL 第一级输出分频系数
u32Postdiv2	PLL 第二级输出分频系数

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_SCREEN_CLK_ATTR_S。

3.2.4.4.10 CVI_HAL_SCREEN_CLK_ATTR_S

【说明】

定义 VO 设备时钟配置结构体。

【定义】

```
typedef struct cviHAL_SCREEN_CLK_ATTR_S {  
    /**< vo clock reverse or not, if screen datasheet not mentioned, the value is true */  
    bool bClkReverse;  
    /**< vo clock division factor, RGB6&8 is 3,RGB16&18 is 1, MIPI DSI is 1 */  
    uint32_t u32DevDiv;  
    CVI_HAL_SCREEN_CLK_TYPE_E enClkType; /**< vo clk type, pll or lcdmlck*/  
}
```

(下页继续)

(续上页)

```

union {
    CVI_HAL_SCREEN_CLK_PLL_S stClkPll;
    uint32_t u32OutClk;
    /**< for serial:(vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*fps*total_clk_per_pixel,
        for parallel:(vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*fps */
};
} CVI_HAL_SCREEN_CLK_ATTR_S;

```

【成员】

成员名称	描述
bClkReverse	时钟相位是否反向，该值取决于屏幕
u32DevDiv	时钟分频系数
enClkType	时钟源类型
stClkPll	PLL 配置信息
u32OutClk	LCD 分频器时钟大小

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_SCREEN_COMMON_ATTR_S。

3.2.4.4.11 CVI_HAL_SCREEN_COMMON_ATTR_S

【说明】

定义屏幕通用属性结构体。

【定义】

```

typedef struct cviHAL_SCREEN_COMMON_ATTR_S {
    CVI_HAL_SCREEN_SYNC_ATTR_S stSynAttr;/**<screen sync attr*/
    uint32_t u32Width;
    uint32_t u32Height;
    uint32_t u32Framerate;
} CVI_HAL_SCREEN_COMMON_ATTR_S;

```

【成员】

成员名称	描述
stSynAttr	同步时序结构体
u32Width	屏幕显示宽度
u32Height	屏幕显示高度
u32Framerate	屏幕刷新帧率

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.12 CVI_HAL_SCREEN_MIPI_ATTR_S

【说明】

定义 mipi 屏属性结构体。

【定义】

```
typedef struct cviHAL_SCREEN_MIPI_ATTR_S {
    CVI_HAL_SCREEN_MIPI_OUTPUT_TYPE_E enType;
    CVI_HAL_SCREEN_MIPI_VIDEO_MODE_E enMode;
    CVI_HAL_SCREEN_MIPI_VIDEO_FORMAT_E enVideoFormat;
    /**<lane use: value is index from zero start, lane not use: value is -1 */
    int8_t as8LaneId[CVI_HAL_SCREEN_LANE_MAX_NUM];
    /**<mbps* (vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*total_bit_per_pixel/lane_num/
    →100000 */
    uint32_t u32PhyDataRate;
    uint32_t u32PixelClk; /**<KHz* (vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*fps/1000/
    →*/
} CVI_HAL_SCREEN_MIPI_ATTR_S;
```

【成员】

成员名称	描述
enType	视频传输类型
enMode	视频模式
enVideoFormat	视频数据格式
as8LaneId	lane 信息结构体，如果 lane 管脚没用置为-1，使用的 lane 需要连续，并从 0 开始赋值，比如使用 lane0 和 lane1 则结构体赋值为 [0,1,-1,-1]
u32PhyDataRate	单个 lane 的传输速率，计算公式 $(vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*total_bit_per_pixel/lane_num/100000$ ，单位为 mbps
u32PixelClk	像素时钟，计算公式为 $(vbp+vact+vfp+u16Vpw)*(hbp+hact+hfp+u16hpw)*fps/1000$ 单位为 khz

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_SCREEN_ATTR_S。

3.2.4.4.13 CVI_HAL_SCREEN_LCD_ATTR_S

【说明】

定义 LCD 屏属性结构体。

【定义】

```
typedef struct cviHAL_SCREEN_LCD_ATTR_S {  
    CVI_HAL_SCREEN_LCD_INTF_TYPE_E enType;  
    CVI_HAL_SCREEN_LCD_CFG_S stLcdCfg;  
} CVI_HAL_SCREEN_LCD_ATTR_S;
```

【成员】

成员名称	描述
enType	lcd 屏接口类型
stLcdCfg	lcd 配置

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.14 CVI_HAL_SCREEN_ATTR_S

【说明】

定义 LCD 屏属性结构体。

【定义】

```
typedef struct cviHAL_SCREEN_ATTR_S {  
    CVI_HAL_SCREEN_TYPE_E enType;  
    union tagHAL_SCREEN_ATTR_U {  
        CVI_HAL_SCREEN_LCD_ATTR_S stLcdAttr;  
        CVI_HAL_SCREEN_MIPI_ATTR_S stMipiAttr;  
        CVI_HAL_SCREEN_I80_ATTR_S stI80Attr;  
    } unScreenAttr;  
    CVI_HAL_SCREEN_COMMON_ATTR_S stAttr;  
} CVI_HAL_SCREEN_ATTR_S;
```

【成员】

成员名称	描述
enType	屏幕类型
unScreenAttr	屏幕属性共用体
stAttr	屏幕通用属性结构体

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.15 CVI_HAL_SCREEN_PWM_S

【说明】

定义屏幕 PWM 结构体。

【定义】

```
typedef struct cviHAL_SCREEN_PWM_S {  
    uint8_t group;    //组号  
    uint8_t channel;  //通道号  
    uint32_t period;  //周期  
    uint32_t duty_cycle; //占空比  
} CVI_HAL_SCREEN_PWM_S;
```

【成员】

成员名称	描述
group	组号
channel	通道号
period	周期
duty_cycle	占空比

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.16 CVI_HAL_SCREEN_STATE_E

【说明】

定义屏幕状态，有屏幕点亮和屏幕熄屏两种状态。

【定义】

```
typedef enum cviHAL_SCREEN_STATE_E {  
    CVI_HAL_SCREEN_STATE_OFF = 0, /**<screen off*/  
    CVI_HAL_SCREEN_STATE_ON,    /**<screen on*/  
    CVI_HAL_SCREEN_STATE_BUIT  
} CVI_HAL_SCREEN_STATE_E;
```

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.4.4.17 CVI_HAL_SCREEN_OBJ_S**【说明】**

定义屏幕对象结构体。

【定义】

```
typedef struct cviHAL_SCREEN_OBJ_S {  
    int32_t (*pfnInit)(void);  
    int32_t (*pfnGetAttr)(CVI_HAL_SCREEN_ATTR_S *pstAttr);  
    int32_t (*pfnGetDisplayState)(CVI_HAL_SCREEN_STATE_E *penDisplayState);  
    int32_t (*pfnSetDisplayState)(CVI_HAL_SCREEN_STATE_E enDisplayState);  
    int32_t (*pfnGetBackLightState)(CVI_HAL_SCREEN_STATE_E *penBackLightState);  
    int32_t (*pfnSetBackLightState)(CVI_HAL_SCREEN_STATE_E enBackLightState);  
    CVI_HAL_SCREEN_PWM_S (*pfnGetLuma)(void);  
    int32_t (*pfnSetLuma)(CVI_HAL_SCREEN_PWM_S pwmAttr);  
    int32_t (*pfnGetSaturation)(uint32_t *pu32Saturation);  
    int32_t (*pfnSetSaturation)(uint32_t u32Saturation);  
    int32_t (*pfnGetContrast)(uint32_t *pu32Contrast);  
    int32_t (*pfnSetContrast)(uint32_t u32Contrast);  
    int32_t (*pfnDeinit)(void);  
} CVI_HAL_SCREEN_OBJ_S;
```

【成员】

成员名称	描述
pfnInit	屏幕初始化函数指针，实现驱动加载和上下文初始化
pfnGetAttr	获取屏幕属性获取函数指针，获取屏幕自定义时序、接口类型等参数
pfnGetDisplayState	获取屏幕显示状态获取函数指针，获取屏幕显示状态
pfnSetDisplayState	设置屏幕显示状态获取函数指针，设置屏幕显示状态
pfnGetBackLightState	获取屏幕背光状态获取函数指针，获取屏幕背光状态
pfnSetBackLightState	设置屏幕背光状态获取函数指针，设置屏幕背光状态
pfnGetLuma	获取屏幕亮度获取函数指针，获取屏幕亮度
pfnSetLuma	设置屏幕亮度获取函数指针，设置屏幕亮度
pfnGetSaturation	获取屏幕饱和度获取函数指针，获取屏幕饱和度
pfnSetSaturation	设置屏幕饱和度获取函数指针，设置屏幕饱和度
pfnGetContrast	获取屏幕对比度获取函数指针，获取屏幕对比度
pfnSetContrast	设置屏幕对比度获取函数指针，设置屏幕对比度
pfnDeinit	屏幕去初始化，实现驱动卸载

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_SCREEN_Register。

3.2.4.5 客制化场景修改指南

注解： 下面介绍如何添加新增一款显示屏。

步骤 1 引脚配置

在 `cvi_board_init.c` 配置 `panel` 引脚，请用户根据硬件原理图合理配置引脚。

步骤 2 hal 层修改

- 1、仿照 `cpsl/hal/screen` 下面的文件夹，以新款屏幕来命名建立文件夹，并新建 `hal_screen.c` 和 `hal_screen.h`。
- 2、参考《屏幕对接使用指南》修改 `hal_screen.h` 中的结构体属性值。
- 3、修改 `cpsl/hal/screen` 下的 `kconfig`、`makefile` 和 `makefile.aliases`（以 I80ST7789 为例）。

`kconfig`

```
config SCREEN_I80ST7789
bool "Enable screen I80ST7789"
help
Say Y if you want to enable screen I80ST7789.
```

makefile

```
screen-$(CONFIG_SCREEN_I80ST7789)      += i80st7789

else ifeq ($(CONFIG_SCREEN_I80ST7789), y)
SCREEN_DRIVER_DIR      := i80st7789
```

makefile.aliOS

```
screen-$(CONFIG_SCREEN_I80ST7789)      += i80st7789

else ifeq ($(CONFIG_SCREEN_I80ST7789), y)
SCREEN_DRIVER_DIR      := i80st7789
```

步骤 3 配置修改

开机屏幕默认背光亮度（PWM）配置修改请参考 config_devmng.ini

```
[pwm_config]
enable      = 1 ; 0:disabled 1:enable
group       = 2 ; pwmchip0/4/8/12
channel     = 0 ; pwmchip0:pwm0~3 pwmchip1:pwm4~7
period      = 10000 ; unit:ns 100MHz
duty_cycle  = 8000 ;0 ~ 10000
```

3.2.5 重力加速度计

3.2.5.1 概述

重力加速度计（gsensor）模块提供碰撞检测相关功能接口，调用者为重力加速度计管理模块（gsensormng），上下文依赖关系如图 3-7 所示。

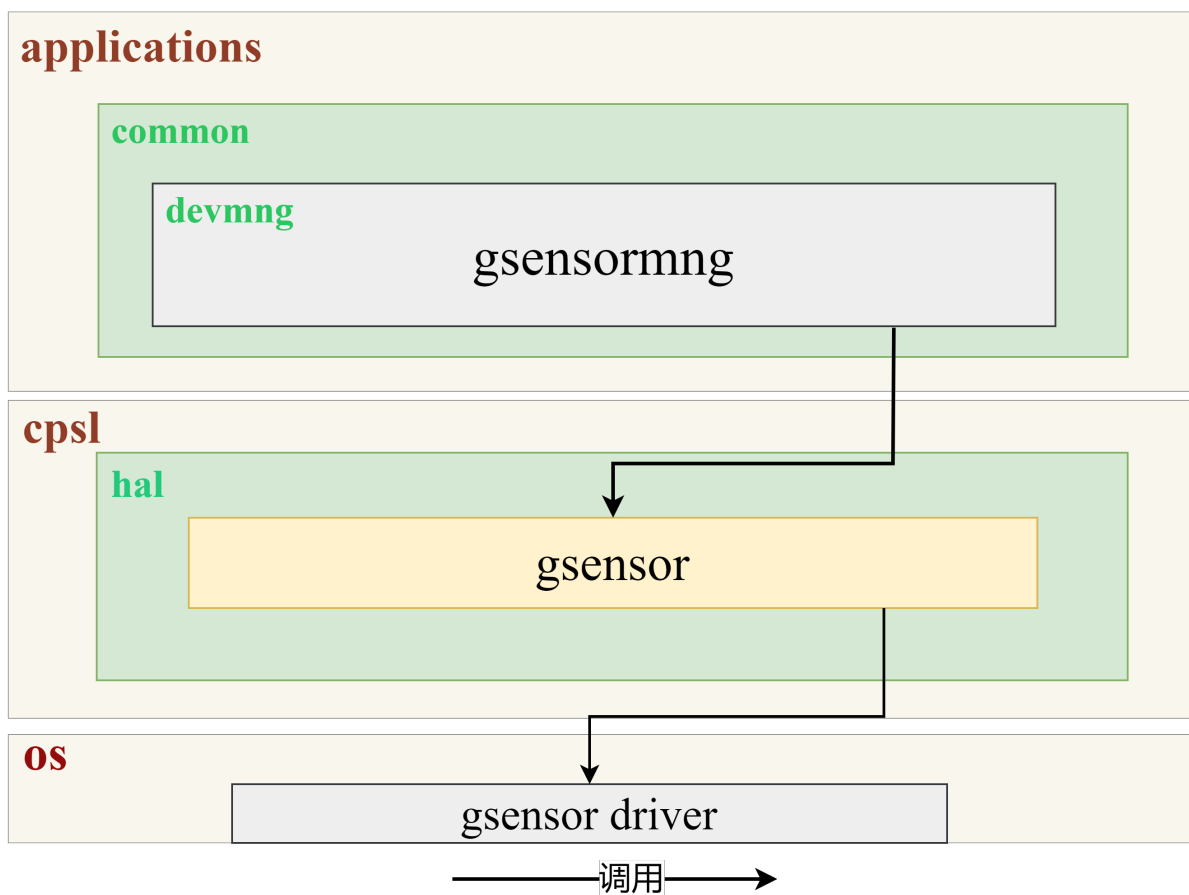


图 3.7: gsensor 模块上下文

3.2.5.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 `gsensor` 开关和 `gsensor` 型号选择。当 `gsensor` 功能开启后，对外提供是否发生碰撞查询接口。碰撞检测流程如图 3-8 所示。

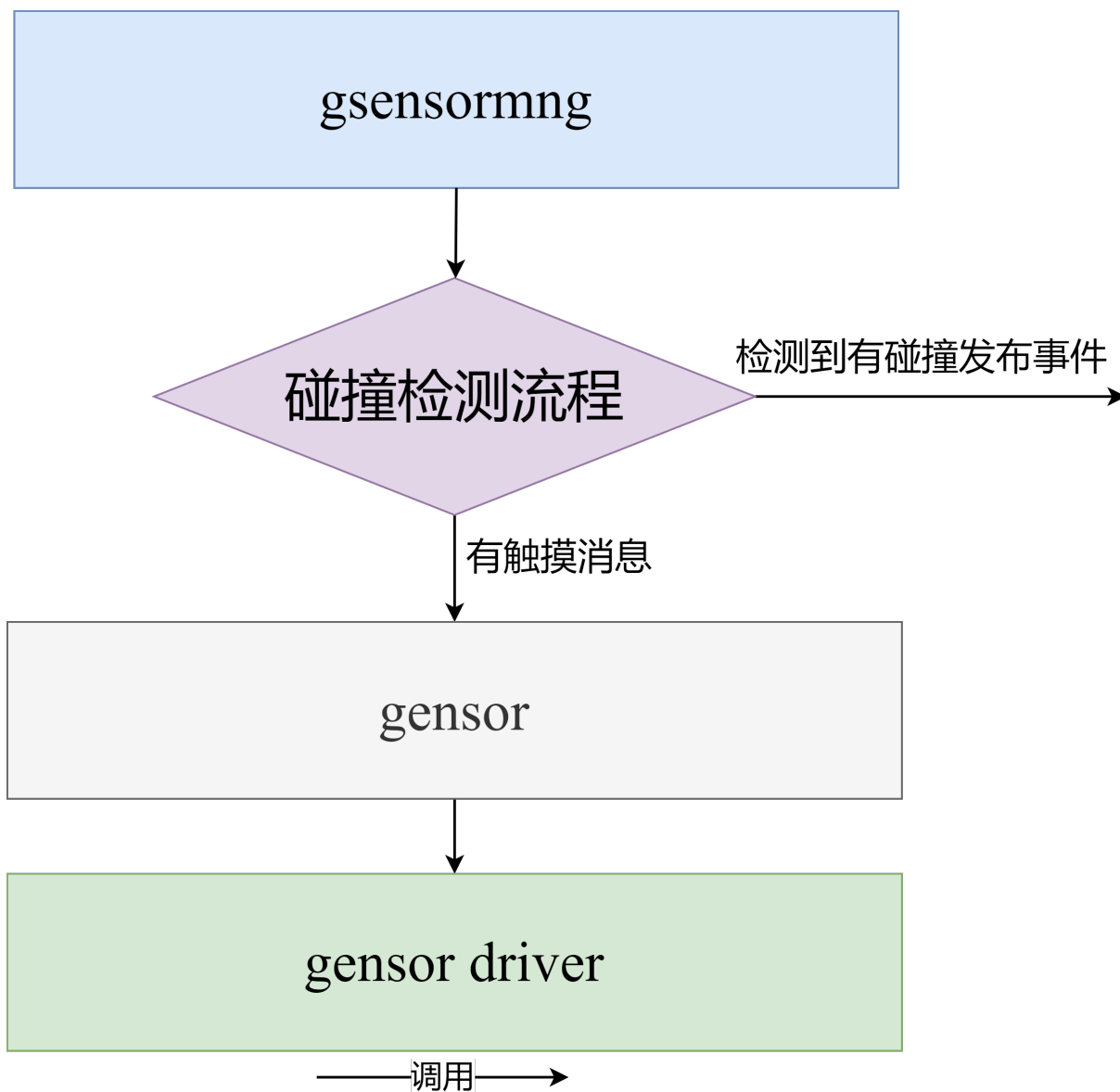


图 3.8: 碰撞检测流程图

3.2.5.3 API 参考

该功能模块为用户提供以下 API:

- CVI_HAL_GSENSOR_Init : 初始化 gsensor 管脚复用, 加载 gsensor 驱动。
- CVI_HAL_GSENSOR_DeInit : 去初始化 gsensro 模块, 卸载 gsensor 驱动。
- CVI_HAL_GSENSOR_SetSensitivity : 设置碰撞灵敏度。
- CVI_HAL_GSENSOR_SetAttr : 设置 gsensor 属性。
- CVI_HAL_GSENSOR_GetCurValue : 获取 gsensor 当前采集重力加速度值。
- CVI_HAL_GSENSOR_GetCollisionStatus : 获取 gsensor 是否检测到发生碰撞。
- CVI_HAL_GSENSOR_Register : gsensor 对象注册回调函数。

- CVI_HAL_GSNESOR_ReadInterrupt：读取中断状态。
- CVI_HAL_GSENSOR_OpenInterrupt：打开中断。

3.2.5.3.1 CVI_HAL_GSENSOR_Init

【描述】

初始化 WIFI 模块。

【语法】

```
int32_t CVI_HAL_GSENSOR_Init(const CVI_HAL_GSENSOR_CFG_S *pstCfg);
```

【参数】

参数名称	描述	输入/输出
pstCfg	gsensor 配置参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_gsenser.h
- 库文件：libcvi_hal_gsenser.a / libcvi_hal_gsenser.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.5.3.2 CVI_HAL_GSENSOR_DeInit

【描述】

去初始化 gsenser 模块，卸载 gsenser 驱动。

【语法】

```
int32_t CVI_HAL_GSENSOR_DeInit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_gsensor.h`
- 库文件: `libcvi_hal_gsensor.a` / `libcvi_hal_gsensor.so`

【注意】

不支持重复去初始化。

【举例】

无。

3.2.5.3.3 CVI_HAL_GSENSOR_SetSensitivity

【描述】

设置碰撞灵敏度。

【语法】

```
int32_t CVI_HAL_GSENSOR_SetSensitivity(CVI_HAL_GSENSOR_SENSITIVITY_E enSensitivity);
```

【参数】

参数名称	描述	输入/输出
<code>enSensitivity</code>	碰撞灵敏度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_gsensor.h`
- 库文件: `libcvi_hal_gsensor.a` / `libcvi_hal_gsensor.so`

【注意】

- 灵敏度越高, 越容易检测到有碰撞。
- 灵敏度为 off 表示关闭 gsensor 功能。

【举例】

无。

3.2.5.3.4 CVI_HAL_GSENSOR_SetAttr**【描述】**

设置碰撞灵敏度。

【语法】

```
int32_t CVI_HAL_GSENSOR_SetAttr(const CVI_HAL_GSENSOR_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
pstAttr	gsensor 属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensor.h
- 库文件: libcvi_hal_gsensor.a / libcvi_hal_gsensor.so

【注意】

无。

【举例】

无。

3.2.5.3.5 CVI_HAL_GSENSOR_GetCurValue**【描述】**

获取 gsensor 当前采集重力加速度值。

【语法】

```
int32_t CVI_HAL_GSENSOR_GetCurValue(CVI_HAL_GSENSOR_VALUE_S *pstCurValue);
```

【参数】

参数名称	描述	输入/输出
pstCurValue	gsensor x y z 三个方向当前检测值	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensorsensor.h
- 库文件: libcvi_hal_gsensorsensor.a / libcvi_hal_gsensorsensor.so

【注意】

无。

【举例】

无。

3.2.5.3.6 CVI_HAL_GSENSOR_GetCollisionStatus

【描述】

获取 gsensorsensor 是否检测到发生碰撞。

【语法】

```
int32_t CVI_HAL_GSENSOR_GetCollisionStatus(uint8_t *pbOnCollison);
```

【参数】

参数名称	描述	输入/输出
pbOnCollison	查询是否产生碰撞。1 表示产生碰撞；0 表示没有产生碰撞。	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensorsensor.h
- 库文件: libcvi_hal_gsensorsensor.a / libcvi_hal_gsensorsensor.so

【注意】

无。

【举例】

无。

3.2.5.3.7 CVI_HAL_GSENSOR_Register

【描述】

gsensor 对象注册回调函数。

【语法】

```
int32_t CVI_HAL_GSENSOR_Register(const CVI_HAL_GSENSOR_OBJ_S *pstGsensorObj);
```

【参数】

参数名称	描述	输入/输出
pstGsensorObj	gsensor 对象	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensor.h
- 库文件: libcvi_hal_gsensor.a / libcvi_hal_gsensor.so

【注意】

无。

【举例】

无。

3.2.5.3.8 CVI_HAL_GSNESOR_ReadInterrupt

【描述】

gsensor 读中断状态。

【语法】

```
int32_t CVI_HAL_GSNESOR_ReadInterrupt(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensorsensor.h
- 库文件: libcvi_hal_gsensorsensor.a / libcvi_hal_gsensorsensor.so

【注意】

无。

【举例】

无。

3.2.5.3.9 CVI_HAL_GSENSOR_OpenInterrupt**【描述】**

gsensor 打开中断。

【语法】

```
int32_t CVI_HAL_GSENSOR_OpenInterrupt(int32_t IntNum);
```

【参数】

参数名称	描述	输入/输出
IntNum	0 或者 1	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gsensorsensor.h
- 库文件: libcvi_hal_gsensorsensor.a / libcvi_hal_gsensorsensor.so

【注意】

无。

【举例】

无。

3.2.5.4 数据类型

相关数据类型定义如下：

- CVI_HAL_GSENSOR_SENSITIVITY_E：定义 gsensor 碰撞灵敏度枚举。
- CVI_HAL_GSENSOR_VALUE_S：定义 gsensor x y z 三个方向当前检测值。
- CVI_HAL_GSENSOR_ATTR_S：定义 gsensor 属性。
- CVI_HAL_GSENSOR_CFG_S：定义 gsensor 配置，用于初始化。
- CVI_HAL_GSENSOR_OBJ_S：定义 gsensor 对象。

3.2.5.4.1 CVI_HAL_GSENSOR_SENSITIVITY_E

【说明】

定义 gsensor 碰撞灵敏度。

【定义】

```
typedef enum cvHAL_GSENSOR_SENSITIVITY_E {
    CVI_HAL_GSENSOR_SENSITIVITY_OFF = 0, /**<gsensor off*/
    CVI_HAL_GSENSOR_SENSITIVITY_LOW,    /**<low sensitivity*/
    CVI_HAL_GSENSOR_SENSITIVITY_MIDDLE,  /**<middle sensitivity*/
    CVI_HAL_GSENSOR_SENSITIVITY_HIGH,    /**<high sensitivity*/
    CVI_HAL_GSENSOR_SENSITIVITY_BUIT
} CVI_HAL_GSENSOR_SENSITIVITY_E;
```

【成员】

成员名称	描述
CVI_HAL_GSENSOR_SENSITIVITY_OFF	关闭 gsensor
CVI_HAL_GSENSOR_SENSITIVITY_LOW	碰撞灵敏度低
CVI_HAL_GSENSOR_SENSITIVITY_MIDDLE	碰撞灵敏度中
CVI_HAL_GSENSOR_SENSITIVITY_HIGH	碰撞灵敏度高
CVI_HAL_GSENSOR_SENSITIVITY_BUIT	碰撞灵敏度最大值 +1

【注意事项】

若关闭碰撞检测功能，请使用 CVI_HAL_GSENSOR_SetSensitivity 传入参数 CVI_HAL_GSENSOR_SENSITIVITY_OFF 关闭。

【相关数据类型及接口】

CVI_HAL_GSENSOR_SetSensitivity。

3.2.5.4.2 CVI_HAL_GSENSOR_VALUE_S

【说明】

定义 gsensor 碰撞灵敏度。

【定义】

```
typedef struct cvHAL_GSENSOR_VALUE_S {  
    int16_t s16XDirValue; /**<x direction value,unit acceleration of gravity g*/  
    int16_t s16YDirValue; /**<y direction value,unit acceleration of gravity g*/  
    int16_t s16ZDirValue; /**<z direction value,unit acceleration of gravity g*/  
} CVI_HAL_GSENSOR_VALUE_S;
```

【成员】

成员名称	描述
s16XDirValue	重力加速度 x 方向采集值
s16YDirValue	重力加速度 y 方向采集值
s16ZDirValue	重力加速度 z 方向采集值

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_GSENSOR_GetCurValue。

3.2.5.4.3 CVI_HAL_GSENSOR_ATTR_S

【说明】

定义 gsensor 属性结构体。

【定义】

```
typedef struct cvHAL_GSENSOR_ATTR_S {  
    uint32_t u32SampleRate; /**<sample rate,0 mean Adopt default,not config,unit  
                                kps*/  
} CVI_HAL_GSENSOR_ATTR_S;
```

【成员】

成员名称	描述
u32SampleRate	gsensor 采样率,0 采用 gsensor 芯片值默认

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_GSENSOR_SetAttr。

3.2.5.4.4 CVI_HAL_GSENSOR_CFG_S

【说明】

定义 gsensor 配置，用于初始化。

【定义】

```
typedef struct cvHAL_GSENSOR_CFG_S {  
    CVI_HAL_GSENSOR_SENSITIVITY_E enSensitivity;  
    CVI_HAL_GSENSOR_ATTR_S stAttr;  
    int32_t busnum;  
} CVI_HAL_GSENSOR_CFG_S;
```

【成员】

成员名称	描述
enSensitivity	gsensor 碰撞灵敏度
stAttr	gsensor 采集属性
busnum	bus 总线

【注意事项】

无。

【相关数据类型及接口】

CVI_HAL_GSENSOR_Init。

3.2.5.4.5 CVI_HAL_GSENSOR_OBJ_S

【说明】

定义 gsensor 对象结构体。

【定义】

```
typedef struct cvHAL_GSENSOR_OBJ_S {  
    int32_t (*i2c_bus_init)(int32_t busname);  
    int32_t (*i2c_bus_deinit)(void);  
    int32_t (*init)(void);  
    int32_t (*deinit)(void);  
    int32_t (*read_data)(short *x, short *y, short *z);  
    int32_t (*set_sensitivity)(unsigned char sensitivity);  
    int32_t (*read_int_status)(unsigned char *int_status);  
    int32_t (*open_interrupt)(int32_t num);  
    int32_t (*read_interrupt)(void);  
} CVI_HAL_GSENSOR_OBJ_S;
```

【成员】

成员名称	描述
i2c_bus_init	i2c 总线初始化
i2c_bus_deinit	i2c 总线去初始化
init	gsensor 初始化
deinit	gsensor 去初始化
read_data	gsensor 读数据
set_sensitivity	gsensor 设置碰撞灵敏度
read_int_status	读数据
open_interrupt	打开中断
read_interrupt	读中断状态

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.5.5 客制化场景修改指南

注解： 下面介绍如何添加新增一款 gsensor。

以 SC7A20E gsensor 驱动为例，添加具体步骤如下：

步骤 1 引脚配置

在 `cvi_board_init.c` 配置 `panel` 引脚，请用户根据硬件原理图合理配置引脚

步骤 2 hal 层修改

- 1、在 `cpsl/hal/gsensor` 下添加 `SC7A20E` 目录，放置驱动文件
- 2、修改 `cpsl/hal/gsensor` 下的 `kconfig` 和 `makefile`

`kconfig`

```
config GSENSOR_SC7A20E
    bool "Enable GSENSOR sc7a20e"
    help
        Say Y if you want to enable gsensor sc7a20e.
```

`makefile`

```
gsensor-$(CONFIG_GSENSOR_SC7A20E) += sc7a20e
```


3.2.6 GPS

3.2.6.1 概述

GPS 模块开启后可以提供当前位置信息，调用者为 GPS 管理模块 (gpsmng)，上下文依赖关系如图 3-9 所示。

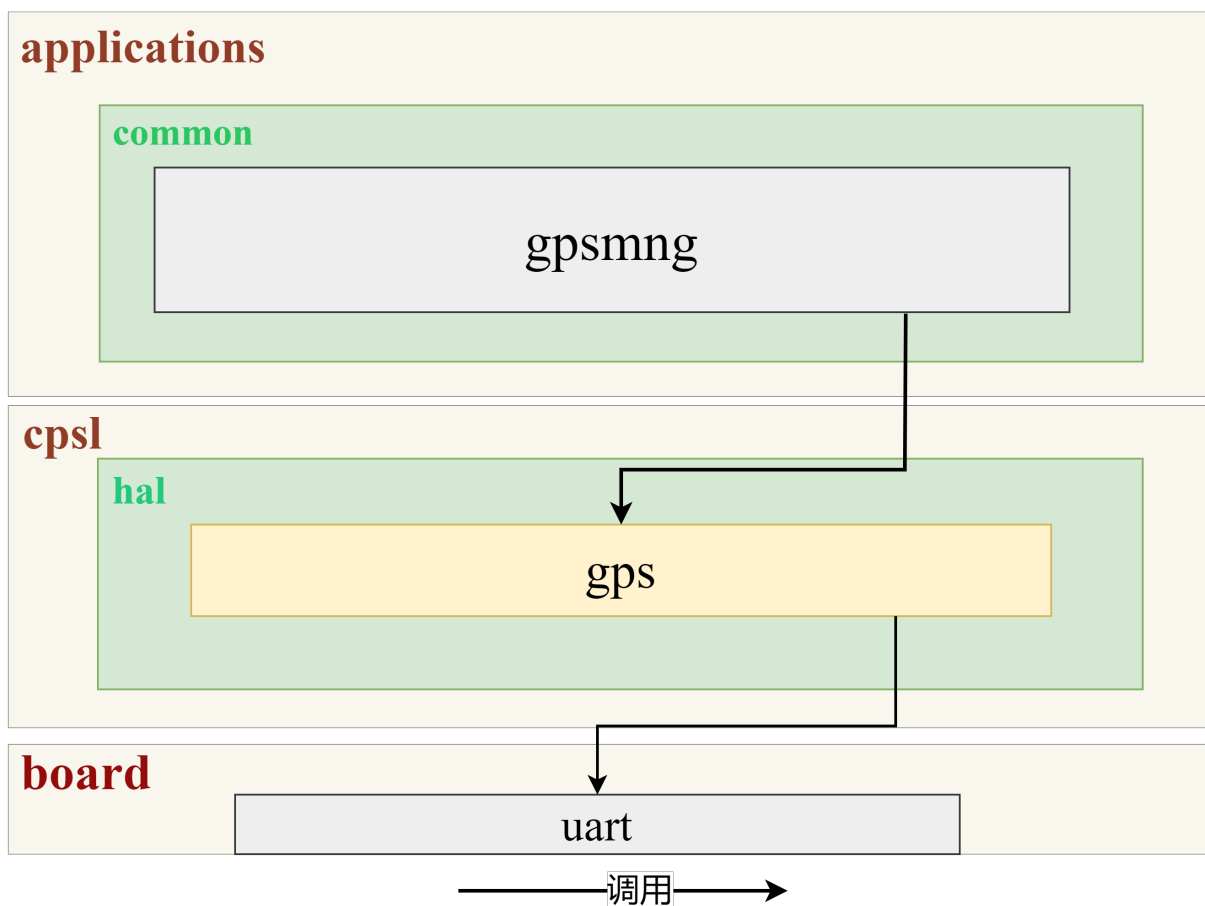


图 3.9: GPS 模块上下文

3.2.6.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 GPS 开关选项。

3.2.6.3 API 参考

该功能模块为用户提供以下 API:

- CVI_HAL_GPS_Init : GPS 模块初始化。
- CVI_HAL_GPS_Deinit : GPS 模块去初始化。
- CVI_HAL_GPS_GetRawData : GPS 模块获得行数据。

3.2.6.3.1 CVI_HAL_GPS_Init

【描述】

GPS 模块初始化。

【语法】

```
int32_t CVI_HAL_GPS_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gps.h
- 库文件: libcvi_hal_gps.a/libcvi_hal_gps.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.6.3.2 CVI_HAL_GPS_Deinit

【描述】

GPS 模块去初始化。

【语法】

```
int32_t CVI_HAL_GPS_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gps.h
- 库文件: libcvi_hal_gps.a/libcvi_hal_gps.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.6.3.3 CVI_HAL_GPS_GetRawData

【描述】

GPS 模块获得行数据。

【语法】

```
int32_t CVI_HAL_GPS_GetRawData(CVI_GPSDATA_S *gpsData, int32_t timeout_ms);
```

【参数】

参数名称	描述	输入/输出
gpsData	GPS 数据指针	输出
timeout_ms	等待阻塞时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gps.h
- 库文件: libcvi_hal_gps.a/libcvi_hal_gps.so

【注意】

无。

【举例】

无。

3.2.6.4 数据类型

相关数据类型定义如下：

- CVI_GPSDATA_S：定义 GPS 数据结构体。

3.2.6.4.1 CVI_GPSDATA_S

【说明】

定义 GPS 数据结构体。

【定义】

```
typedef struct cviGPSDATA
{
    uint32_t wantReadLen; /**want read length */
    uint32_t actualReadLen; /**actual read length */
    unsigned char rawData[CVI_HAL_GPS_DATA_SIZE];
} CVI_GPSDATA_S;
```

【成员】

成员名称	描述
wantReadLen	预期读长度
actualReadLen	实际读长度
rawData	行数据

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.6.5 客制化场景修改指南

3.2.7 LED

3.2.7.1 概述

LED 模块提供设置板端 LED 灯亮暗状态的接口，调用者为 ledmng，上下文依赖关系如图 3-10 所示。

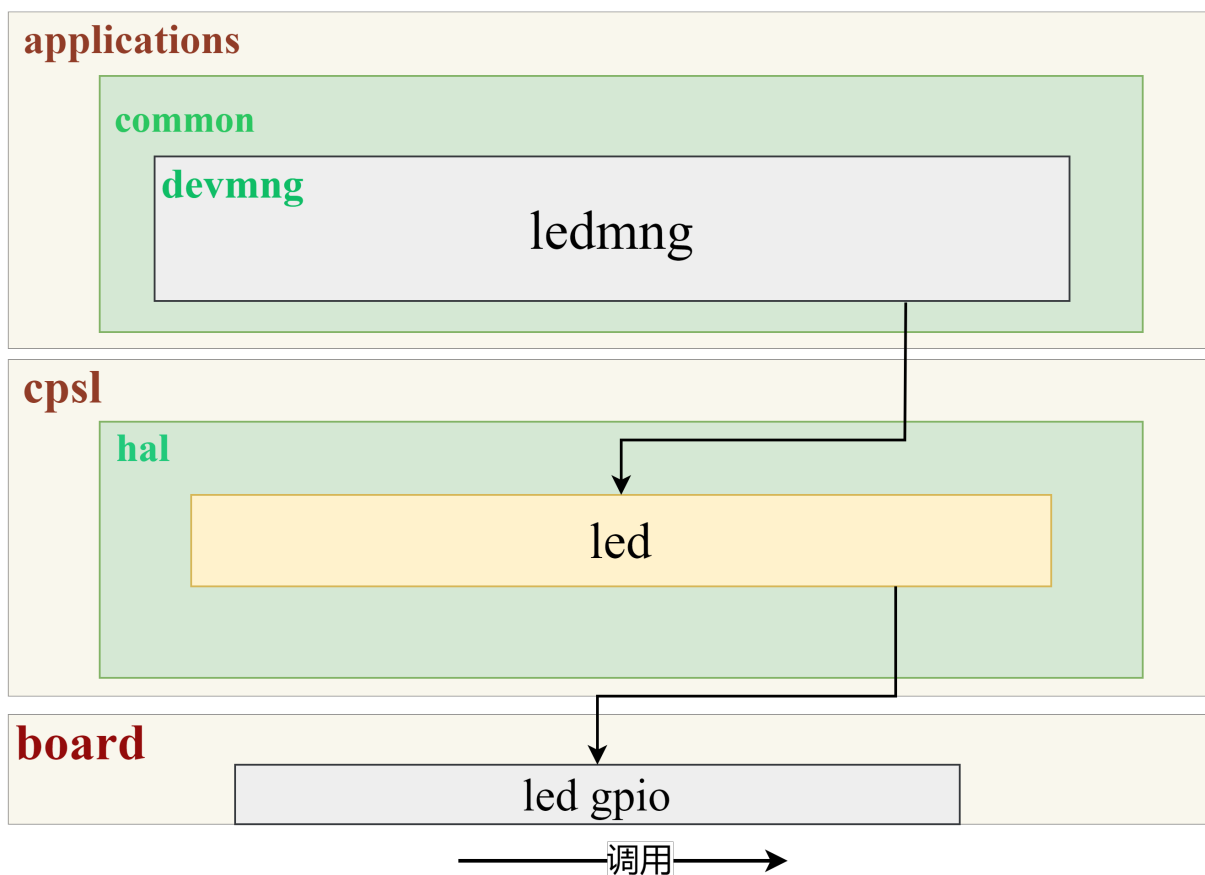


图 3.10: LED 模块上下文

3.2.7.2 功能开关

用户可以在 menuconfig 的 Peripheral options 选项中配置 led 开关。当 led 功能开启后，对外提供设置多个 led 等亮、暗、闪烁的功能。LED 亮暗流程如图 3-11 所示。

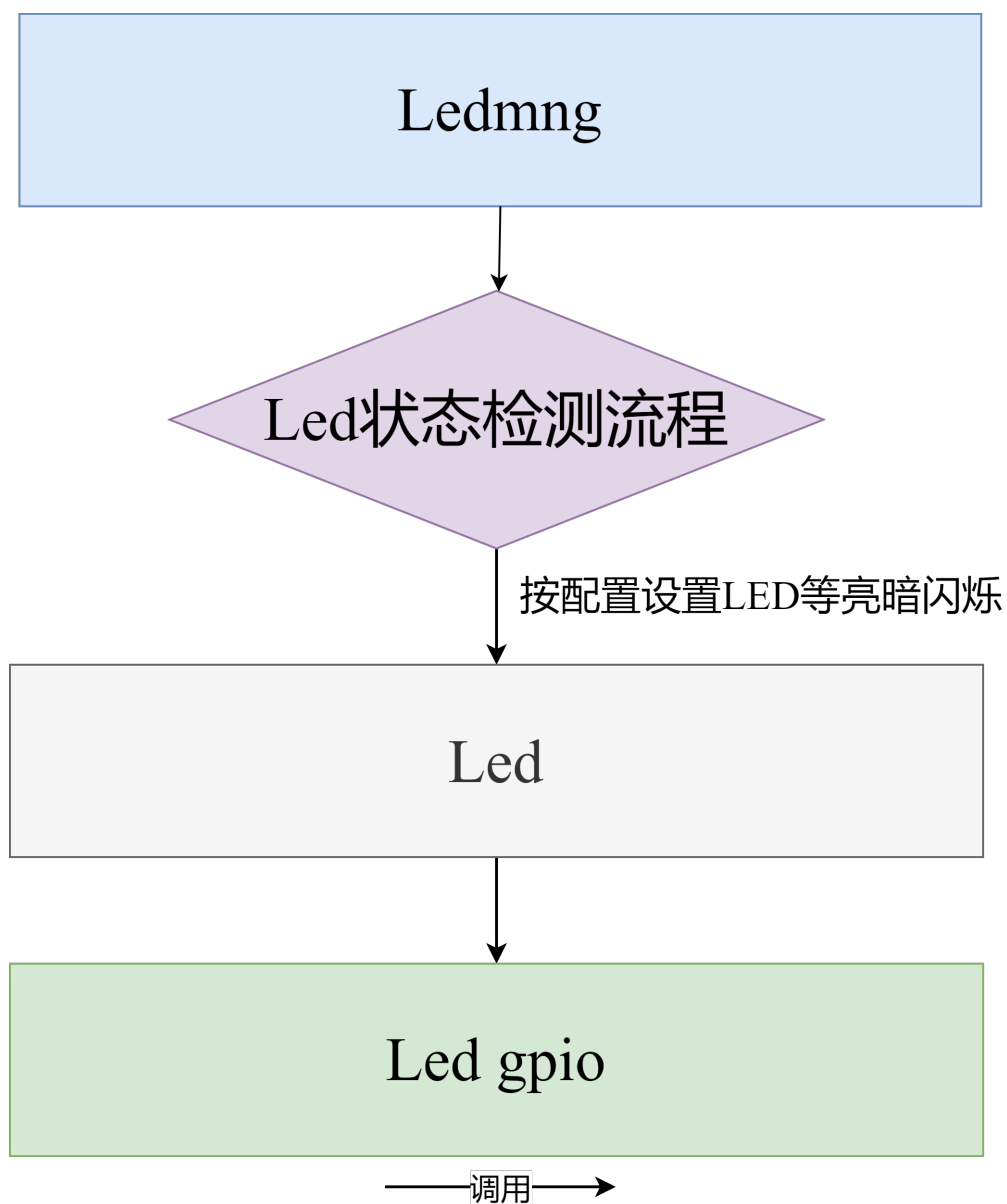


图 3.11: LED 亮暗流程图

3.2.7.3 API 参考

该功能模块为用户提供以下 API:

- CVI_HAL_LED_Init : 初始化 LED, 让 LED 处于初始状态。
- CVI_HAL_LED_GetState : 获得 LED 灯状态。
- CVI_HAL_LED_Deinit : 去初始化 LED, 让 LED 处于初始状态。
- CVI_HAL_LED_SetValue : 设置 LED GPIO 值。

3.2.7.3.1 CVI_HAL_LED_Init

【描述】

初始化 LED，让 LED 处于初始状态。

【语法】

```
int32_t CVI_HAL_LED_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_led.h
- 库文件：libcvi_hal_led.a / libcvi_hal_led.so

【注意】

无。

【举例】

无。

3.2.7.3.2 CVI_HAL_LED_GetState

【描述】

获得 LED 灯状态。

【语法】

```
int32_t CVI_HAL_LED_GetState(CVI_HAL_LED_IDX_E enKeyIdx, CVI_HAL_LED_STATE_E* penKeyState);
```

【参数】

参数名称	描述	输入/输出
enKeyIdx	LED 灯的索引	输入
penKeyState	LED 灯的状态，亮或者暗	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_led.h
- 库文件: libcvi_hal_led.a / libcvi_hal_led.so

【注意】

无。

【举例】

无。

3.2.7.3.3 CVI_HAL_LED_Deinit

【描述】

去初始化 LED，让 LED 处于初始状态。

【语法】

```
int32_t CVI_HAL_LED_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_led.h
- 库文件: libcvi_hal_led.a / libcvi_hal_led.so

【注意】

无。

【举例】

无。

3.2.7.3.4 CVI_HAL_LED_SetValue

【描述】

设置 LED GPIO 值。

【语法】

```
int32_t CVI_HAL_LED_SetValue(CVI_GPIO_NUM_E gpio, CVI_GPIO_VALUE_E value);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_led.h
- 库文件: libcvi_hal_led.a / libcvi_hal_led.so

【注意】

无。

【举例】

无。

3.2.7.4 数据类型

相关数据类型定义如下：

- CVI_HAL_LED_STATE_E：定义 LED 灯状态。
- CVI_HAL_LED_IDX_E：定义 LED 灯索引枚举。
- CVI_LED_GPIO_ID_E：定义 LED 灯的 GPIO 枚举。

3.2.7.4.1 CVI_HAL_LED_STATE_E

【说明】

定义 LED 灯状态。

【定义】

```
typedef enum cviHAL_LED_STATE_E
{
    CVI_HAL_LED_STATE_H = 0, /**<Led high state*/
    CVI_HAL_LED_STATE_L, /**<led low state*/
    CVI_HAL_LED_STATE_BUIT
} CVI_HAL_LED_STATE_E;
```

【成员】

成员名称	描述
CVI_HAL_LED_STATE_H	LED 高状态
CVI_HAL_LED_STATE_L	LED 低状态
CVI_HAL_LED_STATE_BUIT	状态最大值 +1

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.7.4.2 CVI_HAL_LED_IDX_E

【说明】

定义 LED 灯索引枚举。

【定义】

```
typedef enum cviHAL_LED_IDX_E
{
    CVI_HAL_LED_IDX_0 = 0, /**<led index 0*/
    CVI_HAL_LED_IDX_1,
    CVI_HAL_LED_IDX_BUIT
} CVI_HAL_LED_IDX_E;
```

【成员】

成员名称	描述
CVI_HAL_LED_IDX_0	LED 索引 0
CVI_HAL_LED_IDX_1	LED 索引 1
CVI_HAL_LED_IDX_BUIT	索引最大值 +1

【注意事项】

按照单板上预设的 LED 数量来增删 LED 的索引。

【相关数据类型及接口】

无。

3.2.7.4.3 CVI_LED_GPIO_ID_E

【说明】

定义 LED 灯对应的 GPIO。

【定义】

```
typedef struct cviGPIO_ID_E {  
    CVI_HAL_LED_IDX_E id;  
    int32_t gpioid;  
} CVI_LED_GPIO_ID_E;
```

【成员】

成员名称	描述
id	LED 索引 ID
gpioid	GPIO ID

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.7.5 客制化场景修改指南

注解： 下面介绍如何添加 LED。

步骤 1 引脚配置

在 `cvi_board_init.c` 配置 panel 引脚，请用户根据硬件原理图合理配置引脚

步骤 2 hal 层修改

1、修改 `cpsl/hal/led/src/cvi_hal_led.c`

```
static CVI_LED_GPIO_ID_E GPIOID[] = {{CVI_HAL_LED_IDX_0, CVI_GPIOA_28},  
                                       {CVI_HAL_LED_IDX_1, CVI_GPIOA_29}};
```

步骤 3 应用层修改

对应修改 `applications/common/devmng/src/cvi_ledmng.c`

3.2.8 看门狗

3.2.8.1 概述

看门狗（watchdog）模块提供异常复位功能接口，调用者为看门狗管理模块（watchdogmng）。上下文依赖关系如图 3-12 所示。

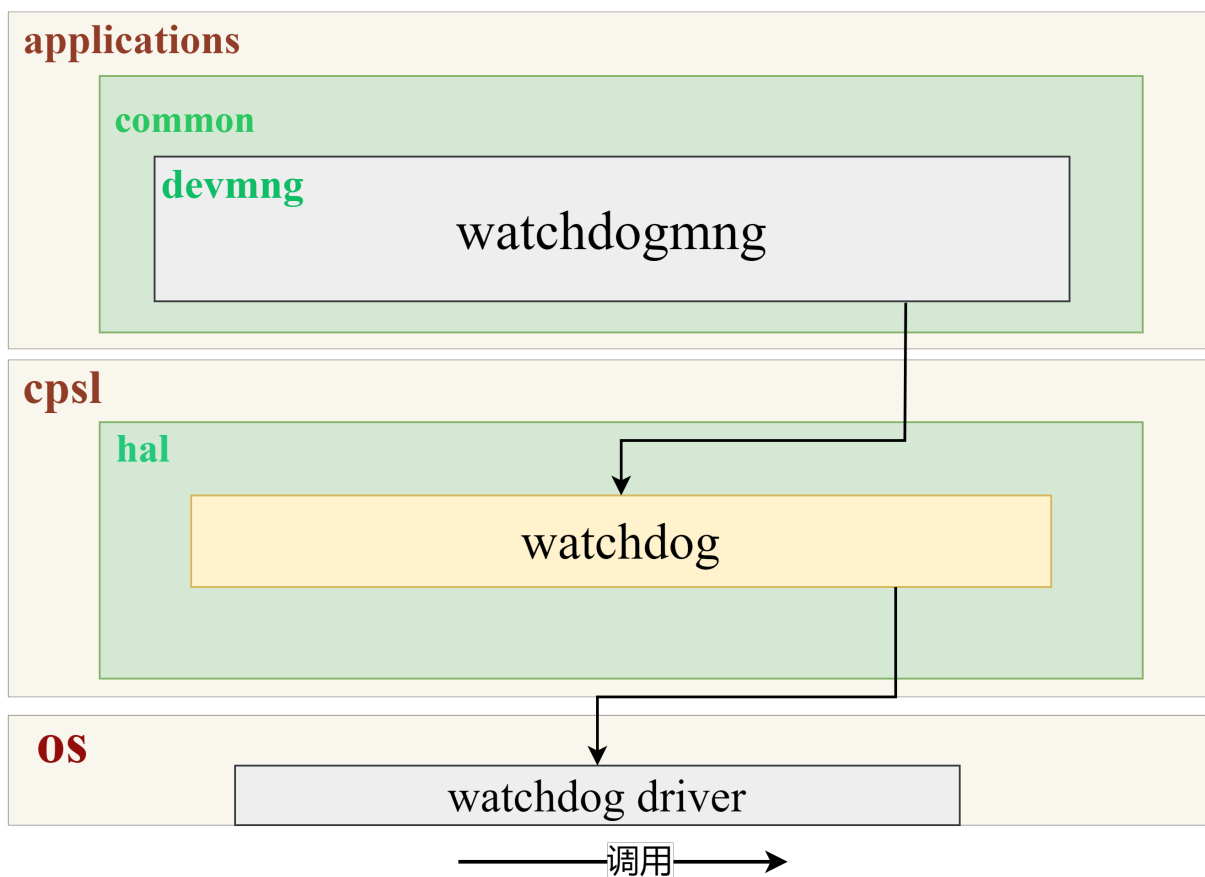


图 3.12: watchdog 模块上下文

3.2.8.2 功能开关

menuconfig 中 Peripheral options 选项中提供了开关选项。

当 watchdog 功能开启后，对外提供主进程挂死后系统复位的功能。流程如图 3-13 所示。

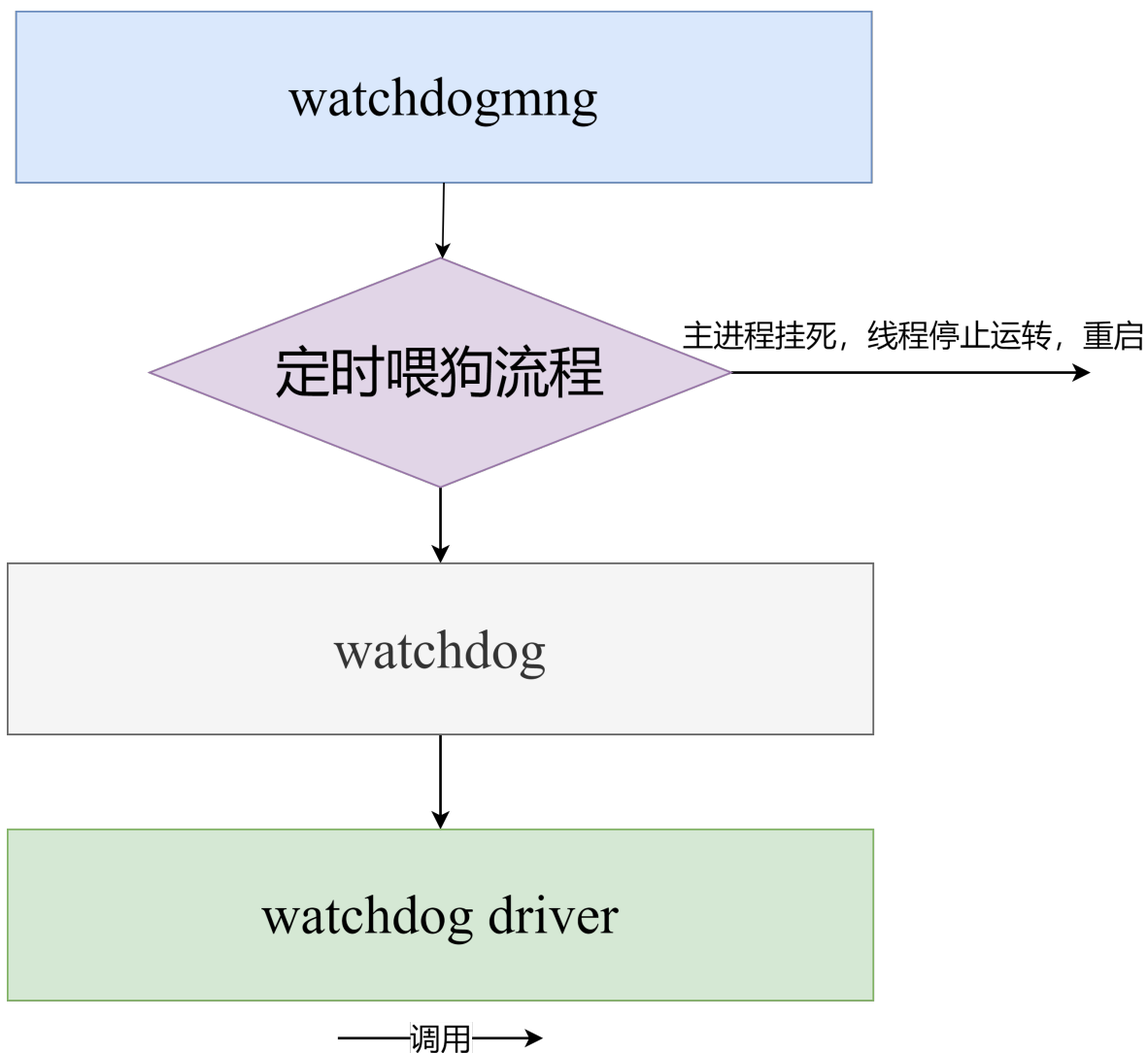


图 3.13: 看门狗流程图

3.2.8.3 API 参考

该功能模块为用户提供以下 API:

- `CVI_HAL_WATCHDOG_Init`: 初始化 watchdog, 加载 watchdog 驱动, 打开设备节点。
- `CVI_HAL_WATCHDOG_Deinit`: 去初始化 watchdog 模块, 卸载 watchdog 驱动。
- `CVI_HAL_WATCHDOG_Feed`: 喂狗接口。

3.2.8.3.1 CVI_HAL_WATCHDOG_Init

【描述】

初始化 watchdog，加载 watchdog 驱动，打开设备节点。

【语法】

```
int32_t CVI_HAL_WATCHDOG_Init(int32_t s32Time_s);
```

【参数】

参数名称	描述	输入/输出
s32Time_s	给驱动的喂狗间隔，如果没有按时喂狗，系统会重新复位	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_watchdog.h
- 库文件：libcvi_hal_watchdog.a/libcvi_hal_watchdog.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.8.3.2 CVI_HAL_WATCHDOG_Deinit

【描述】

初始化 watchdog 模块，卸载 watchdog 驱动。

【语法】

```
int32_t CVI_HAL_WATCHDOG_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_watchdog.h`
- 库文件: `libcvi_hal_watchdog.a/libcvi_hal_watchdog.so`

【注意】

不支持重复去初始化。

【举例】

无。

3.2.8.3.3 CVI_HAL_WATCHDOG_Feed

【描述】

喂狗接口。

【语法】

```
int32_t CVI_HAL_WATCHDOG_Feed(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_watchdog.h`
- 库文件: `libcvi_hal_watchdog.a/libcvi_hal_watchdog.so`

【注意】

当做了看门狗初始化后会主动喂狗, 间隔时间内如果没有调用 `CVI_HAL_WATCHDOG_Feed`, 系统会复位。

【举例】

无。

3.2.8.4 数据类型

相关数据类型、数据结构无。

3.2.8.5 客制化场景修改指南

menuconfig 选项选择打开即可使用。

3.2.9 电量计

3.2.9.1 概述

当 ADC 功能开启后，对外提供主动查询当前电池充电状态与电池电量，以及状态发生变化后抛事件的功能。电池充电状态和电量检测流程如图 3-14 所示。

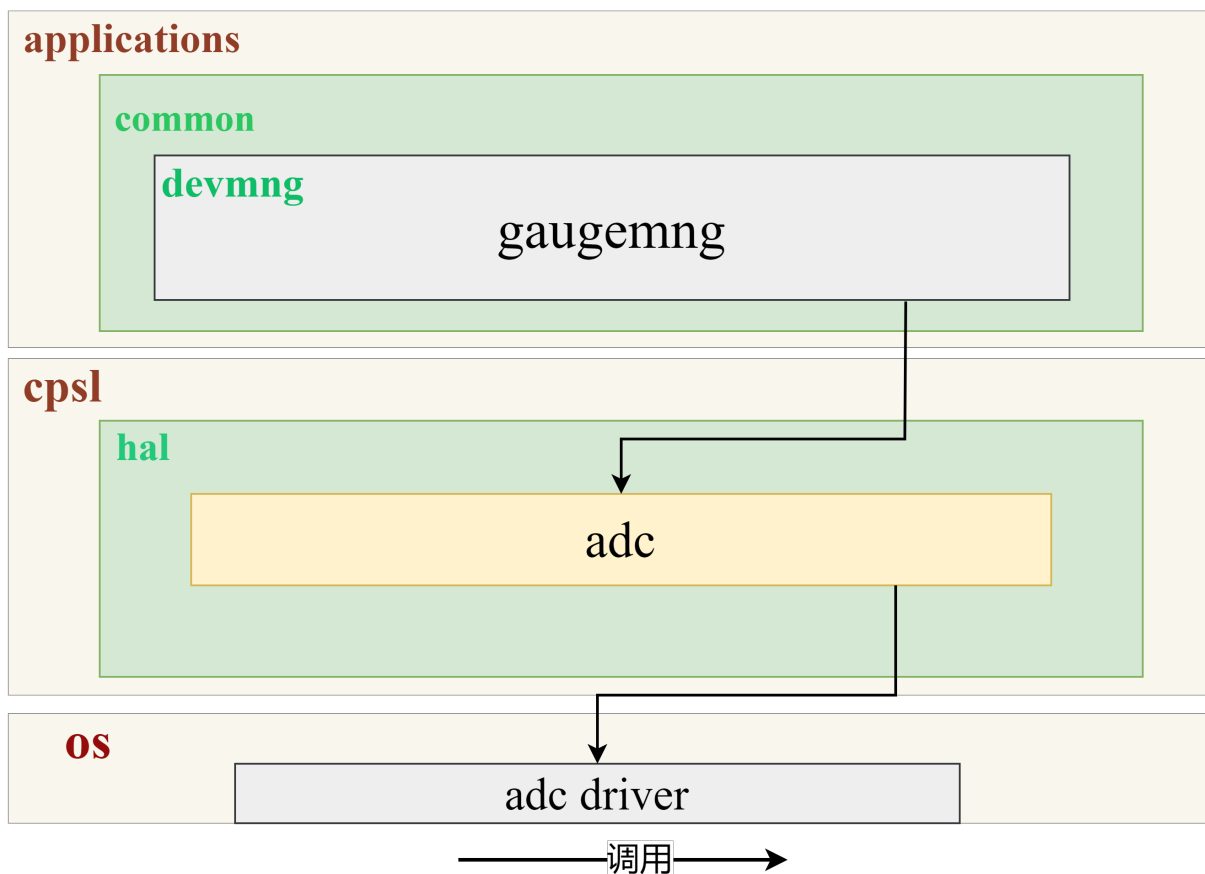


图 3.14: ADC 模块上下文

3.2.9.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 ADC 开关选项。

当 ADC 功能开启后，对外提供主动查询当前电池充电状态与电池电量，以及状态发生变化后抛事件的功能。电池充电状态和电量检测流程如图 3-15 所示。

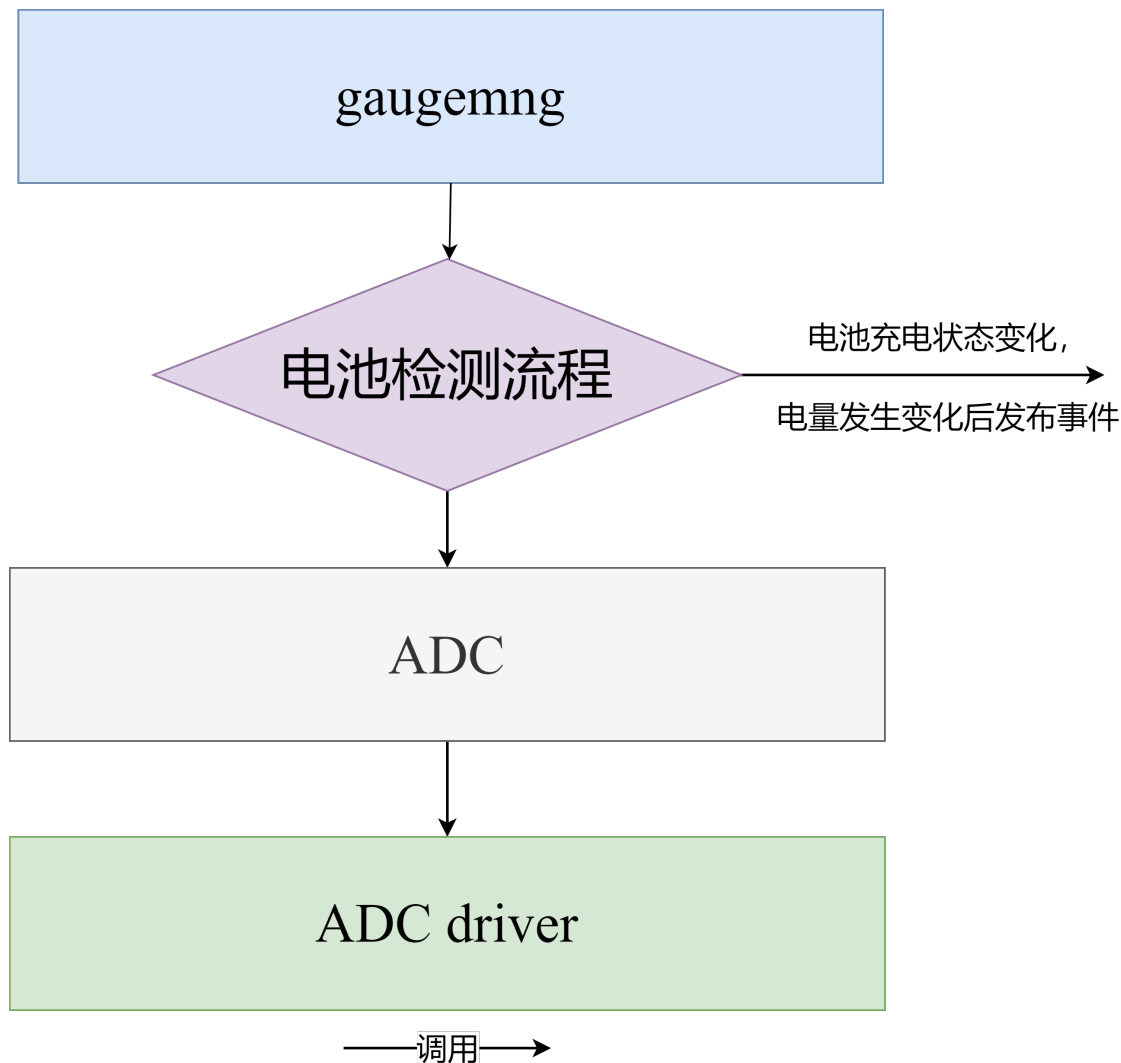


图 3.15: 电池状态检测流程图

3.2.9.3 API 参考

该功能模块为用户提供以下 API:

- `CVI_HAL_ADC_Init`: 初始化 ADC，加载 ADC 驱动。
- `CVI_HAL_ADC_Deinit`: 去初始化 ADC，卸载 ADC 驱动。
- `CVI_HAL_ADC_GetValue`: 获取电池电量。

3.2.9.3.1 CVI_HAL_ADC_Init

【描述】

初始化 ADC，加载 ADC 驱动。

【语法】

```
int32_t CVI_HAL_ADC_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_adc.h
- 库文件：libcvi_hal_adc.a / libcvi_hal_adc.so

【注意】

不支持重复初始化。

【举例】

无。

3.2.9.3.2 CVI_HAL_ADC_Deinit

【描述】

去初始化 ADC，卸载 ADC 驱动。

【语法】

```
int32_t CVI_HAL_ADC_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_adc.h
- 库文件: libcvi_hal_adc.a / libcvi_hal_adc.so

【注意】

不支持重复去初始化。

【举例】

无。

3.2.9.3.3 CVI_HAL_ADC_GetValue

【描述】

获取电池电量。

【语法】

```
int32_t CVI_HAL_ADC_GetValue(int32_t int_adc_channel);
```

【参数】

参数名称	描述	输入/输出
int_adc_channel	adc 通道	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_adc.h
- 库文件: libcvi_hal_adc.a / libcvi_hal_adc.so

【注意】

无。

【举例】

无。

3.2.9.4 数据类型

相关数据类型、数据结构无

3.2.9.5 客制化场景修改指南

menuconfig 选项选择打开即可使用。

3.2.10 GPIO && PWM

3.2.10.1 概述

GPIO、PWM 功能被封装成接口供其余外设驱动使用。

3.2.10.2 功能开关

menuconfig 中 Peripheral options 选项中提供了 GPIO 和 PWM 对应的开关选项。

3.2.10.3 API 参考

GPIO 功能模块为用户提供以下 API:

- CVI_GPIO_Export : 打开 GPIO。
- CVI_GPIO_Unexport : 释放 GPIO。
- CVI_GPIO_Direction_Input : 设置 GPIO 方向输入。
- CVI_GPIO_Direction_Output : 设置 GPIO 方向输出。
- CVI_GPIO_Set_Value : 设置 GPIO 输出为高电平或低电平
- CVI_GPIO_Get_Value : 获得 GPIO 的输入值。
- CVI_GPIO_Poll : 监听 GPIO 动作触发。

3.2.10.3.1 CVI_GPIO_Export

【描述】

获取电池电量。

【语法】

```
int32_t CVI_GPIO_Export(CVI_GPIO_NUM_E gpio);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_gpio.h`
- 库文件: `libcvi_hal_gpio.a/libcvi_hal_gpio.so`

【注意】

无。

【举例】

无。

3.2.10.3.2 CVI_GPIO_Unexport

【描述】

释放 GPIO。

【语法】

```
int32_t CVI_GPIO_Unexport(CVI_GPIO_NUM_E gpio);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_gpio.h`
- 库文件: `libcvi_hal_gpio.a/libcvi_hal_gpio.so`

【注意】

无。

【举例】

无。

3.2.10.3.3 CVI_GPIO_Direction_Input

【描述】

设置 GPIO 方向为输入。

【语法】

```
int32_t CVI_GPIO_Direction_Input(CVI_GPIO_NUM_E gpio);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gpio.h
- 库文件: libcvi_hal_gpio.a/libcvi_hal_gpio.so

【注意】

无。

【举例】

无。

3.2.10.3.4 CVI_GPIO_Direction_Output

【描述】

设置 GPIO 方向为输出。

【语法】

```
int32_t CVI_GPIO_Direction_Output(CVI_GPIO_NUM_E gpio);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gpio.h
- 库文件: libcvi_hal_gpio.a/libcvi_hal_gpio.so

【注意】

无。

【举例】

无。

3.2.10.3.5 CVI_GPIO_Set_Value

【描述】

设置 GPIO 输出为高电平或低电平

【语法】

```
int32_t CVI_GPIO_Set_Value(CVI_GPIO_NUM_E gpio, CVI_GPIO_VALUE_E value);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入
value	GPIO 值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gpio.h
- 库文件: libcvi_hal_gpio.a/libcvi_hal_gpio.so

【注意】

当 GPIO 设置为输出才可以使用

【举例】

无。

3.2.10.3.6 CVI_GPIO_Get_Value

【描述】

获得 GPIO 的输入值。

【语法】

```
int32_t CVI_GPIO_Get_Value(CVI_GPIO_NUM_E gpio, CVI_GPIO_VALUE_E *value);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入
value	GPIO 值	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_gpio.h
- 库文件: libcvi_hal_gpio.a/libcvi_hal_gpio.so

【注意】

当 GPIO 设置为输入才可以使用。

【举例】

无。

3.2.10.3.7 CVI_GPIO_Poll

【描述】

监听 GPIO 触发并执行。

【语法】

```
int32_t CVI_GPIO_Poll(CVI_GPIO_NUM_E gpio, CVI_GPIO_EDGE_E edge, FunType Fp);
```

【参数】

参数名称	描述	输入/输出
gpio	GPIO 索引	输入
edge	GPIO 动作	输入
Fp	回调函数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_gpio.h`
- 库文件: `libcvi_hal_gpio.a/libcvi_hal_gpio.so`

【注意】

无。

【举例】

无。

PWM 功能模块为用户提供以下 API:

- `CVI_PWM_Init` : PWM 初始化。
- `CVI_PWM_Deinit` : PWM 去初始化。
- `CVI_PWM_Get_Percent` : PWM 获得百分比 (占空比 / 周期 * 100)。
- `CVI_PWM_Set_Percent` : PWM 设置百分比。
- `CVI_PWM_Set_Param` : PWM 设置参数。

3.2.10.3.8 CVI_PWM_Init**【描述】**

PWM 模块初始化。

【语法】

```
int32_t CVI_PWM_Init(CVI_HAL_PWM_S pwmAttr);
```

【参数】

参数名称	描述	输入/输出
pwmAttr	PWM 结构体	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_pwm.h`
- 库文件: `libcvi_hal_pwm.a/libcvi_hal_pwm.so`

【注意】

无。

【举例】

无。

3.2.10.3.9 CVI_PWM_Deinit

【描述】

PWM 模块去初始化。

【语法】

```
int32_t CVI_PWM_Deinit(CVI_HAL_PWM_S pwmAttr);
```

【参数】

参数名称	描述	输入/输出
<code>pwmAttr</code>	PWM 结构体	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hal_pwm.h`
- 库文件: `libcvi_hal_pwm.a/libcvi_hal_pwm.so`

【注意】

无。

【举例】

无。

3.2.10.3.10 CVI_PWM_Get_Percent

【描述】

PWM 获得百分比（占空比 / 周期 * 100）。

【语法】

```
int32_t CVI_PWM_Get_Percent(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hal_pwm.h
- 库文件：libcvi_hal_pwm.a/libcvi_hal_pwm.so

【注意】

无。

【举例】

无。

3.2.10.3.11 CVI_PWM_Set_Percent

【描述】

PWM 设置百分比。

【语法】

```
int32_t CVI_PWM_Set_Percent(int32_t percentage);
```

【参数】

参数名称	描述	输入/输出
percentage	百分比	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_pwm.h
- 库文件: libcvi_hal_pwm.a/libcvi_hal_pwm.so

【注意】

无。

【举例】

无。

3.2.10.3.12 CVI_PWM_Set_Param**【描述】**

PWM 设置参数。

【语法】

```
int32_t CVI_PWM_Set_Param(CVI_HAL_PWM_S pwmAttr);
```

【参数】

参数名称	描述	输入/输出
pwmAttr	pwm 结构体	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hal_pwm.h
- 库文件: libcvi_hal_pwm.a/libcvi_hal_pwm.so

【注意】

无。

【举例】

无。

3.2.10.4 数据类型

GPIO 相关数据类型定义如下：

- CVI_GPIO_NUM_E：定义 GPIO 枚举。
- CVI_GPIO_EDGE_E：定义 GPIO 触发枚举。
- CVI_GPIO_VALUE_E：定义 GPIO 值枚举。

3.2.10.4.1 CVI_GPIO_NUM_E

【说明】

定义 GPIO 枚举。

【定义】

```
typedef enum CVI_GPIO_NUM_E {
    CVI_GPIOA_00 = 480,
    CVI_GPIOA_01, CVI_GPIOA_02, CVI_GPIOA_03, CVI_GPIOA_04, CVI_GPIOA_05,
    CVI_GPIOA_06, CVI_GPIOA_07, CVI_GPIOA_08, CVI_GPIOA_09, CVI_GPIOA_10,
    CVI_GPIOA_11, CVI_GPIOA_12, CVI_GPIOA_13, CVI_GPIOA_14, CVI_GPIOA_15,
    CVI_GPIOA_16, CVI_GPIOA_17, CVI_GPIOA_18, CVI_GPIOA_19, CVI_GPIOA_20,
    CVI_GPIOA_21, CVI_GPIOA_22, CVI_GPIOA_23, CVI_GPIOA_24, CVI_GPIOA_25,
    CVI_GPIOA_26, CVI_GPIOA_27, CVI_GPIOA_28, CVI_GPIOA_29, CVI_GPIOA_30,
    CVI_GPIOA_31,

    CVI_GPIOB_00 = 448,
    CVI_GPIOB_01, CVI_GPIOB_02, CVI_GPIOB_03, CVI_GPIOB_04, CVI_GPIOB_05,
    CVI_GPIOB_06, CVI_GPIOB_07, CVI_GPIOB_08, CVI_GPIOB_09, CVI_GPIOB_10,
    CVI_GPIOB_11, CVI_GPIOB_12, CVI_GPIOB_13, CVI_GPIOB_14, CVI_GPIOB_15,
    CVI_GPIOB_16, CVI_GPIOB_17, CVI_GPIOB_18, CVI_GPIOB_19, CVI_GPIOB_20,
    CVI_GPIOB_21, CVI_GPIOB_22, CVI_GPIOB_23, CVI_GPIOB_24, CVI_GPIOB_25,
    CVI_GPIOB_26, CVI_GPIOB_27, CVI_GPIOB_28, CVI_GPIOB_29, CVI_GPIOB_30,
    CVI_GPIOB_31,

    CVI_GPIOC_00 = 416,
    CVI_GPIOC_01, CVI_GPIOC_02, CVI_GPIOC_03, CVI_GPIOC_04, CVI_GPIOC_05,
    CVI_GPIOC_06, CVI_GPIOC_07, CVI_GPIOC_08, CVI_GPIOC_09, CVI_GPIOC_10,
    CVI_GPIOC_11, CVI_GPIOC_12, CVI_GPIOC_13, CVI_GPIOC_14, CVI_GPIOC_15,
    CVI_GPIOC_16, CVI_GPIOC_17, CVI_GPIOC_18, CVI_GPIOC_19, CVI_GPIOC_20,
    CVI_GPIOC_21, CVI_GPIOC_22, CVI_GPIOC_23, CVI_GPIOC_24, CVI_GPIOC_25,
    CVI_GPIOC_26, CVI_GPIOC_27, CVI_GPIOC_28, CVI_GPIOC_29, CVI_GPIOC_30,
    CVI_GPIOC_31,

    CVI_GPIOD_00 = 384,
    CVI_GPIOD_01, CVI_GPIOD_02, CVI_GPIOD_03, CVI_GPIOD_04, CVI_GPIOD_05,
    CVI_GPIOD_06, CVI_GPIOD_07, CVI_GPIOD_08, CVI_GPIOD_09, CVI_GPIOD_10,
    CVI_GPIOD_11, CVI_GPIOD_12, CVI_GPIOD_13, CVI_GPIOD_14, CVI_GPIOD_15,
    CVI_GPIOD_16, CVI_GPIOD_17, CVI_GPIOD_18, CVI_GPIOD_19, CVI_GPIOD_20,
    CVI_GPIOD_21, CVI_GPIOD_22, CVI_GPIOD_23, CVI_GPIOD_24, CVI_GPIOD_25,
    CVI_GPIOD_26, CVI_GPIOD_27, CVI_GPIOD_28, CVI_GPIOD_29, CVI_GPIOD_30,
    CVI_GPIOD_31,
```

(下页继续)

(续上页)

```
CVI_GPIOE_00 = 352,
CVI_GPIOE_01, CVI_GPIOE_02, CVI_GPIOE_03, CVI_GPIOE_04, CVI_GPIOE_05,
CVI_GPIOE_06, CVI_GPIOE_07, CVI_GPIOE_08, CVI_GPIOE_09, CVI_GPIOE_10,
CVI_GPIOE_11, CVI_GPIOE_12, CVI_GPIOE_13, CVI_GPIOE_14, CVI_GPIOE_15,
CVI_GPIOE_16, CVI_GPIOE_17, CVI_GPIOE_18, CVI_GPIOE_19, CVI_GPIOE_20,
CVI_GPIOE_21, CVI_GPIOE_22, CVI_GPIOE_23, CVI_GPIOE_24, CVI_GPIOE_25,
CVI_GPIOE_26, CVI_GPIOE_27, CVI_GPIOE_28, CVI_GPIOE_29, CVI_GPIOE_30,
CVI_GPIOE_31,

CVI_GPIO_MIN = CVI_GPIOE_00,
CVI_GPIO_MAX = CVI_GPIOA_31
} CVI_GPIO_NUM_E;
```

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.10.4.2 CVI_GPIO_EDGE_E

【说明】

定义 GPIO 触发枚举。

【定义】

```
typedef enum _CVI_GPIO_EDGE_E {
    CVI_GPIO_NONE ,
    CVI_GPIO_RISING,
    CVI_GPIO_FALLING,
    CVI_GPIO_BOTH
} CVI_GPIO_EDGE_E;
```

【成员】

成员名称	描述
CVI_GPIO_NONE	非中断引脚
CVI_GPIO_RISING	上升沿触发
CVI_GPIO_FALLING	下降沿触发
CVI_GPIO_BOTH	边沿触发

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.10.4.3 CVI_GPIO_VALUE_E

【说明】

定义 GPIO 值。

【定义】

```
typedef enum _CVI_GPIO_VALUE_E {  
    CVI_GPIO_VALUE_L,  
    CVI_GPIO_VALUE_H  
} CVI_GPIO_VALUE_E;
```

【成员】

成员名称	描述
CVI_GPIO_VALUE_L	低电平
CVI_GPIO_VALUE_H	高电平

【注意事项】

无。

【相关数据类型及接口】

无。

PWM 相关数据类型定义如下：

- CVI_HAL_PWM_S：定义 PWM 结构体。

3.2.10.4.4 CVI_HAL_PWM_S

【说明】

定义 PWM 结构体。

【定义】

```
typedef struct cviHAL_PWM_S {  
    uint8_t group;  
    uint8_t channel;  
    uint32_t period;  
    uint32_t duty_cycle;  
} CVI_HAL_PWM_S;
```

【成员】

成员名称	描述
group	组号
channel	通道号
period	周期
duty_cycle	占空比

【注意事项】

无。

【相关数据类型及接口】

无。

3.2.10.5 客制化场景修改指南

menuconfig 选项选择打开即可使用。

4 OSAL

4.1 概述

操作系统抽象层 (Operating System Abstraction Layer, OSAL) 是一种以实现多任务为核心的系统资源管理机制, 当前实现提供了以 mutex、task 和 time 为核心的接口。

4.2 API 参考

该功能模块为用户提供以下 API:

- `cvi_osal_get_boot_time_us`: 返回当前系统时间, 以微妙 us 为单位。
- `cvi_osal_get_boot_time_ms`: 返回当前系统时间, 以毫秒 ms 为单位。
- `cvi_osal_get_boot_time_ns`: 返回当前系统时间, 以纳秒 ns 为单位。
- `cvi_osal_mutex_create`: 创建锁。
- `cvi_osal_mutex_destroy`: 销毁锁。
- `cvi_osal_mutex_lock`: 加锁。
- `cvi_osal_mutex_unlock`: 解锁。
- `cvi_osal_cond_create`: 创建条件变量。
- `cvi_osal_cond_destroy`: 销毁条件变量。
- `cvi_osal_cond_signal`: 激发条件变量。
- `cvi_osal_cond_wait`: 无条件等待条件变量。
- `cvi_osal_cond_timedwait`: 计时等待条件变量。
- `cvi_osal_sem_create`: 创建信号量。
- `cvi_osal_sem_destroy`: 销毁信号量。
- `cvi_osal_sem_post`: 激发信号量。
- `cvi_osal_sem_wait`: 等待信号量。
- `cvi_osal_task_create`: 创建任务。
- `cvi_osal_task_destroy`: 销毁锁。

- `cvi_osal_task_join`：阻塞任务，等待新任务执行完成才能继续执行。
- `cvi_osal_task_sleep`：睡眠任务，进入阻塞状态。
- `cvi_osal_task_resched`：当前任务释放 cpu，重新进入可运行状态。

4.2.1 `cvi_osal_get_boot_time_us`

【描述】

返回当前系统时间，以微妙 us 为单位。

【语法】

```
int32_t cvi_osal_get_boot_time_us( uint64_t *time_us);
```

【参数】

参数名称	描述	输入/输出
time_us	系统时间指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.2 `cvi_osal_get_boot_time_ms`

【描述】

返回当前系统时间，以毫秒 ms 为单位。

【语法】

```
int32_t cvi_osal_get_boot_time_ms( uint64_t *time_ms);
```

【参数】

参数名称	描述	输入/输出
time_ms	系统时间指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.3 cvi_osal_get_boot_time_ns

【描述】

返回当前系统时间，以纳秒 ns 为单位。

【语法】

```
int32_t cvi_osal_get_boot_time_ns( uint64_t *time_ns);
```

【参数】

参数名称	描述	输入/输出
time_ns	系统时间指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.4 cvi_osal_mutex_create

【描述】

创建锁。

【语法】

```
int32_t cvi_osal_mutex_create(cvi_osal_mutex_attr_t *attr, cvi_osal_mutex_handle_t *mutex);
```

【参数】

参数名称	描述	输入/输出
attr	锁属性指针	输入
mutex	锁句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.5 cvi_osal_mutex_destroy

【描述】

销毁锁。

【语法】

```
int32_t cvi_osal_mutex_destroy(cvi_osal_mutex_handle_t mutex);
```

【参数】

参数名称	描述	输入/输出
mutex	锁句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.6 `cvi_osal_mutex_lock`

【描述】

加锁。

【语法】

```
int32_t cvi_osal_mutex_lock(cvi_osal_mutex_handle_t mutex, int64_t timeout_us);
```

【参数】

参数名称	描述	输入/输出
mutex	锁句柄指针	输入
timeout_us	超时等待时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.7 `cvi_osal_mutex_unlock`

【描述】

解锁。

【语法】

```
int32_t cvi_osal_mutex_unlock(cvi_osal_mutex_handle_t mutex);
```

【参数】

参数名称	描述	输入/输出
<code>mutex</code>	锁句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.8 cvi_osal_task_create

【描述】

创建任务。

【语法】

```
int32_t cvi_osal_task_create(cvi_osal_task_attr_t *attr, cvi_osal_task_handle_t *task);
```

【参数】

参数名称	描述	输入/输出
attr	任务属性指针	输入
task	任务句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.9 cvi_osal_task_destroy

【描述】

销毁任务。

【语法】

```
int32_t cvi_osal_task_destroy(cvi_osal_task_handle_t* task);
```

【参数】

参数名称	描述	输入/输出
task	任务句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.10 `cvi_osal_task_join`

【描述】

阻塞任务，等待新任务执行完成。

【语法】

```
int32_t cvi_osal_task_join(cvi_osal_task_handle_t task);
```

【参数】

参数名称	描述	输入/输出
task	任务句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.11 cvi_osal_task_sleep

【描述】

睡眠任务。

【语法】

```
void cvi_osal_task_sleep(int64_t time_us);
```

【参数】

参数名称	描述	输入/输出
time_us	时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.12 cvi_osal_task_resched

【描述】

当前任务释放 cpu 占用。

【语法】

```
void cvi_osal_task_resched(void);
```

【参数】

无。

【返回值】

无。

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.13 `cvi_osal_cond_create`

【描述】

创建条件变量。

【语法】

```
int32_t cvi_osal_cond_create(cvi_osal_cond_attr_t *attr, cvi_osal_cond_handle_t *cond);
```

【参数】

参数名称	描述	输入/输出
attr	锁属性指针	输入
cond	条件变量句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.14 cvi_osal_cond_destroy

【描述】

销毁条件变量。

【语法】

```
int32_t cvi_osal_cond_destroy(cvi_osal_cond_handle_t cond);
```

【参数】

参数名称	描述	输入/输出
cond	条件变量句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.15 cvi_osal_cond_signal

【描述】

激发条件变量。

【语法】

```
int32_t cvi_osal_cond_signal(cvi_osal_cond_handle_t cond);
```

【参数】

参数名称	描述	输入/输出
cond	条件变量句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.16 `cvi_osal_cond_wait`

【描述】

无条件等待条件变量。

【语法】

```
int32_t cvi_osal_cond_wait(cvi_osal_cond_handle_t cond, cvi_osal_mutex_handle_t mutex);
```

【参数】

参数名称	描述	输入/输出
<code>cond</code>	条件变量句柄	输入
<code>mutex</code>	锁句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.17 cvi_osal_cond_timedwait

【描述】

计时等待条件变量。

【语法】

```
int32_t cvi_osal_cond_timedwait(cvi_osal_cond_handle_t cond, cvi_osal_mutex_handle_t mutex,  
→ int64_t timeout_us);
```

【参数】

参数名称	描述	输入/输出
cond	条件变量句柄	输入
mutex	锁句柄	输入
timeout_us	超时时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_osal.h
- 库文件: libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.18 cvi_osal_sem_create

【描述】

创建信号量。

【语法】

```
int32_t cvi_osal_sem_create(cvi_osal_sem_attr_t *attr, cvi_osal_sem_handle_t *sem);
```

【参数】

参数名称	描述	输入/输出
attr	信号量属性指针	输入
sem	信号量句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.19 `cvi_osal_sem_destroy`

【描述】

销毁信号量。

【语法】

```
int32_t cvi_osal_sem_destroy(cvi_osal_sem_handle_t sem);
```

【参数】

参数名称	描述	输入/输出
<code>sem</code>	信号量句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.2.20 cvi_osal_sem_wait

【描述】

等待信号量。

【语法】

```
int32_t cvi_osal_sem_wait(cvi_osal_sem_handle_t sem, int64_t timeout_us);
```

【参数】

参数名称	描述	输入/输出
sem	信号量句柄	输入
timeout_us	超时等待时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_osal.h
- 库文件：libcvi_osal.a/libcvi_osal.so

【注意】

无。

【举例】

无。

4.2.21 cvi_osal_sem_post

【描述】

激发信号量。

【语法】

```
int32_t cvi_osal_sem_post(cvi_osal_sem_handle_t sem);
```

【参数】

参数名称	描述	输入/输出
sem	信号量句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_osal.h`
- 库文件: `libcvi_osal.a/libcvi_osal.so`

【注意】

无。

【举例】

无。

4.3 数据类型

相关数据类型定义如下：

- `cvi_osal_mutex_attr_t`：定义锁属性结构体。
- `cvi_osal_mutex_t` / `*cvi_osal_mutex_handle_t`：定义锁结构体/锁结构体指针。
- `cvi_osal_task_attr_t`：定义任务属性结构体。
- `cvi_osal_task_t` / `*cvi_osal_task_handle_t`：定义任务结构体/任务结构体指针。
- `cvi_osal_task_entry_t`：定义任务属性结构体。
- `cvi_osal_cond_attr_t`：定义条件变量属性结构体。
- `cvi_osal_cond_t` / `*cvi_osal_cond_handle_t`：定义条件变量结构体/条件变量结构体指针。
- `cvi_osal_sem_attr_t`：定义信号量属性结构体。
- `cvi_osal_sem_t` / `*cvi_osal_sem_handle_t`：定义信号量结构体/信号量结构体指针。

4.3.1 `cvi_osal_mutex_attr_t`

【说明】

定义锁属性。

【定义】

```
typedef struct {  
    const char          *name;  
    int32_t type;  
} cvi_osal_mutex_attr_t;
```


【成员】

成员名称	描述
name	名称
type	类型

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.2 `cvi_osal_mutex_t` / `*cvi_osal_mutex_handle_t`

【说明】

定义锁结构体/锁结构体指针。

【定义】

```
typedef struct {  
    cvi_osal_mutex_attr_t    attr;  
    pthread_mutex_t mutex;  
} cvi_osal_mutex_t, *cvi_osal_mutex_handle_t;
```

【成员】

成员名称	描述
attr	锁属性
mutex	锁

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.3 `cvi_osal_task_attr_t`

【说明】

定义任务属性。

【定义】

```
typedef struct {  
    const char          *name;  
    cvi_osal_task_entry_t entry;  
    void                *param;  
    int32_t             priority;  
    bool                detached;  
    int32_t stack_size;  
} cvi_osal_task_attr_t;
```

【成员】

成员名称	描述
name	名称
entry	任务回调函数
param	参数指针
priority	优先级
detached	是否分离
stack_size	栈大小

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.4 cvi_osal_task_t / *cvi_osal_task_handle_t

【说明】

定义任务结构体/任务结构体指针。

【定义】

```
typedef struct {  
    cvi_osal_task_attr_t attr;  
    pthread_t            task;  
} cvi_osal_task_t, *cvi_osal_task_handle_t;
```

【成员】

成员名称	描述
attr	任务属性
task	线程

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.5 cvi_osal_task_entry_t

【说明】

定义任务回调函数。

【定义】

```
typedef void (*cvi_osal_task_entry_t)(void *param);
```

【成员】

成员名称	描述
param	参数指针

【注意事项】

无。

【相关数据类型及接口】

无

4.3.6 cvi_osal_cond_attr_t

【说明】

定义条件变量属性结构体。

【定义】

```
typedef struct
{
    const char *name;
    int32_t clock;
} cvi_osal_cond_attr_t;
```

【成员】

成员名称	描述
name	名称
clock	时钟 id

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.7 cvi_osal_cond_t / * cvi_osal_cond_handle_t

【说明】

定义条件变量结构体/条件变量结构体指针。

【定义】

```
typedef struct
{
    cvi_osal_cond_attr_t attr;
    pthread_cond_t cond;
} cvi_osal_cond_t, *cvi_osal_cond_handle_t;
```

【成员】

成员名称	描述
attr	条件变量属性
cond	条件变量

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.8 cvi_osal_sem_attr_t

【说明】

定义信号量属性结构体

【定义】

```
typedef struct
{
    const char *name;
} cvi_osal_sem_attr_t;
```

【成员】

成员名称	描述
name	名称

【注意事项】

无。

【相关数据类型及接口】

无。

4.3.9 `cvi_osal_sem_t` / `*cvi_osal_sem_handle_t`

【说明】

定义信号量结构体/信号量结构体指针。

【定义】

```
typedef struct
{
    cvi_osal_sem_attr_t attr;
    sem_t sem;
} cvi_osal_sem_t, *cvi_osal_sem_handle_t;
```

【成员】

成员名称	描述
attr	信号量属性
sem	信号量

【注意事项】

无。

【相关数据类型及接口】

无。

5 COMMON

5.1 HFSM

5.1.1 概述

层次状态机 (hierarchical finite state machine, HFSM) 是 MediaSDK 的基础组件, 可以帮助我们更好梳理业务, 降低业务的复杂度。在车载 Turnkey 场景下有众多的复杂业务, 仅实现一种状态机是不够的, 需要层次状态机实现可扩展的, 易用的状态机框架。

本文主要介绍了层次状态机框架模块以及使用方法, 用于指导用户开发。

5.1.2 架构原理

状态机模块提供了一种层次状态机框架, 用户可以自定义状态以及状态的转移。层次状态机是基于消息驱动的, 具体系统框图组成及对外接口关系如图 5-1 所示。

状态机模块主要由以下部分构成:

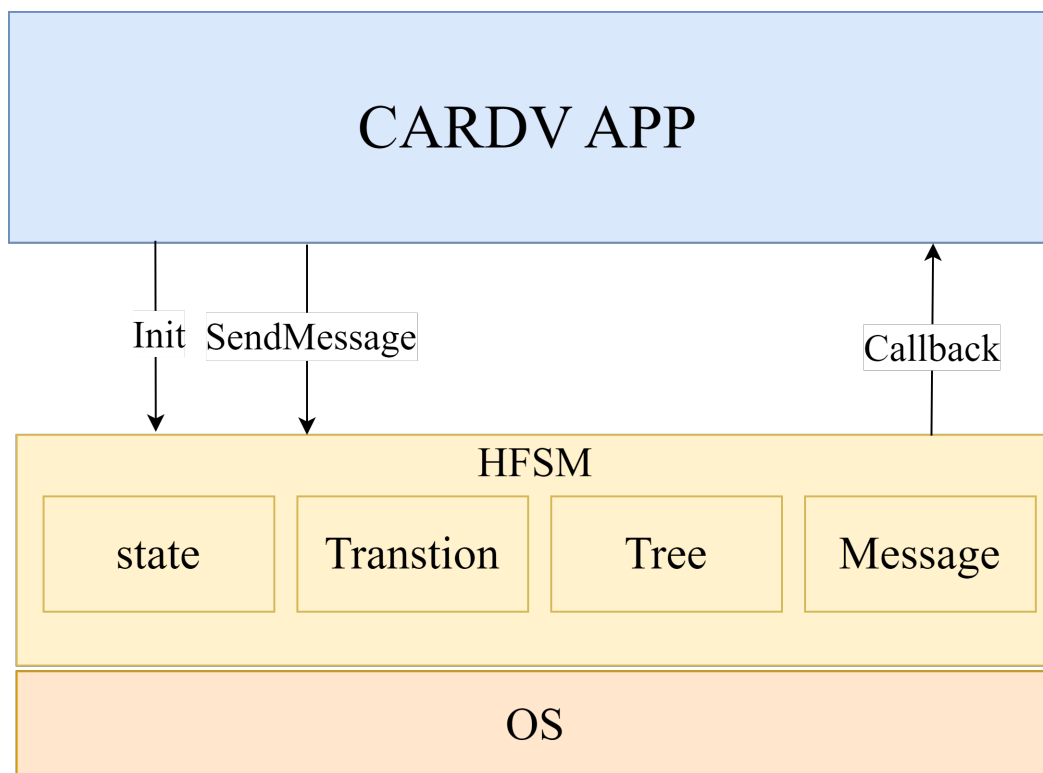


图 5.1: 状态机模块系统框图

- 状态定义

用户自定义状态：状态名称，状态进入回调，状态退出回调，消息处理回调。

- 消息

消息自定义：消息，参数 1，参数 2，处理结果，创建时间，负载。

- 状态转移

支持状态从某一状态转移到任一其他状态，具体转移策略由用户自定义。

- 层次树。

状态机内部维护一个状态树，只有一个父节点，由用户通过添加接口构建该树。

5.1.2.1 状态机模型

如图 5-2 所示，状态机的状态是以树形状态维护的，只有一个根节点，且每个节点有一个或者多个叶子节点。

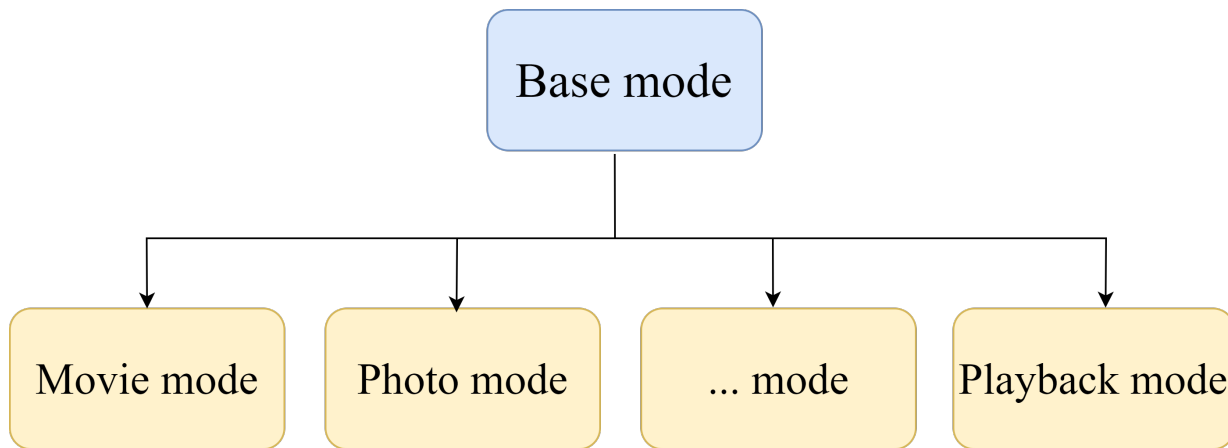


图 5.2: 状态机内部状态组织形式

5.1.2.2 状态机切换规则

状态的切换主要有以下规则：

由某一状态进入其子状态时，会调用子状态的进入函数。

由某一状态回退到其父状态时，会调用子状态的退出函数。

举例：如图 5-2 中从 Movie 状态切换到 Photo 状态，会依次调用下列回调：

Movie state Close -> Photo state start

5.1.3 API 参考

该功能模块为用户提供以下 API：

- CVI_HFSM_Create：创建 HFSM 实例。
- CVI_HFSM_Destroy：销毁 HFSM 实例。
- CVI_HFSM_AddState：添加状态。
- CVI_HFSM_SetInitialState：设置初始化状态。
- CVI_HFSM_GetCurrentState：获取当前状态。
- CVI_HFSM_Start：启动状态机。
- CVI_HFSM_Stop：停止状态机。
- CVI_HFSM_SendAsyncMessage：发送消息。

5.1.3.1 CVI_HFSM_Create

【描述】

创建 HFSM 实例。

【语法】

```
int32_t CVI_HFSM_Create(CVI_HFSM_ATTR_S *fsmAttr, CVI_HFSM_HANDLE *hfsm);
```

【参数】

参数名称	描述	输入/输出
fsmAttr	层次状态参数指针	输入
hfsm	状态机句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hfsm.h
- 库文件: libcvi_hfsm.a/libcvi_hfsm.so

【注意】

无。

【举例】

无。

5.1.3.2 CVI_HFSM_Destroy

【描述】

销毁 HFSM 实例。

【语法】

```
int32_t CVI_HFSM_Destroy(CVI_HFSM_HANDLE hfsm);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hfsm.h`
- 库文件: `libcvi_hfsm.a/libcvi_hfsm.so`

【注意】

HFSM 创建后才能调用该接口。

【举例】

无。

5.1.3.3 CVI_HFSM_AddState**【描述】**

添加状态。

【语法】

```
int32_t CVI_HFSM_AddState(CVI_HFSM_HANDLE hfsm, CVI_STATE_S *state, CVI_STATE_S *parent);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入
state	状态指针	输入
parent	父状态指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hfsm.h`
- 库文件: `libcvi_hfsm.a/libcvi_hfsm.so`

【注意】

根状态的父状态指针需要为空。

【举例】

无。

5.1.3.4 CVI_HFSM_SetInitialState

【描述】

设置初始化状态。

【语法】

```
int32_t CVI_HFSM_SetInitialState(CVI_HFSM_HANDLE hfsm, uint32_t stateID);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入
stateID	状态 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hfsm.h
- 库文件：libcvi_hfsm.a/libcvi_hfsm.so

【注意】

当前默认的初始化状态是 movie。

【举例】

无。

5.1.3.5 CVI_HFSM_GetCurrentState

【描述】

获得当前状态。

【语法】

```
int32_t CVI_HFSM_GetCurrentState(CVI_HFSM_HANDLE hfsm, CVI_STATE_S *state);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入
state	当前状态指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hfsm.h`
- 库文件: `libcvi_hfsm.a/libcvi_hfsm.so`

【注意】

无。

【举例】

无。

5.1.3.6 CVI_HFSM_Start

【描述】

启动状态机。

【语法】

```
int32_t CVI_HFSM_Start(CVI_HFSM_HANDLE hfsm);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_hfsm.h`
- 库文件: `libcvi_hfsm.a/libcvi_hfsm.so`

【注意】

无。

【举例】

无。

5.1.3.7 CVI_HFSM_Stop

【描述】

停止状态机。

【语法】

```
int32_t CVI_HFSM_Stop(CVI_HFSM_HANDLE hfsm);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_hfsm.h
- 库文件：libcvi_hfsm.a/libcvi_hfsm.so

【注意】

无。

【举例】

无。

5.1.3.8 CVI_HFSM_SendAsyncMessage

【描述】

发送异步消息。

【语法】

```
int32_t CVI_HFSM_SendAsyncMessage(CVI_HFSM_HANDLE hfsm, CVI_MESSAGE_S *msg);
```

【参数】

参数名称	描述	输入/输出
hfsm	状态机实例句柄	输入
msg	信息结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_hfsm.h
- 库文件: libcvi_hfsm.a/libcvi_hfsm.so

【注意】

无。

【举例】

无。

5.1.4 数据类型

相关数据类型定义如下：

- `CVI_STATE_NAME_LEN`：状态机名称最大长度。
- `CVI_STATE_MAX_AMOUNT`：定义支持状态的最大个数。
- `CVI_PROCESS_MSG_RESULTE_OK`：定义消息处理 OK 返回值。
- `CVI_PROCESS_MSG_UNHANDLER`：消息未处理返回值。
- `CVI_STATE_S`：定义状态类型结构体。
- `CVI_HFSM_EVENT_E`：定义状态机事件回调枚举。
- `CVI_HFSM_EVENT_INFO_S`：定义事件信息结构体。
- `CVI_HFSM_EVENT_CALLBACK`：定义事件回调函数。
- `CVI_HFSM_ATTR_S`：定义状态机参数结构体。
- `CVI_MESSAGE_S`：定义消息结构体

5.1.4.1 CVI_STATE_NAME_LEN

【说明】

状态机名称最大长度。

【定义】

```
#define CVI_STATE_NAME_LEN 64
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.2 CVI_STATE_MAX_AMOUNT**【说明】**

状态的最大数量。

【定义】

```
#define CVI_STATE_MAX_AMOUNT 32
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.3 CVI_PROCESS_MSG_RESULTE_OK**【说明】**

消息处理结果 OK。

【定义】

```
#define CVI_PROCESS_MSG_RESULTE_OK (0)
```

【注意事项】

状态的消息处理函数中，对某个消息处理完成后返回 CVI_PROCESS_MSG_RESULTE_OK。

【相关数据类型及接口】

无。

5.1.4.4 CVI_PROCESS_MSG_UNHANDLER**【说明】**

未处理消息返回值。

【定义】

```
#define CVI_PROCESS_MSG_UNHANDLER (-1)
```

【注意事项】

状态的消息处理函数中，没有对某个消息进行处理时返回 CVI_PROCESS_MSG_UNHANDLER。

【相关数据类型及接口】

无。

5.1.4.5 CVI_STATE_S**【说明】**

定义状态类型结构体。

【定义】

```
typedef struct CviStateInfo {  
    uint32_t stateID;           /* state id */  
    char name[CVI_STATE_NAME_LEN]; /* state name */  
    int32_t (*open)(void);      /* call when state open */  
    int32_t (*close)(void);     /* call when state close */  
    /* call when process a message */  
    int32_t (*processMessage)(CVI_MESSAGE_S *msg, void* argv, uint32_t *stateID);  
    void* argv;                 /* User private, used by processMessage */  
}CVI_STATE_S;
```

【成员】

成员名称	描述
stateID	状态标识
name	状态名称
open	状态进入回调函数
close	状态退出回调函数
processMessage	消息处理回调函数
argv	用户自定义指针函数

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.6 CVI_HFSM_EVENT_E

【说明】

定义状态机事件类型枚举。

【定义】

```
typedef enum CviHfsmEventE {  
    CVI_HFSM_EVENT_HANDLE_MSG = 0,      /* handler message */  
    CVI_HFSM_EVENT_UNHANDLE_MSG,        /* unhandler message */  
    CVI_HFSM_EVENT_TRANSTION_ERROR,      /* transtion error */  
    CVI_HFSM_EVENT_BUTT  
} CVI_HFSM_EVENT_E;
```

【成员】

成员名称	描述
CVI_HFSM_EVENT_HANDLE_MSG	处理的消息事件类型
CVI_HFSM_EVENT_UNHANDLE_MSG	未处理的消息事件类型
CVI_HFSM_EVENT_TRANSTION_ERROR	状态转移错误事件类型
CVI_HFSM_EVENT_BUTT	无效事件类型

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.7 CVI_HFSM_EVENT_INFO_S

【说明】

定义状态机事件消息结构体。

【定义】

```
typedef struct HiHfsmEventInfo {  
    int32_t s32ErrorNo;  
    CVI_HFSM_EVENT_E enEventCode;  
    CVI_MESSAGE_S *pstunHandlerMsg;  
} CVI_HFSM_EVENT_INFO_S;
```

【成员】

成员名称	描述
s32ErrorNo	错误码
enEventCode	事件类型
pstunHandlerMsg	消息结构体指针

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.8 CVI_HFSM_EVENT_CALLBACK

【说明】

定义状态机事件回调函数类型。

【定义】

```
typedef int32_t (*CVI_HFSM_EVENT_CALLBACK)(CVI_HFSM_HANDLE hfsmHandle, const_
↪CVI_HFSM_EVENT_INFO_S *eventInfo);
```

【成员】

成员名称	描述
hfsmHandle	状态机句柄
eventInfo	事件消息结构体指针

【注意事项】

无。

【相关数据类型及接口】

无。

5.1.4.9 CVI_HFSM_ATTR_S

【说明】

定义状态机创建参数结构体。

【定义】

```
typedef struct CviHfsmAttr {
    CVI_HFSM_EVENT_CALLBACK fnHfsmEventCallback;
    uint32_t u32StateMaxAmount;
    uint32_t u32MessageQueueSize;
} CVI_HFSM_ATTR_S;
```

【成员】

成员名称	描述
fnHfsmEventCallback	事件回调函数指针
u32StateMaxAmount	支持的状态数量最大数量
u32MessageQueueSize	消息队列长度

【注意事项】

u32MessageQueueSize 约束了状态机内部一个时刻存在的未处理消息的最大数量，如果当前未处理消息数量达到该值，状态机将无法接收新的消息。用户根据实际业务场景配置该值。

【相关数据类型及接口】

无。

5.1.4.10 CVI_MESSAGE_S

【说明】

定义消息结构体。

【定义】

```
typedef CVI_EVENT_S    CVI_MESSAGE_S;

typedef struct ps_msg_s {
    uint32_t topic; // Message topic
    int32_t arg1;
    int32_t arg2;
    int32_t s32Result;
    uint64_t u64CreateTime;
    uint8_t aszPayload[MSG_PAYLOAD_LEN];
} CVI_EVENT_S;
```

【成员】

成员名称	描述
topic	消息 ID
arg1	参数 1
arg2	参数 2
s32Result	执行结果
u64CreateTime	创建时间
aszPayload	负载内容

【注意事项】

无。

【相关数据类型及接口】

无。

5.2 EVENTHUB

5.2.1 概述

事件路由（eventhub）是 MediaSDK 的基础组件。本模块根据业务场景定制了一个基本的事件路由模块。下文主要介绍事件路由模块以及使用方法，用于指导用户开发。

5.2.2 架构原理

事件路由模块提供了一种事件分发的框架，用户可以自定义事件以及事件的处理策略。具体系统框图组成及对外接口关系如图 5-3 所示。

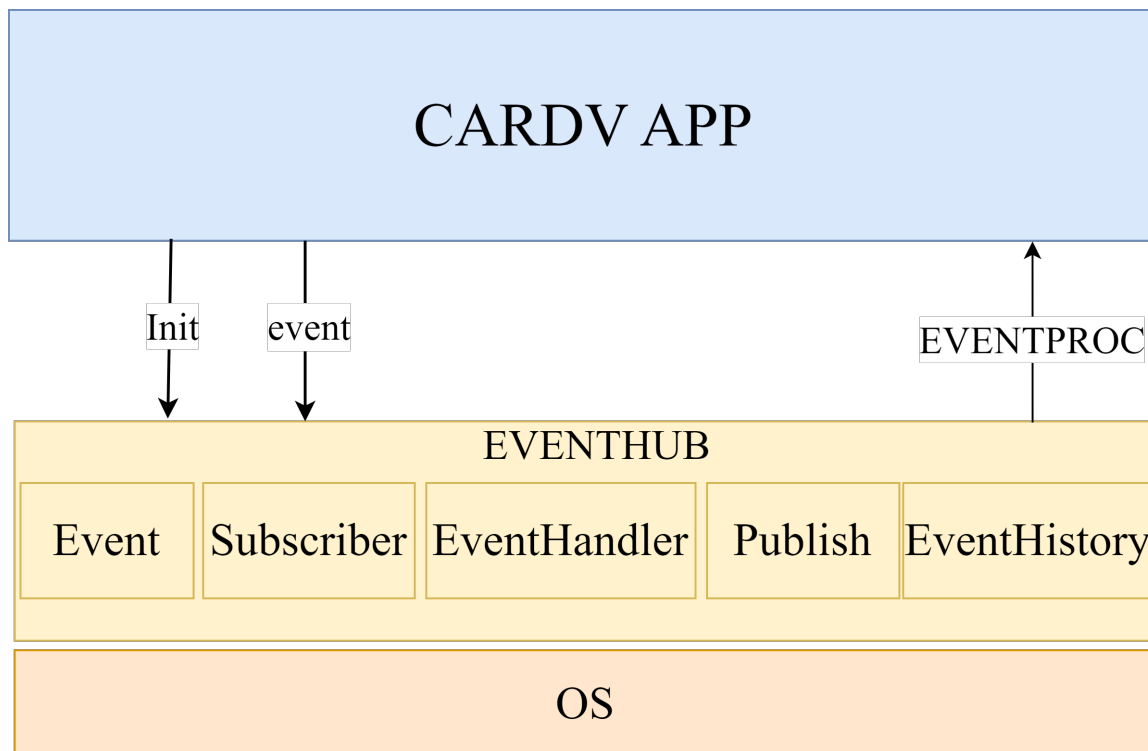


图 5.3: 事件路由模块系统框图

概括来说，事件路由模块主要由以下部分构成：

- 订阅者

用户自定义订阅者：订阅者名称，事件处理函数，用户扩展参数，事件是否同步处理。

- 事件定义

用户自定义事件：事件 ID，参数 1，参数 2，处理结果，创建时间，负载。

- 事件发布

支持事件统一发布。

- 事件处理者。

订阅者支持异步处理事件，内部启动线程独立处理。

- 历史事件

支持上一次事件的查询。

5.2.3 API 参考

该功能模块为用户提供以下 API:

- `CVI_EVTHUB_Init` : 初始化事件路由模块。
- `CVI_EVTHUB_Deinit` : 去初始化事件路由模块。
- `CVI_EVTHUB_Register` : 注册事件。
- `CVI_EVTHUB_UnRegister` : 解注册事件。
- `CVI_EVTHUB_Publish` : 发布事件。
- `CVI_EVTHUB_CreateSubscriber` : 创建订阅者。
- `CVI_EVTHUB_DestroySubscriber` : 销毁订阅者。
- `CVI_EVTHUB_Subscribe` : 订阅事件。
- `CVI_EVTHUB_UnSubscribe` : 解订阅事件。
- `CVI_EVTHUB_GetEventHistory` : 获取历史事件。

5.2.3.1 `CVI_EVTHUB_Init`

【描述】

初始化事件路由模块。

【语法】

```
int32_t CVI_EVENTHUB_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_eventhub.h`
- 库文件: `libcvi_eventhub.a/libcvi_eventhub.so`

【注意】

该模块是单实例，仅需一次初始化。

【举例】

无。

5.2.3.2 CVI_EVTHUB_Deinit

【描述】

去初始化事件路由模块。

【语法】

```
int32_t CVI_EVENTHUB_DeInit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.3 CVI_EVTHUB_Register

【描述】

注册事件。

【语法】

```
int32_t CVI_EVENTHUB_RegisterTopic(CVI_TOPIC_ID topic);
```

【参数】

参数名称	描述	输入/输出
topic	事件 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_eventhub.h`
- 库文件: `libcvi_eventhub.a/libcvi_eventhub.so`

【注意】

无。

【举例】

无。

5.2.3.4 CVI_EVTHUB_UnRegister

【描述】

解注册事件。

【语法】

```
int32_t CVI_EVENTHUB_UnRegisterTopic(CVI_TOPIC_ID topic);
```

【参数】

参数名称	描述	输入/输出
topic	事件 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_eventhub.h`
- 库文件: `libcvi_eventhub.a/libcvi_eventhub.so`

【注意】

无。

【举例】

无。

5.2.3.5 CVI_EVTHUB_Publish

【描述】

发布事件。

【语法】

```
int32_t CVI_EVENTHUB_Publish(CVI_EVENT_S *pEvent);
```

【参数】

参数名称	描述	输入/输出
pEvent	事件指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.6 CVI_EVTHUB_CreateSubscriber

【描述】

创建订阅者。

【语法】

```
int32_t CVI_EVENTHUB_CreateSubscriber(CVI_EVENTHUB_SUBSCRIBER_S *pstSubscriber,  
→ CVI_MW_PTR *ppSubscriber);
```

【参数】

参数名称	描述	输入/输出
pstSubscriber	订阅者参数	输入
ppSubscriber	订阅者句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.7 CVI_EVTHUB_DestroySubscriber

【描述】

销毁订阅者。

【语法】

```
int32_t CVI_EVTHUB_DestroySubscriber(CVI_MW_PTR ppSubscriber);
```

【参数】

参数名称	描述	输入/输出
ppSubscriber	订阅者句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.8 CVI_EVTHUB_Subscribe

【描述】

订阅事件。

【语法】

```
int32_t CVI_EVENTHUB_Subscribe(CVI_MW_PTR pSubscriber, CVI_TOPIC_ID topic);
```

【参数】

参数名称	描述	输入/输出
pSubscriber	订阅者句柄	输入
topic	消息 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.9 CVI_EVTHUB_UnSubscribe

【描述】

取消订阅事件。

【语法】

```
int32_t CVI_EVENTHUB_UnSubscribe(CVI_MW_PTR pSubscriber, CVI_TOPIC_ID topic);
```

【参数】

参数名称	描述	输入/输出
pSubscriber	订阅者句柄	输入
topic	消息 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.3.10 CVI_EVTHUB_GetEventHistory

【描述】

获取历史事件。

【语法】

```
int32_t CVI_EVENTHUB_GetEventHistory(CVI_TOPIC_ID topic, CVI_EVENT_S *msg);
```

【参数】

参数名称	描述	输入/输出
topic	消息 ID	输入
msg	事件指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_eventhub.h
- 库文件: libcvi_eventhub.a/libcvi_eventhub.so

【注意】

无。

【举例】

无。

5.2.4 数据类型

相关数据类型定义如下：

- `MSG_PAYLOAD_LEN`：定义事件负载数据最大长度。
 - `CVI_EVENTHUB_SUBSCRIBE_NAME_LEN`：定义订阅者名称最大长度。
 - `CVI_TOPIC_ID`：定义事件 ID。
 - `CVI_EVENT_S`：定义事件结构体。
 - `CVI_EVENTHUB_SUBSCRIBER_S`：定义订阅者结构体。
-

5.2.4.1 MSG_PAYLOAD_LEN

【说明】

定义事件负载数据最大长度。

【定义】

```
#define MSG_PAYLOAD_LEN 128
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.2.4.2 CVI_EVENTHUB_SUBSCRIBE_NAME_LEN

【说明】

定义订阅者名称最大长度。

【定义】

```
#define CVI_EVENTHUB_SUBSCRIBE_NAME_LEN 16
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.2.4.3 CVI_TOPIC_ID

【说明】

定义事件 ID。

【定义】

```
typedef uint32_t CVI_TOPIC_ID;
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.2.4.4 CVI_EVENT_S

【说明】

定义事件结构体。

【定义】

```
typedef struct ps_msg_s {  
    uint32_t topic; // Message topic  
    int32_t arg1;  
    int32_t arg2;  
    int32_t s32Result;  
    uint64_t u64CreateTime;  
    uint8_t aszPayload[MSG_PAYLOAD_LEN];  
} CVI_EVENT_S;
```

【成员】

成员名称	描述
topic	消息 ID
arg1	参数 1
arg2	参数 2
s32Result	执行结果
u64CreateTime	创建时间
aszPayload	负载内容

【注意事项】

无。

【相关数据类型及接口】

无。

5.2.4.5 CVI_EVENTHUB_SUBSCRIBER_S

【说明】

定义订阅者结构体。

【定义】

```
typedef struct {  
    char asName[CVI_EVENTHUB_SUBSCRIBE_NAME_LEN];  
    void *argv;  
    int32_t (*new_msg_cb)(void *argv, CVI_EVENT_S *msg);  
    bool sync;  
} CVI_EVENTHUB_SUBSCRIBER_S;
```

【成员】

成员名称	描述
asName	订阅者名称
argv	用户定义私有参数指针
new_msg_cb	订阅者事件处理函数
sync	事件处理是否同步

【注意事项】

创建时间在事件发布时，由模块内部获取系统 clock 时间填充，单位秒/s。

【相关数据类型及接口】

无。

5.3 MQ

5.3.1 概述

消息队列（MQ）主要用于进程间通信，如利用 cc_tools 工具发消息给应用层执行对应操作，底层是基于 socket 实现。

5.3.2 API 参考

该功能模块为用户提供以下 API：

- CVI_MQ_CreateEndpoint：消息队列创建端点。
- CVI_MQ_DestroyEndpoint：消息队列销毁端点。
- CVI_MQ_SendRAW：消息队列发送消息。
- CVI_MQ_Send：消息队列发送消息，内部调用 CVI_MQ_SendRAW。
- CVI_MQ_SendAck：消息队列发送确认消息。

- CVI_MQ_SendRAW_ACK：给 sample 用的。
- CVI_MQ_Send_NeedAck：给 sample 用的。

5.3.2.1 CVI_MQ_CreateEndpoint

【描述】

消息队列创建端点。

【语法】

```
int32_t CVI_MQ_CreateEndpoint(CVI_MQ_ENDPOINT_CONFIG_S *config, CVI_MQ_
↪ENDPOINT_HANDLE_t *ep);
```

【参数】

参数名称	描述	输入/输出
config	端点配置结构体指针	输入
ep	端点结构体句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_mq.h
- 库文件：libcvi_mq.a/libcvi_mq.so

【注意】

无。

【举例】

无。

5.3.2.2 CVI_MQ_DestroyEndpoint

【描述】

消息队列销毁端点。

【语法】

```
int32_t CVI_MQ_DestroyEndpoint(CVI_MQ_ENDPOINT_HANDLE_t ep);
```

【参数】

参数名称	描述	输入/输出
ep	端点结构体句柄指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_mq.h
- 库文件: libcvi_mq.a/libcvi_mq.so

【注意】

无。

【举例】

无。

5.3.2.3 CVI_MQ_SendRAW

【描述】

消息队列发送消息。

【语法】

```
int32_t CVI_MQ_SendRAW(CVI_MQ_MSG_S *msg);
```

【参数】

参数名称	描述	输入/输出
msg	消息结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_mq.h
- 库文件: libcvi_mq.a/libcvi_mq.so

【注意】

无。

【举例】

无。

5.3.2.4 CVI_MQ_Send

【描述】

消息队列发送消息，内部调用 CVI_MQ_SendRAW。

【语法】

```
int32_t CVI_MQ_Send(
    CVI_MQ_ID_t          target_id,
    int32_t               arg1,
    int32_t               arg2,
    int16_t               seq_no,
    char                  *payload,
    int16_t               payload_len);
```

【参数】

参数名称	描述	输入/输出
target_id	消息 ID	输入
arg1	参数 1	输入
arg2	参数 2	输入
seq_no	消息序列	输入
payload	负载消息	输入
payload_len	负载长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_mq.h
- 库文件：libcvi_mq.a/libcvi_mq.so

【注意】

无。

【举例】

无。

5.3.2.5 CVI_MQ_SendAck

【描述】

消息队列发送确认消息。

【语法】

```
int32_t CVI_MQ_SendAck(CVI_MQ_ENDPOINT_HANDLE_t ep, MSG_ACK_t *ack_msg,  
↪int16_t client_id);
```

【参数】

参数名称	描述	输入/输出
ep	消息队列端点句柄	输入
ack_msg	确认消息指针	输入
client_id	客户端 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_mq.h
- 库文件：libcvi_mq.a/libcvi_mq.so

【注意】

无。

【举例】

无。

5.3.3 数据类型

相关数据类型定义如下：

- CVI_MQ_MSG_S：定义消息队列消息结构体。
- CVI_MQ_ENDPOINT_S：定义消息队列端点结构体。
- CVI_MQ_ENDPOINT_CONFIG_S：定义消息队列端点配置结构体。

5.3.3.1 CVI_MQ_MSG_S

【说明】

定义消息队列消息结构体。

【定义】

```
typedef struct CVI_MQ_MSG_s {  
    CVI_MQ_ID_t target_id;  
    int32_t arg1;  
    int32_t arg2;  
    int16_t seq_no;  
    int16_t len;  
    uint64_t create_time;  
    uint32_t needack;  
    int16_t client_id;  
    char payload[CVI_MQ_MSG_PAYLOAD_LEN];  
} __attribute__((__packed__)) CVI_MQ_MSG_S;
```

【成员】

成员名称	描述
target_id	消息 ID
arg1	参数 1
arg2	参数 2
seq_no	消息序列
len	消息长度
seq_no	消息序列
create_time	消息创建时间
needack	0 或者 1，是否确认
client_id	客户端 ID
payload	负载

【注意事项】

无。

【相关数据类型及接口】

无。

5.3.3.2 CVI_MQ_ENDPOINT_S

【说明】

定义消息队列端点结构体。

【定义】

```
struct CVI_MQ_ENDPOINT_S {  
    CVI_MQ_ENDPOINT_CONFIG_S config;  
    cvi_osal_task_handle_t recv_task;  
    volatile bool exit;
```

(下页继续)

(续上页)

```
int32_t socket_fd;  
};
```

【成员】

成员名称	描述
config	端点配置结构体
recv_task	任务句柄
exit	是否退出
socket_fd	socket 文件描述符

【注意事项】

无。

【相关数据类型及接口】

无。

5.3.3.3 CVI_MQ_ENDPOINT_CONFIG_S

【说明】

定义消息队列端点配置结构体。

【定义】

```
typedef struct {  
    const char *name;  
    CVI_MQ_ID_t id;  
    CVI_MQ_RECV_CB_t recv_cb;  
    void *recv_cb_arg;  
} CVI_MQ_ENDPOINT_CONFIG_S;
```

【成员】

成员名称	描述
name	名称
id	索引
recv_cb	接受回调函数
recv_cb_arg	接受回调参数

【注意事项】

无。

【相关数据类型及接口】

无。

5.4 DTCF

5.4.1 概述

日期时间文件系统模块（DTCF）是 MediaSDK 的基础组件，为录像、拍照、预览和回放等业务层应用提供媒体文件的命名添加、检索、删除和关联功能。

文件的命名及目录组织方式，是决定文件管理方式的核心要素之一；不同的产品形态根据业务需求可以选择不同的文件管理方式；目前本模块主要提供了按日期时间的文件管理方式。其在整个系统中的位置如图 5-4 所示。

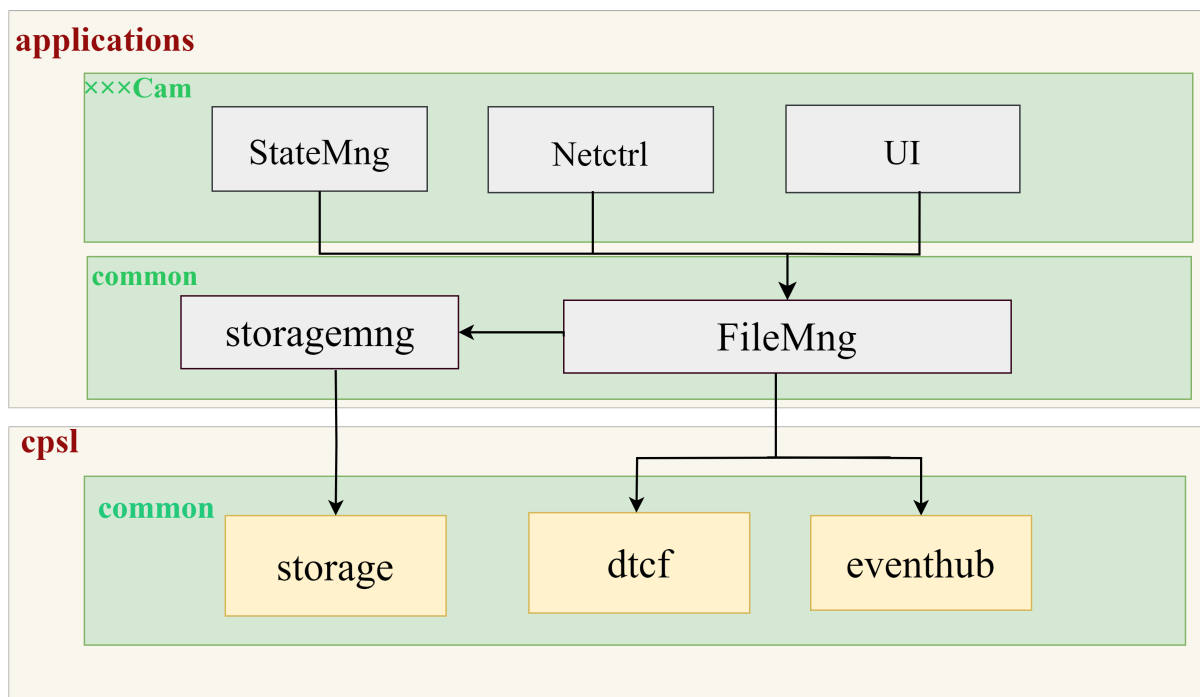


图 5.4: 文件管理模块系统框图

5.4.2 架构原理

日期时间文件管理模块主要用来提供以下功能：

- 支持扫描指定目录的文件，并按时间排序。
- 支持获取扫描的文件数量。
- 支持按索引获取扫描文件的信息。
- 支持按系统时间日期生成新的文件名。
- 支持添加和删除文件。
- 支持文件修复功能。

DTCF 提供的文件目录规则如下：

DTCF 文件管理是按照日期时间命名文件的一种规则，如图 5-5。DTCF 文件的全路径是由根路径、二级目录名、文件名三部分组成。路径中黑色部分由本模块内产生，红色和绿色部分在初始化时由外部配置决定。

规则：

ROOT/DIR/YYYY_MM_DD_HHMMSS_XX_SUFFIX.EXT

示例：

/mnt/sd/CARDV/MOVIE_b/2022_01_02_073245_00_b.MOV

图 5.5: DTCF 文件命名规则

目录名由外部参数决定，每种目录类型中的文件名都有默认的后缀名称，如下表 5-1 所示：

表 5.1: 目录类型说明

目录类型	文件名后缀	说明
MOVIE	无	Sensor0 普通录像目录
MOVIE_b	_b	Sensor1 普通录像目录
PHOTO	无	Sensor0 抓拍目录
PHOTO_b	_b	Sensor1 抓拍目录
EMR	无	Sensor0 紧急录像目录
EMR_b	_b	Sensor1 紧急录像目录

目前支持的文件后缀类型如下表 5-2 所示：

表 5.2: 当前支持的文件后缀类型

功能	文件扩展名
录像	MOV、MP4、TS、ES
抓拍	JPG
回放	MOV、MP4、TS

5.4.3 API 参考

该功能模块为用户提供以下 API:

- `CVI_DTCF_Init` : 初始化文件管理模块。
- `CVI_DTCF_GetDirNames` : 获得所有二级目录名称。
- `CVI_DTCF_DeInit` : 去初始化文件管理模块。
- `CVI_DTCF_Scan` : 文件目录扫描。
- `CVI_DTCF_GetFileByIndex` : 通过索引获取文件。
- `CVI_DTCF_DelFileByIndex` : 通过索引删除文件。
- `CVI_DTCF_AddFile` : 添加文件。
- `CVI_DTCF_GetOldestFileIndex` : 获取最老文件索引。
- `CVI_DTCF_GetFileAmount` : 获取文件数量。
- `CVI_DTCF_GetOldestFilePath` : 获取最老文件路径。
- `CVI_DTCF_CreateFilePath` : 创建文件路径。
- `CVI_DTCF_GetRelatedFilePath` : 获取关联文件路径。
- `CVI_DTCF_GetEmrFilePath` : 获取紧急录像文件路径。
- `CVI_DTCF_GetFileDirType` : 获取文件所在目录类型。

5.4.3.1 `CVI_DTCF_Init`

【描述】

初始化文件管理模块。

【语法】

```
int32_t CVI_DTCF_Init(const char *pazRootDir, const char azDirNames[DTCF_DIR_BUTT][CVI_DIR_LEN_MAX]);
```

【参数】

参数名称	描述	输入/输出
pazRootDir	文件系统根目录指针	输入
azDirNames	二级目录名称	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

无。

【举例】

无。

5.4.3.2 CVI_DTCF_GetDirNames**【描述】**

获得所有二级目录名称。

【语法】

```
int32_t CVI_DTCF_GetDirNames(char (*pazDirNames)[CVI_DIR_LEN_MAX], uint32_t  
→u32DirAmount);
```

【参数】

参数名称	描述	输入/输出
pazDirNames	二级目录名称指针	输出
u32DirAmount	二级目录个数。取值范围: (0, DTCF_DIR_BUTT]	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

无。

【举例】

无。

5.4.3.3 CVI_DTCF_DeInit

【描述】

去初始化文件管理模块。

【语法】

```
int32_t CVI_DTCF_DeInit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_dtcf.h
- 库文件: libdtcf.a/libdtcf.so

【注意】

无。

【举例】

无。

5.4.3.4 CVI_DTCF_Scan

【描述】

文件目录扫描。

【语法】

```
int32_t CVI_DTCF_Scan(CVI_DTCF_DIR_E enDirs[DTCF_DIR_BUTT], uint32_t u32DirCount,  
    ↪ uint32_t *pu32FileAmount);
```

【参数】

参数名称	描述	输入/输出
enDirs	要扫描的文件目录类型的数组	输入
u32DirCount	要扫描的文件目录类型的数量	输入
pu32FileAmount	扫描到的文件数量的指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

无。

【举例】

无。

5.4.3.5 CVI_DTCF_GetFileByIndex

【描述】

文件目录扫描。

【语法】

```
int32_t CVI_DTCF_GetFileByIndex(uint32_t u32Index, char *pazFileName, uint32_t u32Length,  
↪CVI_DTCF_DIR_E *penDir);
```

【参数】

参数名称	描述	输入/输出
u32Index	文件索引	输入
pazFileName	文件名称的指针	输出
u32Length	文件名称的长度	输入
penDir	文件目录类型的指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

无。

【举例】

无。

5.4.3.6 CVI_DTCF_DelFileByIndex

【描述】

通过索引删除文件。

【语法】

```
int32_t CVI_DTCF_DelFileByIndex(uint32_t u32Index, uint32_t *pu32FileAmount);
```

【参数】

参数名称	描述	输入/输出
u32Index	文件索引	输入
pu32FileAmount	文件删除后，剩余文件的总数量	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_dtcf.h
- 库文件：libdtcf.a/libdtcf.so

【注意】

只删除内部扫描的文件信息，不删除真实文件。

【举例】

无。

5.4.3.7 CVI_DTCF_AddFile

【描述】

添加文件。

【语法】

```
int32_t CVI_DTCF_AddFile(const char *pazSrcFilePath, CVI_DTCF_DIR_E enDir);
```

【参数】

参数名称	描述	输入/输出
pazSrcFilePath	文件路径	输入
enDir	文件目录类型	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_dtcf.h
- 库文件: libdtcf.a/libdtcf.so

【注意】

扫描后才能使用该接口。

【举例】

无。

5.4.3.8 CVI_DTCF_GetOldestFileIndex

【描述】

获取最老文件索引。

【语法】

```
int32_t CVI_DTCF_GetOldestFileIndex(CVI_DTCF_DIR_E enDirs[DTCF_DIR_BUTT], uint32_t  
→ u32DirCount, uint32_t *pu32Index);
```

【参数】

参数名称	描述	输入/输出
enDirs	目录类型数据	输入
u32DirCount	目录个数	输入
pu32Index	文件索引指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_dtcf.h
- 库文件: libdtcf.a/libdtcf.so

【注意】

- 扫描后才能使用该接口。
- 文件目录类型仅支持扫描的文件类型。

【举例】

无。

5.4.3.9 CVI_DTCF_GetFileAmount**【描述】**

获取文件数量。

【语法】

```
int32_t CVI_DTCF_GetFileAmount(uint32_t *pu32FileAmount);
```

【参数】

参数名称	描述	输入/输出
pu32FileAmount	文件数量指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_dtcf.h
- 库文件: libdtcf.a/libdtcf.so

【注意】

扫描后才能使用该接口。

【举例】

无。

5.4.3.10 CVI_DTCF_GetOldestFilePath**【描述】**

获取最老文件路径。

【语法】

```
int32_t CVI_DTCF_GetOldestFilePath(CVI_DTCF_DIR_E enDir, char *pazFilePath, uint32_t u32Length);
```

【参数】

参数名称	描述	输入/输出
enDir	文件数量指针	输入
pazFilePath	最老文件路径指针	输出
u32Length	最老文件路径长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_dtcf.h
- 库文件: libdtcf.a/libdtcf.so

【注意】

无。

【举例】

无。

5.4.3.11 CVI_DTCF_CreateFilePath

【描述】

创建文件路径。

【语法】

```
int32_t CVI_DTCF_CreateFilePath(CVI_DTCF_FILE_TYPE_E enFileType, CVI_DTCF_DIR_
↳E enDir, char *pazFilePath, uint32_t u32Length);
```

【参数】

参数名称	描述	输入/输出
enFileType	文件类型	输入
enDir	文件目录类型	输入
pazFilePath	文件路径指针	输出
u32Length	文件路径长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

仅按规则创建路径，不真实创建文件。

【举例】

无。

5.4.3.12 CVI_DTCF_GetRelatedFilePath**【描述】**

获取关联文件路径。

【语法】

```
int32_t CVI_DTCF_GetRelatedFilePath(const char *pazSrcFilePath, CVI_DTCF_DIR_E enDir,
→char *pazDstFilePath, uint32_t u32Length);
```

【参数】

参数名称	描述	输入/输出
<code>pazSrcFilePath</code>	原文件路径	输入
<code>enDir</code>	关联文件目录类型	输入
<code>pazDstFilePath</code>	关联文件路径指针	输出
<code>u32Length</code>	关联文件路径长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_dtcf.h`
- 库文件: `libdtcf.a/libdtcf.so`

【注意】

无。

【举例】

无。

5.4.3.13 CVI_DTCF_GetEmrFilePath

【描述】

获取紧急录像文件路径。

【语法】

```
int32_t CVI_DTCF_GetEmrFilePath(const char *pazSrcFilePath, char *pazDstFilePath, uint32_t u32Length);
```

【参数】

参数名称	描述	输入/输出
pazSrcFilePath	原文件路径	输入
pazDstFilePath	紧急录像路径指针	输出
u32Length	紧急录像路径长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_dtcf.h
- 库文件：libdtcf.a/libdtcf.so

【注意】

无。

【举例】

无。

5.4.3.14 CVI_DTCF_GetFileDirType

【描述】

获取文件所在目录类型。

【语法】

```
int32_t CVI_DTCF_GetFileDirType(const char *pazSrcFilePath, CVI_DTCF_DIR_E *penDir);
```

【参数】

参数名称	描述	输入/输出
pazSrcFilePath	原文件路径	输入
penDir	文件目录路径指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_dtcf.h
- 库文件：libdtcf.a/libdtcf.so

【注意】

无。

【举例】

无。

5.4.4 数据类型

相关数据类型定义如下：

- `CVI_FILE_PATH_LEN_MAX`：定义文件路径的最大长度。
- `CVI_DIR_LEN_MAX`：定义目录名称的最大长度。
- `CVI_DTCF_DIR_E`：定义文件目录类型枚举。
- `CVI_DTCF_FILE_TYPE_E`：定义文件类型枚举。

5.4.4.1 CVI_FILE_PATH_LEN_MAX

【说明】

定义文件路径的最大长度。

【定义】

```
#define CVI_FILE_PATH_LEN_MAX (256)
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.4.4.2 CVI_DIR_LEN_MAX

【说明】

定义目录名称的最大长度。

【定义】

```
#define CVI_DIR_LEN_MAX (64)
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.4.4.3 CVI_DTCF_DIR_E

【说明】

定义状态机事件类型枚举。

【定义】

```
typedef enum cviDTCF_DIR_E
{
    DTCF_DIR_EMR_FRONT = 0,
    DTCF_DIR_EMR_FRONT_SUB,
    DTCF_DIR_NORM_FRONT,
    DTCF_DIR_NORM_FRONT_SUB,
    DTCF_DIR_PARK_FRONT,
    DTCF_DIR_PARK_FRONT_SUB,
    DTCF_DIR_EMR_REAR,
    DTCF_DIR_EMR_REAR_SUB,
    DTCF_DIR_NORM_REAR,
    DTCF_DIR_NORM_REAR_SUB,
    DTCF_DIR_PARK_REAR,
    DTCF_DIR_PARK_REAR_SUB,
    DTCF_DIR_PHOTO_FRONT,
    DTCF_DIR_PHOTO_REAR,
    DTCF_DIR_PHOTO_AHD,
    DTCF_DIR_EMR_AHD,
    DTCF_DIR_EMR_AHD_SUB,
    DTCF_DIR_NORMAL_AHD,
    DTCF_DIR_NORMAL_AHD_SUB,
    DTCF_DIR_PARK_AHD,
    DTCF_DIR_PARK_AHD_SUB,
    DTCF_DIR_BUTT
} CVI_DTCF_DIR_E;
```

【成员】

成员名称	描述
DTCF_DIR_EMR_FRONT	前摄像头紧急录像大码流目录
DTCF_DIR_EMR_FRONT_SUB	前摄像头紧急录像小码流目录
DTCF_DIR_NORM_FRONT	前摄像头普通录像大码流目录
DTCF_DIR_NORM_FRONT_SUB	前摄像头普通录像小码流目录
DTCF_DIR_PARK_FRONT	前摄像头停车录像大码流目录
DTCF_DIR_PARK_FRONT_SUB	前摄像头停车录像小码流目录
DTCF_DIR_EMR_REAR	后摄像头紧急录像大码流目录
DTCF_DIR_EMR_REAR_SUB	后摄像头紧急录像小码流目录
DTCF_DIR_NORM_REAR	后摄像头普通录像大码流目录
DTCF_DIR_NORM_REAR_SUB	后摄像头普通录像小码流目录
DTCF_DIR_PARK_REAR	后摄像头停车录像大码流目录
DTCF_DIR_PARK_REAR_SUB	后摄像头停车录像小码流目录
DTCF_DIR_PHOTO_FRONT	前摄像头拍照目录
DTCF_DIR_PHOTO_REAR	后摄像头拍照目录
DTCF_DIR_BUTT	无效目录类型

【注意事项】

无。

【相关数据类型及接口】

无。

5.4.4.4 CVI_DTCF_FILE_TYPE_E

【说明】

定义文件类型枚举。

【定义】

```
typedef enum cviDTCF_FILE_TYPE_E
{
    CVI_DTCF_FILE_TYPE_MP4,
    CVI_DTCF_FILE_TYPE_JPG,
    CVI_DTCF_FILE_TYPE_TS,
    CVI_DTCF_FILE_TYPE_MOV,
    CVI_DTCF_FILE_TYPE_BUTT
} CVI_DTCF_FILE_TYPE_E;
```

【成员】

成员名称	描述
CVI_DTCF_FILE_TYPE_MP4	MP4 文件类型
CVI_DTCF_FILE_TYPE_JPG	JPG 文件类型
CVI_DTCF_FILE_TYPE_TS	TS 文件类型
CVI_DTCF_FILE_TYPE_MOV	MOV 文件类型
CVI_DTCF_FILE_TYPE_BUTT	无效文件类型

【注意事项】

无。

【相关数据类型及接口】

无。

5.5 EXIF

5.5.1 概述

可交换图像文件格式常被简称为 Exif (Exchangeable image file format)，可以记录照片的属性信息和拍摄数据。Exif 可以被附加在 JPEG、TIFF、RIF 等文件之中，为其增加有关数码相机拍摄信息的内容和缩略图或图像处理软件的一些版本信息。

5.5.2 API 参考

该功能模块为用户提供以下 API：

- `CVI_EXIF_MakeExifFile`：制作 exif 文件。
- `CVI_EXIF_MakeExifParam`：制作 exif 元数据信息。
- `CVI_EXIF_MakeNewSatJpgFromBuf`：从缓冲区中制作 exif 文件。

5.5.2.1 `CVI_EXIF_MakeExifFile`

【描述】

制作 exif 文件。

【语法】

```
int32_t CVI_EXIF_MakeExifFile(char *ExifOut, uint32_t *totalLen, CVI_EXIF_FILE_INFO_S_
↪ *exifFileInfo);
```

【参数】

参数名称	描述	输入/输出
ExifOut	exif 指针	输出
totalLen	文件总大小	输出
exifFileInfo	exif 文件信息结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_exif.h
- 库文件: libexif.a / libexif.so

【注意】

无。

【举例】

无。

5.5.2.2 CVI_EXIF_MakeExifParam

【描述】

制作 exif 元数据信息。

【语法】

```
void CVI_EXIF_MakeExifParam(CVI_EXIF_FILE_INFO_S *exifFileInfo);
```

【参数】

参数名称	描述	输入/输出
exifFileInfo	exif 文件结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_exif.h
- 库文件: libexif.a / libexif.so

【注意】

无。

【举例】

无。

5.5.2.3 CVI_EXIF_MakeNewSatJpgFromBuf

【描述】

从缓冲区中制作 exif 文件。

【语法】

```
int32_t CVI_EXIF_MakeNewSatJpgFromBuf(const char *src_file, const char *dest_file, char *buf,
→int32_t size);
```

【参数】

参数名称	描述	输入/输出
src_file	原文件	输入
dest_file	目标文件	输出
buf	缓冲区	输入
size	大小	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_exif.h
- 库文件：libexif.a / libexif.so

【注意】

无。

【举例】

无。

5.5.3 数据类型

相关数据类型定义如下：

- CVI_EXIF_FILE_INFO_S：定义文件信息结构体。

5.5.3.1 CVI_EXIF_FILE_INFO_S

【说明】

定义文件信息结构体。

【定义】

```
typedef struct tagExifFileInfo
{
    char Make[32];
    char Model[32];
    char Version[32];
    char DateTime[32];
    char CopyRight[32];
    char description[32];
    uint32_t Height;
    uint32_t Width;
    uint32_t Orientation;
    uint32_t ColorSpace;
    uint32_t Process;
    uint32_t Flash;
    uint32_t FocalLengthNum;
    uint32_t FocalLengthDen;
    uint32_t ExposureTimeNum;
    uint32_t ExposureTimeDen;
    uint32_t FNumberNum;
    uint32_t FNumberDen;
    uint32_t ApertureFNumber;
    int32_t SubjectDistanceNum;
    int32_t SubjectDistanceDen;
    uint32_t CCDWidth;
    int32_t ExposureBiasNum;
    int32_t ExposureBiasDen;
    int32_t WhiteBalance;
    uint32_t MeteringMode;
    int32_t ExposureProgram;
    uint32_t ISOSpeedRatings[2];
    uint32_t FocalPlaneXResolutionNum;
    uint32_t FocalPlaneXResolutionDen;
    uint32_t FocalPlaneYResolutionNum;
    uint32_t FocalPlaneYResolutionDen;
    uint32_t FocalPlaneResolutionUnit;
    uint32_t XResolutionNum;
    uint32_t XResolutionDen;
    uint32_t YResolutionNum;
    uint32_t YResolutionDen;
    uint32_t RUnit;
    int32_t BrightnessNum;
    int32_t BrightnessDen;
    char UserComments[150];
    char GpsLatitudeRef;
    unsigned char GpsLatitude[20];
    char GpsLongitudeRef;
    unsigned char GpsLongitude[20];
    unsigned char GpsAltitude[10];
}
```

(下页继续)

(续上页)

```
bool param;
} CVI_EXIF_FILE_INFO_S;
```

【成员】

成员名称	描述	成员名称	描述	成员名称	描述
Make	制造商	FNumber-Num	焦距（分子）	FocalPlaneResolutionUnit	分辨率单位
Model	型号	FNumber-Den	焦距（分母）	XResolution-Num	水平分辨率（分子）
Version	版本号	ApertureFNumber	光圈	XResolution-Den	水平分辨率（分母）
DateTime	日期时间	SubjectDistanceNum	物距（分子）	YResolution-Num	垂直分辨率（分子）
CopyRight	版权信息	SubjectDistanceDen	物距（分母）	YResolution-Den	垂直分辨率（分母）
description	描述	CCDWidth	CCD 宽度	RUnit	分辨率单位
Height	高	ExposureBiasNum	曝光补偿（分子）	Brightness-Num	亮度（分子）
Width	宽	ExposureBiasDen	曝光补偿（分母）	Brightness-Den	亮度（分母）
Orientation	方向	WhiteBalance	白平衡	UserComments	用户注释
ColorSpace	颜色空间	Metering-Mode	测光方式	GpsLatitudeRef	纬度方位
Process	处理	Exposure-Program	曝光方式	GpsLatitude	GPS 纬度
Flash	闪光	ISOSpeedRatings	ISO 感光度	GpsLongitudeRef	经度方位
FocalLength-Num	拍摄时焦距（分子）	FocalPlaneXResolutionNum	水平分辨率（分子）	GpsLongitude	GPS 经度
FocalLength-Den	拍摄时焦距（分母）	FocalPlaneXResolutionDen	水平分辨率（分母）	GpsAltitude	GPS 高度
ExposureTimeNum	曝光时间（分子）	FocalPlaneYResolutionNum	垂直分辨率（分子）	param	参数
ExposureTimeDen	曝光时间（分母）	FocalPlaneYResolutionDen	垂直分辨率（分母）		

【注意事项】

无。

【相关数据类型及接口】

无。

5.6 LOG

5.6.1 概述

日志 (LOG) 模块用于程序开发过程中的调试和运行记录，帮助开发和运维人员根据日志记录定位程序的运行状态。

5.6.2 API 参考

该功能模块为用户提供以下 API:

- `CVI_LOG_INIT` : 初始化日志模块。
 - `CVI_LOG_SET_LEVEL` : 设置日志级别。
 - `CVI_LOG_SET_TAG` : 设置日志打印前缀。
-

5.6.2.1 `CVI_LOG_INIT`

【描述】

初始化日志模块。

【语法】

```
void CVI_LOG_INIT(void);
```

【参数】

无。

【返回值】

无。

【需求】

- 头文件: `cvi_log.h`
- 库文件: `libcvi_log.a/libcvi_log.so`

【注意】

无。

【举例】

无。

5.6.2.2 CVI_LOG_SET_LEVEL

【描述】

设置日志级别。

【语法】

```
void CVI_LOG_SET_LEVEL(const int32_t lvl);
```

【参数】

参数名称	描述	输入/输出
lvl	日志级别	输入

【返回值】

无。

【需求】

- 头文件: cvi_log.h
- 库文件: libcvi_log.a/libcvi_log.so

【注意】

无。

【举例】

无。

5.6.2.3 CVI_LOG_SET_TAG

【描述】

设置日志前缀。

【语法】

```
void CVI_LOG_SET_TAG(const char *const tag);
```

【参数】

参数名称	描述	输入/输出
tag	日志前缀	输入

【返回值】

无。

【需求】

- 头文件: cvi_log.h
- 库文件: libcvi_log.a/libcvi_log.so

【注意】

无。

【举例】

无。

5.6.3 数据类型

日志级别定义如下：

- CVI_LOG_VERBOSE : 1
- CVI_LOG_DEBUG : 2
- CVI_LOG_INFO : 3
- CVI_LOG_WARN : 4
- CVI_LOG_ERROR : 5
- CVI_LOG_FATAL : 6
- CVI_LOG_NONE : 0xFF

5.7 QUEUE

5.7.1 概述

queue 模块提供了队列操作的一系列接口。

5.7.2 API 参考

该功能模块为用户提供以下 API：

- CVI_QUEUE_Create : 创建队列。
 - CVI_QUEUE_Destroy : 销毁队列。
 - CVI_QUEUE_Clear : 清空队列。
 - CVI_QUEUE_GetLen : 获得队列长度。
 - CVI_QUEUE_Push : 加入队列。
 - CVI_QUEUE_Pop : 弹出队列。
-

5.7.2.1 CVI_QUEUE_Create

【描述】

创建队列。

【语法】

```
CVI_QUEUE_HANDLE_T CVI_QUEUE_Create(uint32_t nodeSize, uint32_t maxLen);
```

【参数】

参数名称	描述	输入/输出
nodeSize	队列节点大小	输入
maxLen	队列最大长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_queue.h
- 库文件: libcvi_queue.a/libcvi_queue.so

【注意】

无。

【举例】

无。

5.7.2.2 CVI_QUEUE_Destroy

【描述】

创建队列。

【语法】

```
void CVI_QUEUE_Destroy(CVI_QUEUE_HANDLE_T queueHdl);
```

【参数】

参数名称	描述	输入/输出
queueHdl	队列句柄	输入

【返回值】

无。

【需求】

- 头文件: `cvi_queue.h`
- 库文件: `libcvi_queue.a/libcvi_queue.so`

【注意】

无。

【举例】

无。

5.7.2.3 CVI_QUEUE_Clear

【描述】

清空队列。

【语法】

```
void CVI_QUEUE_Clear(CVI_QUEUE_HANDLE_T queueHdl);
```

【参数】

参数名称	描述	输入/输出
queueHdl	队列句柄	输入

【返回值】

无。

【需求】

- 头文件: `cvi_queue.h`
- 库文件: `libcvi_queue.a/libcvi_queue.so`

【注意】

无。

【举例】

无。

5.7.2.4 CVI_QUEUE_GetLen

【描述】

获得队列长度。

【语法】

```
int32_t CVI_QUEUE_GetLen(CVI_QUEUE_HANDLE_T queueHdl);
```

【参数】

参数名称	描述	输入/输出
queueHdl	队列句柄	输入

【返回值】

返回值	描述
> = 0	队列长度
-1	失败

【需求】

- 头文件：cvi_queue.h
- 库文件：libcvi_queue.a/libcvi_queue.so

【注意】

无。

【举例】

无。

5.7.2.5 CVI_QUEUE_Push

【描述】

加入队列。

【语法】

```
int32_t CVI_QUEUE_Push(CVI_QUEUE_HANDLE_T queueHdl, const void *node);
```

【参数】

参数名称	描述	输入/输出
queueHdl	队列句柄	输入
node	队列节点	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_queue.h
- 库文件：libcvi_queue.a/libcvi_queue.so

【注意】

无。

【举例】

无。

5.7.2.6 CVI_QUEUE_Pop

【描述】

弹出队列。

【语法】

```
int32_t CVI_QUEUE_Pop(CVI_QUEUE_HANDLE_T queueHdl, void *node);
```

【参数】

参数名称	描述	输入/输出
queueHdl	队列句柄	输入
node	队列节点	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_queue.h
- 库文件：libcvi_queue.a/libcvi_queue.so

【注意】

无。

【举例】

无。

5.7.3 数据类型

相关数据类型定义如下：

- `CVI_QUEUE_HANDLE_T`：定义队列句柄。

5.7.3.1 CVI_QUEUE_HANDLE_T

【说明】

定义队列句柄。

【定义】

```
typedef void * CVI_QUEUE_HANDLE_T;
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.8 SIGNAL_SLOT

5.8.1 概述

信号槽（Signal-Slot）机制是一种在软件开发中常用的设计模式，用于实现对象间的通信和事件处理。信号槽机制通过定义信号（Signal）和槽（Slot）来实现对象间的解耦和事件传递。信号是对象发出的一种特定事件，槽是用于响应信号的特定函数或方法。当一个对象发出信号时，与之相关联的槽会被自动调用，从而实现了对象间的通信和协作。

- 在信号槽机制中，一个对象可以声明一个或多个信号，用于指示它可以发出的事件。另外，一个对象也可以定义一个或多个槽，用于指定它可以响应的事件处理函数。信号和槽通过特定的语法和宏来进行声明和连接，从而建立起对象间的连接关系。
- 当一个信号被触发时，所有与之相关联的槽都会被调用。这种松耦合的设计使得对象间的通信更加灵活和可扩展。一个信号可以连接到多个槽，而一个槽也可以响应多个信号，从而实现了多对多的关系。

5.8.2 API 参考

该功能模块为用户提供以下 API：

- `CVI_SIGNAL_InitByType`：初始化类型信号。
- `CVI_SIGNAL_Init`：初始化信号。
- `CVI_SLOT_Init`：初始化槽。
- `CVI_SIGNAL_Connect`：信号连接槽。
- `CVI_SIGNAL_Emit`：发射信号。
- `CVI_SIGNAL_Disconnect`：信号取消连接槽。
- `CVI_SIGNAL_Deinit`：去初始化信号。
- `CVI_SLOT_Deinit`：去初始化槽。

5.8.2.1 CVI_SIGNAL_InitByType

【描述】

初始化信号，底层调用 CVI_SLOT_Init。

【语法】

```
void CVI_SIGNAL_InitByType(CVI_SIGNAL_S *cvi_signal, CVI_SIGNAL_SLOT_TYPE_E_  
↪type);
```

【参数】

参数名称	描述	输入/输出
cvi_signal	信号结构体指针	输入
type	类型	输入

【返回值】

无。

【需求】

- 头文件：cvi_signal_slot.h
- 库文件：libcvi_signal_slot.a/libcvi_signal_slot.so

【注意】

初始类型需与槽处理程序的类型相同。

【举例】

无。

5.8.2.2 CVI_SIGNAL_Init

【描述】

初始化信号。

【语法】

```
void CVI_SIGNAL_Init(CVI_SIGNAL_S *cvi_signal);
```

【参数】

参数名称	描述	输入/输出
cvi_signal	信号结构体指针	输入

【返回值】

无。

【需求】

- 头文件: cvl_signal_slot.h
- 库文件: libcvl_signal_slot.a/libcvl_signal_slot.so

【注意】

无。

【举例】

无。

5.8.2.3 CVI_SLOT_Init**【描述】**

初始化槽。

【语法】

```
#define CVI_SLOT_Init(SLOT, HANLDE, HANDLER) \
    _Generic((HANDLER), \
        CVI_INT_SLOT_HANLDER: CVI_INT_SLOT_NONE_Init, \
        CVI_INT_SLOT_VOID_HANLDER: CVI_INT_SLOT_VOID_Init, \
        CVI_INT_SLOT_BOOL_HANLDER: CVI_INT_SLOT_BOOL_Init, \
        CVI_INT_SLOT_INT_HANLDER: CVI_INT_SLOT_INT_Init, \
        CVI_INT_SLOT_INT64_HANLDER: CVI_INT_SLOT_INT64_Init, \
        CVI_INT_SLOT_FLOAT_HANLDER: CVI_INT_SLOT_FLOAT_Init, \
        CVI_INT_SLOT_STRING_HANLDER: CVI_INT_SLOT_STRING_Init, \
        CVI_INT_SLOT_INT_INT_HANLDER: CVI_INT_SLOT_INT_INT_Init, \
        CVI_INT_SLOT_UINT32_UINT32_HANLDER: CVI_INT_SLOT_UINT32_UINT32_Init, \
        CVI_VOID_SLOT_HANLDER: CVI_VOID_SLOT_NONE_Init, \
        CVI_VOID_SLOT_VOID_HANLDER: CVI_VOID_SLOT_VOID_Init, \
        CVI_VOID_SLOT_BOOL_HANLDER: CVI_VOID_SLOT_BOOL_Init, \
        CVI_VOID_SLOT_INT_HANLDER: CVI_VOID_SLOT_INT_Init, \
        CVI_VOID_SLOT_INT64_HANLDER: CVI_VOID_SLOT_INT64_Init, \
        CVI_VOID_SLOT_FLOAT_HANLDER: CVI_VOID_SLOT_FLOAT_Init, \
        CVI_VOID_SLOT_STRING_HANLDER: CVI_VOID_SLOT_STRING_Init, \
        CVI_VOID_SLOT_INT_INT_HANLDER: CVI_VOID_SLOT_INT_INT_Init, \
        CVI_VOID_SLOT_UINT32_UINT32_HANLDER: CVI_VOID_SLOT_UINT32_UINT32_Init, \
        default: CVI_INT_SLOT_NONE_Init \
    )(SLOT, HANLDE, HANDLER)
```

【参数】

参数名称	描述	输入/输出
SLOT	槽结构体指针	输入
HANLDE	执行函数	输入
HANDLER	槽句柄	输入

【返回值】

无。

【需求】

- 头文件: `cvi_signal_slot.h`
- 库文件: `libcvi_signal_slot.a/libcvi_signal_slot.so`

【注意】

无。

【举例】

无。

5.8.2.4 CVI_SIGNAL_Connect**【描述】**

信号连接槽。

【语法】

```
int32_t CVI_SIGNAL_Connect(CVI_SIGNAL_S cvi_signal, CVI_SLOT_S cvi_slot);
```

【参数】

参数名称	描述	输入/输出
<code>cvi_signal</code>	信号指针	输入
<code>cvi_slot</code>	槽指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_signal_slot.h`
- 库文件: `libcvi_signal_slot.a/libcvi_signal_slot.so`

【注意】

信号的类型应与插槽处理程序的类型相同，否则将连接失败。

【举例】

无。

5.8.2.5 CVI_SIGNAL_Emit

【描述】

发射信号。

【语法】

```
void CVI_SIGNAL_EmitCpp(CVI_SIGNAL_S cvi_signal) // 重载函数，以该声明为例
```

【参数】

参数名称	描述	输入/输出
cvi_signal	信号指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_signal_slot.h
- 库文件：libcvi_signal_slot.a/libcvi_signal_slot.so

【注意】

无。

【举例】

无。

5.8.2.6 CVI_SIGNAL_Disconnect

【描述】

信号取消连接槽。

【语法】

```
int32_t CVI_SIGNAL_Disconnect(CVI_SIGNAL_S cvi_signal, CVI_SLOT_S cvi_slot);
```

【参数】

参数名称	描述	输入/输出
cvi_signal	信号指针	输入
cvi_slot	槽指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_signal_slot.h`
- 库文件: `libcvi_signal_slot.a/libcvi_signal_slot.so`

【注意】

无。

【举例】

无。

5.8.2.7 CVI_SIGNAL_Deinit

【描述】

去初始化信号。

【语法】

```
void CVI_SIGNAL_Deinit(CVI_SIGNAL_S *cvi_signal);
```

【参数】

参数名称	描述	输入/输出
<code>cvi_signal</code>	信号指针	输入

【返回值】

无。

【需求】

- 头文件: `cvi_signal_slot.h`
- 库文件: `libcvi_signal_slot.a/libcvi_signal_slot.so`

【注意】

无。

【举例】

无。

5.8.2.8 CVI_SLOT_Deinit

【描述】

去初始化槽。

【语法】

```
void CVI_SLOT_Deinit(CVI_SLOT_S *cvi_slot);
```

【参数】

参数名称	描述	输入/输出
cvi_slot	槽指针	输入

【返回值】

无。

【需求】

- 头文件：cvi_signal_slot.h
- 库文件：libcvi_signal_slot.a/libcvi_signal_slot.so

【注意】

无。

【举例】

无。

5.8.3 数据类型

相关数据类型定义如下：

- CVI_SIGNAL_SLOT_TYPE_E：定义信号槽类型枚举。
 - CVI_SIGNAL_S：定义信号结构体。
 - CVI_SLOT_S：定义槽结构体。
-

5.8.3.1 CVI_SIGNAL_SLOT_TYPE_E

【说明】

定义信号槽类型枚举。

【定义】

```
typedef enum
{
    CVI_SIGNAL_SLOT_TYPE_NONE,
    CVI_SIGNAL_SLOT_TYPE_VOID,
    CVI_SIGNAL_SLOT_TYPE_BOOL,
    CVI_SIGNAL_SLOT_TYPE_INT,
    CVI_SIGNAL_SLOT_TYPE_INT64,
    CVI_SIGNAL_SLOT_TYPE_FLOAT,
    CVI_SIGNAL_SLOT_TYPE_STRING,
    CVI_SIGNAL_SLOT_TYPE_INT_INT,
    CVI_SIGNAL_SLOT_TYPE_UINT32_UINT32
} CVI_SIGNAL_SLOT_TYPE_E;
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.8.3.2 CVI_SIGNAL_S

【说明】

定义信号结构体。

【定义】

```
typedef struct
{
    CVI_SIGNAL_SLOT_TYPE_E type;
    void *signal;
} CVI_SIGNAL_S;
```

【成员】

成员名称	描述
type	类型
signal	信号指针

【注意事项】

无。

【相关数据类型及接口】

无。

5.8.3.3 CVI_SLOT_S

【说明】

定义槽结构体。

【定义】

```
typedef struct
{
    CVI_SIGNAL_SLOT_TYPE_E type;
    void *slot;
} CVI_SLOT_S;
```

【成员】

成员名称	描述
type	类型
slot	槽指针

【注意事项】

无。

【相关数据类型及接口】

无。

5.9 STORAGE

5.9.1 概述

Storage 模块负责管理存储设备，为上层提供存储设备状态监听和信息获取的功能。Storage 模块的上下文如图 5-6 所示。

Storage 模块内部需要 FSTool 的支持，FSTool (File System Tool) 模块为用户提供检查和格式化文件系统的功能。在录像或拍照业务中，可能会出现热插拔、异常掉电现象，导致文件系统异常。FSTool 模块负责检查和修复文件系统异常，保证文件系统能正常工作。

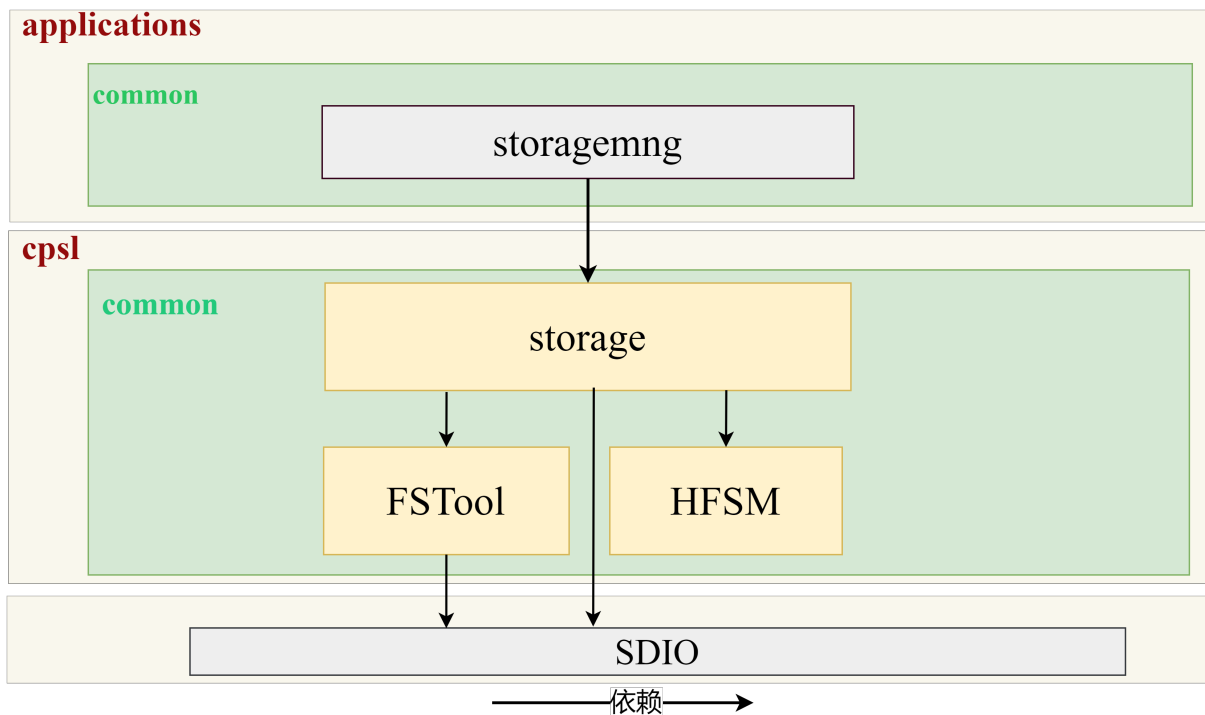


图 5.6: Storage 模块系统框图

5.9.2 API 参考

STORAGE 模块为用户提供以下 API:

- **CVI_STG_Init**: 初始化 Storage 模块。
- **CVI_STG_DeInit**: 去初始化 Storage 模块。
- **CVI_STG_GetFsInfo**: 获得文件系统信息。
- **CVI_STG_GetSDInfo**: 获得 SD 信息。
- **CVI_STG_GetDevState**: 获得设备状态。
- **CVI_STG_Umount**: 文件系统卸载。
- **CVI_STG_Mount**: 文件系统挂载。
- **CVI_STG_Format**: 文件系统格式化。
- **CVI_STG_FormatWithLabel**: 带标签的文件系统格式化。
- **CVI_STG_GetInfo**: 获得设备信息。
- **CVI_STG_RepairFAT32**: 修复 FAT32 文件系统。
- **CVI_STG_DetectPartition**: 检测分区。
- **CVI_STG_TestPartition**: 测试分区。

5.9.2.1 CVI_STG_Init

【描述】

初始化 Storage 模块。

【语法】

```
int32_t CVI_STG_Init(STG_DEVINFO_S *stgdev, CVI_STG_HANDLE_T *stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stgdev	设备信息指针	输入
stg_hdl	Storage 句柄指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

无。

【举例】

无。

5.9.2.2 CVI_STG_DeInit

【描述】

去初始化 Storage 模块。

【语法】

```
int32_t CVI_STG_DeInit(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_stg.h`
- 库文件: `libcvi_stg.a/libcvi_stg.so`

【注意】

无。

【举例】

无。

5.9.2.3 CVI_STG_GetFsInfo

【描述】

获得文件系统信息。

【语法】

```
int32_t CVI_STG_GetFsInfo(CVI_STG_HANDLE_T stg_hdl, CVI_STG_FS_INFO_S *fsinfo);
```

【参数】

参数名称	描述	输入/输出
<code>stg_hdl</code>	Storage 句柄	输入
<code>fsinfo</code>	文件系统信息指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_stg.h`
- 库文件: `libcvi_stg.a/libcvi_stg.so`

【注意】

内部调用 `fstool` 的 `Stg_FsTool_GetFsType`。

【举例】

无。

5.9.2.4 CVI_STG_GetSDInfo

【描述】

获得 SD 信息。

【语法】

```
int32_t CVI_STG_GetSDInfo(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

无。

【举例】

无。

5.9.2.5 CVI_STG_GetDevState

【描述】

获得设备状态。

【语法】

```
int32_t CVI_STG_GetDevState(CVI_STG_HANDLE_T stg_hdl, CVI_STG_DEV_STATE_E  
→*state);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入
state	设备状态指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

无。

【举例】

无。

5.9.2.6 CVI_STG_Umount**【描述】**

文件系统卸载。

【语法】

```
int32_t CVI_STG_Umount(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_Mount。

【举例】

无。

5.9.2.7 CVI_STG_Mount

【描述】

文件系统挂载。

【语法】

```
int32_t CVI_STG_Mount(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_Umount。

【举例】

无。

5.9.2.8 CVI_STG_Format

【描述】

格式化。

【语法】

```
int32_t CVI_STG_Format(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvt_stg.h, fstool.h
- 库文件: libcvt_stg.a/libcvt_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_Format。

【举例】

无。

5.9.2.9 CVI_STG_FormatWithLabel

【描述】

带标签的格式化。

【语法】

```
int32_t CVI_STG_FormatWithLabel(CVI_STG_HANDLE_T stg_hdl, char *labelname);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入
stg_hdl	名称	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvt_stg.h, fstool.h
- 库文件: libcvt_stg.a/libcvt_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_Format_with_Label。

【举例】

无。

5.9.2.10 CVI_STG_GetInfo

【描述】

获取设备信息。

【语法】

```
int32_t CVI_STG_GetInfo(CVI_STG_HANDLE_T stg_hdl, CVI_STG_DEV_INFO_S *pInfo);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入
pInfo	设备信息指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

无。

【举例】

无。

5.9.2.11 CVI_STG_RepairFAT32

【描述】

修复 FAT32 文件系统。

【语法】

```
int32_t CVI_STG_RepairFAT32(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

无。

【举例】

无。

5.9.2.12 CVI_STG_DetectPartition

【描述】

检测分区状态。

【语法】

```
int32_t CVI_STG_DetectPartition(CVI_STG_HANDLE_T stg_hdl);
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_stg.h, fstool.h
- 库文件: libcvi_stg.a/libcvi_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_DetectPartition。

【举例】

无。

5.9.2.13 CVI_STG_TestPartition

【描述】

测试分区状态。

【语法】

```
int32_t CVI_STG_TestPartition(CVI_STG_HANDLE_T stg_hdl)
```

【参数】

参数名称	描述	输入/输出
stg_hdl	Storage 句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_stg.h, fstool.h
- 库文件：libcvi_stg.a/libcvi_stg.so

【注意】

内部调用 fstool 的 Stg_Fstool_TestPartition。

【举例】

无。

5.9.3 数据类型

相关数据类型定义如下：

- CVI_STG_FS_INFO_S：定义文件系统信息结构体。
- CVI_STG_DEV_STATE_E：定义设备状态枚举。
- CVI_STG_FS_TYPE_E：定义文件系统类型枚举。
- CVI_STG_STATE_E：定义 Storage 实例状态枚举。
- CVI_STG_DEV_INFO_S：定义设备信息结构体，对外使用。
- CVI_STG_TRANSMISSION_SPEED_E：定义传输速度枚举。
- STG_DEVINFO_S：定义设备信息结构体，对内使用。

5.9.3.1 CVI_STG_FS_INFO_S

【说明】

定义文件系统信息。

【定义】

```
typedef struct {  
    uint64_t u64ClusterSize; /**<DEV partition cluster size(unit bytes) */  
    uint64_t u64TotalSize; /**<DEV partition total space size(unit bytes) */  
    uint64_t u64AvailableSize; /**<DEV partition free space size(unit bytes) */  
    uint64_t u64UsedSize; /**<DEV partition used space size(unit bytes) */  
} CVI_STG_FS_INFO_S;
```

【成员】

成员名称	描述
u64ClusterSize	设备分区簇大小
u64TotalSize	设备总空间大小
u64AvailableSize	设备空闲空间大小
u64UsedSize	设备分区使用空间大小

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.2 CVI_STG_DEV_STATE_E

【说明】

定义设备状态枚举。

【定义】

```
typedef enum tagSTG_DEV_STATE_E {  
    CVI_STG_DEV_STATE_UNPLUGGED = 0x00,  
    CVI_STG_DEV_STATE_CONNECTING,  
    CVI_STG_DEV_STATE_CONNECTED,  
    CVI_STG_DEV_STATE_IDLE  
} CVI_STG_DEV_STATE_E;
```

【成员】

成员名称	描述
CVI_STG_DEV_STATE_UNPLUGGED	设备拔出
CVI_STG_DEV_STATE_CONNECTING	设备连接中
CVI_STG_DEV_STATE_CONNECTED	设备已连接
CVI_STG_DEV_STATE_IDLE	dev 初始状态

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.3 CVI_STG_FS_TYPE_E

【说明】

定义文件系统类型枚举。

【定义】

```
typedef enum tagSTG_FS_TYPE_E {
    CVI_STG_FS_TYPE_FAT32 = 0x00,
    CVI_STG_FS_TYPE_EXFAT,
    CVI_STG_FS_TYPE_UNKNOWN
} CVI_STG_FS_TYPE_E;
```

【成员】

成员名称	描述
CVI_STG_FS_TYPE_FAT32	FAT32 文件系统
CVI_STG_FS_TYPE_EXFAT	EXFAT 文件系统
CVI_STG_FS_TYPE_UNKNOWN	未知文件系统

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.4 CVI_STG_STATE_E

【说明】

定义 Storage 实例状态。

【定义】

```
typedef enum STORAGE_STATE_E {
    CVI_STG_STATE_DEV_UNPLUGGED = 0x00, /**<device already plugout */
    CVI_STG_STATE_DEV_CONNECTING,      /**<device connecting */
    CVI_STG_STATE_DEV_ERROR,            /**<sd card error */
    CVI_STG_STATE_FS_CHECKING,          /**<device doing fscheck */
    CVI_STG_STATE_FS_CHECK_FAILED,      /**<device fscheck failed */
    CVI_STG_STATE_FS_EXCEPTION,         /**<device file system exception */
    CVI_STG_STATE_MOUNTED,              /**<device mounted */
    CVI_STG_STATE_UNMOUNTED,            /**<device unmounted */
    CVI_STG_STATE_MOUNT_FAILED,         /**<device mount fail */
}
```

(下页继续)

(续上页)

```

    CVI_STG_STATE_FORMATING,          /**<device foramating */
    CVI_STG_STATE_FORMAT_SUCCEEDED,    /**<device foramat succeeded */
    CVI_STG_STATE_FORMAT_FAILED,       /**<device foramat failed */
    CVI_STG_STATE_READ_ONLY,          /**<device read only */
    CVI_STG_STATE_IDLE                 /**init state */
} CVI_STG_STATE_E;

```

【成员】

成员名称	描述
CVI_STG_STATE_DEV_UNPLUGGED	存储设备已拔出
CVI_STG_STATE_DEV_CONNECTING	存储设备连接中
CVI_STG_STATE_DEV_ERROR	存储设备损坏
CVI_STG_STATE_FS_CHECKING	文件系统检查中
CVI_STG_STATE_FS_CHECK_FAILED	文件系统检查失败
CVI_STG_STATE_FS_EXCEPTION	文件系统异常
CVI_STG_STATE_MOUNTED	文件系统已挂载
CVI_STG_STATE_UNMOUNTED	文件系统退出挂载
CVI_STG_STATE_MOUNT_FAILED	文件系统挂载失败
CVI_STG_STATE_FORMATING	文件系统格式化中
CVI_STG_STATE_FORMAT_SUCCEEDED	文件系统格式化成功
CVI_STG_STATE_FORMAT_FAILED	文件系统格式化失败
CVI_STG_STATE_READ_ONLY	文件系统只读
CVI_STG_STATE_IDLE	空闲状态，Storage 创建后的状态

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.5 CVI_STG_DEV_INFO_S**【说明】**

定义设备信息结构体，对外使用。

【定义】

```

typedef struct cviSTG_DEV_INFO_S {
    char aszDevType[CVI_STG_PATH_LEN_MAX]; /**device type,such as SD or MMC */
    char aszWorkMode[CVI_STG_PATH_LEN_MAX]; /**device work mode */
    CVI_STG_TRANSMISSION_SPEED_E enTranSpeed; /**device transmission rate info */
    uint32_t szErrCnt;
} CVI_STG_DEV_INFO_S;

```

【成员】

成员名称	描述
aszDevType	设备类型
aszWorkMode	设备工作模式
enTranSpeed	设备传输速度
szErrCnt	错误数量

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.6 CVI_STG_TRANSMISSION_SPEED_E

【说明】

定义传输速度枚举。

【定义】

```
typedef enum cviSTG_TRANSMISSION_SPEED_E {
    CVI_STG_TRANSMISSION_SPEED_1_4M = 0x00, /**1-4 MB/s */
    CVI_STG_TRANSMISSION_SPEED_4_10M,      /**4-10 MB/s */
    CVI_STG_TRANSMISSION_SPEED_10_30M,      /**10-30 MB/s */
    CVI_STG_TRANSMISSION_SPEED_30_50M,      /**30-50 MB/s */
    CVI_STG_TRANSMISSION_SPEED_50_100M,     /**50-100MB/s */
    CVI_STG_TRANSMISSION_SPEED_EXCEED_100M, /**100MB/s and faster */
    CVI_STG_TRANSMISSION_SPEED_BUTT        /**others*/
} CVI_STG_TRANSMISSION_SPEED_E;
```

【成员】

成员名称	描述
CVI_STG_TRANSMISSION_SPEED_1_4M	存储设备存储速率在 1~4MB/S
CVI_STG_TRANSMISSION_SPEED_4_10M	存储设备存储速率在 4~10MB/S
CVI_STG_TRANSMISSION_SPEED_10_30M	存储设备存储速率在 10~30MB/S
CVI_STG_TRANSMISSION_SPEED_30_50M	存储设备存储速率在 30~50MB/S
CVI_STG_TRANSMISSION_SPEED_50_100M	存储设备存储速率在 50~100MB/S
CVI_STG_TRANSMISSION_SPEED_EXCEED_100M	存储设备存储速率在 100MB/S 以上
CVI_STG_TRANSMISSION_SPEED_BUTT	默认初始状态

【注意事项】

无。

【相关数据类型及接口】

无。

5.9.3.7 STG_DEVINFO_S

【说明】

定义设备信息结构体, 对内使用

【定义】

```
typedef struct {  
    CVI_STG_DEV_STATE_E devState;  
    CVI_STG_STATE_E stgState;  
    char fsType[STG_PATH_LEN_MAX];  
    CVI_STG_FS_TYPE_E fsType_e;  
    char workMode[STG_DEVINFO_PROC_LINE_LEN];  
    char aszDevPort[STG_DEVINFO_PROC_LINE_LEN];  
    char speedClass[STG_DEVINFO_PROC_LINE_LEN];  
    char speedGrade[STG_DEVINFO_PROC_LINE_LEN];  
    char aszDevType[STG_DEVINFO_PROC_LINE_LEN];  
    char aszDevPath[STG_PATH_LEN_MAX];  
    char aszMntPath[STG_PATH_LEN_MAX];  
    char aszErrCnt[STG_PATH_LEN_MAX];  
} STG_DEVINFO_S;
```

【成员】

成员名称	描述
devState	设备状态
stgState	存储状态
fsType	文件系统类型
workMode	工作模式
aszDevPort	存储设备端口号, 即存储设备对应 MCI 卡槽号
speedClass	写入速度等级
speedGrade	写入速度等级高、低级别
aszDevType	设备类型
aszDevPath	设备路径
aszMntPath	挂载路径
aszErrCnt	错误个数

【注意事项】

无。

【相关数据类型及接口】

无。

5.10 SYSUTILS

5.10.1 概述

sysutils 模块主要提供一些系统函数的调用。

5.10.2 API 参考

该功能模块为用户提供以下 API:

- `cvi_insmod` : 加载驱动文件。
- `cvi_rmmmod` : 卸载驱动文件。
- `cvi_PathIsDirectory` : 检测目标路径是否是目录。
- `cvi_rmdir` : 删除目录。
- `cvi_mkdir` : 新增目录
- `cvi_system` : 系统调用。
- `cvi_usleep` : 任务睡眠。
- `cvi_du` : 显示目录大小。
- `cvi_async` : 数据同步。

5.10.2.1 `cvi_insmod`

【描述】

加载 ko。

【语法】

```
int32_t cvi_insmod(const char *pszPath, const char *pszOptions);
```

【参数】

参数名称	描述	输入/输出
pszPath	驱动路径指针	输入
pszOptions	参数指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_sysutils.h`
- 库文件: `libcvi_sysutils.a/libcvi_sysutils.so`

【注意】

无。

【举例】

无。

5.10.2.2 cvi_rmmod**【描述】**

卸载 ko。

【语法】

```
int32_t cvi_rmmod(const char *pszPath);
```

【参数】

参数名称	描述	输入/输出
pszPath	驱动路径指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_sysutils.h`
- 库文件: `libcvi_sysutils.a/libcvi_sysutils.so`

【注意】

无。

【举例】

无。

5.10.2.3 cvi_PathIsDirectory

【描述】

检测路径是否是目录。

【语法】

```
int32_t cvi_PathIsDirectory(const char *pszPath);
```

【参数】

参数名称	描述	输入/输出
pszPath	路径指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_sysutils.h
- 库文件: libcvi_sysutils.a/libcvi_sysutils.so

【注意】

无。

【举例】

无。

5.10.2.4 cvi_rmdir

【描述】

删除目录。

【语法】

```
int32_t cvi_rmdir(const char *pszPath);
```

【参数】

参数名称	描述	输入/输出
pszPath	目录路径指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_sysutils.h`
- 库文件: `libcvi_sysutils.a/libcvi_sysutils.so`

【注意】

无。

【举例】

无。

5.10.2.5 `cvi_mkdir`

【描述】

新增目录。

【语法】

```
int32_t cvi_mkdir(const char *pszPath, mode_t mode);
```

【参数】

参数名称	描述	输入/输出
<code>pszPath</code>	目录路径指针	输入
<code>mode</code>	创建目录的模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_sysutils.h`
- 库文件: `libcvi_sysutils.a/libcvi_sysutils.so`

【注意】

无。

【举例】

无。

5.10.2.6 cvi_system

【描述】

系统调用。

【语法】

```
int32_t cvi_system(const char *pszCmd);
```

【参数】

参数名称	描述	输入/输出
pszCmd	命令行指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_sysutils.h
- 库文件: libcvi_sysutils.a/libcvi_sysutils.so

【注意】

无。

【举例】

无。

5.10.2.7 cvi_usleep

【描述】

当前进程睡眠。

【语法】

```
int32_t cvi_usleep(uint32_t usec);
```

【参数】

参数名称	描述	输入/输出
usec	时间 (us)	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_sysutils.h
- 库文件: libcvi_sysutils.a/libcvi_sysutils.so

【注意】

无。

【举例】

无。

5.10.2.8 cvi_du

【描述】

获得目录大小。

【语法】

```
int32_t cvi_du(const char *pszPath, uint64_t *pu64Size_KB);
```

【参数】

参数名称	描述	输入/输出
pszPath	路径指针	输入
pu64Size_KB	文件大小指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_sysutils.h
- 库文件: libcvi_sysutils.a/libcvi_sysutils.so

【注意】

无。

【举例】

无。

5.10.2.9 cvi_async

【描述】

数据同步写入磁盘。

【语法】

```
int32_t cvi_async(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_sysutils.h
- 库文件：libcvi_sysutils.a/libcvi_sysutils.so

【注意】

无。

【举例】

无。

5.10.3 数据类型

相关数据类型定义如下：

- CVI_MAX_PATH_LEN：定义路径最大长度。

5.10.3.1 CVI_MAX_PATH_LEN

【说明】

定义路径最大长度。

【定义】

```
#define CVI_MAX_PATH_LEN (64)
```

【注意事项】

无。

【相关数据类型及接口】

无。

5.11 TIMER

5.11.1 概述

定时器模块提供定时任务功能，其主要完成的任务类型包括：

- 单次定时任务，创建定时任务之后开始计时，定时时间到达后执行一次定时动作，然后自动销毁定时器。
- 周期性定时任务，创建定时任务之后开始计时，定时动作以指定间隔周期性执行，直至定时任务销毁。

5.11.2 API 参考

该功能模块为用户提供以下 API：

- `CVI_Timer_Init`：初始化定时任务组。
- `CVI_Timer_DeInit`：去初始化定时任务组。
- `CVI_Timer_Create`：创建定时任务。
- `CVI_Timer_Reset`：重置定时任务。
- `CVI_Timer_SetTickValue`：设置定时任务组中所有任务的计时精度。
- `CVI_Timer_Destroy`：销毁定时任务。
- `CVI_Timer_CleanUp`：清除定时任务组中的所有定时任务。
- `CVI_Timer_SetPeriodicAttr`：设置定时任务属性。
- `CVI_Timer_GetPastTime`：获取定时任务当前计时。

5.11.2.1 `CVI_Timer_Init`

【描述】

初始化定时器。

【语法】

```
int32_t CVI_Timer_Init(bool bBlock);
```

【参数】

参数名称	描述	输入/输出
bBlock	定时任务组类型标识符，阻塞或非阻塞	输入

【返回值】

返回值	描述
文件描述符	成功
-1	失败

【需求】

- 头文件：cvi_timer.h
- 库文件：libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.2.2 CVI_Timer_DeInit

【描述】

去初始化定时器。

【语法】

```
int32_t CVI_Timer_DeInit(int32_t grpHdl);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入

【返回值】

返回值	描述
文件描述符	成功
-1	失败

【需求】

- 头文件：cvi_timer.h
- 库文件：libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.2.3 CVI_Timer_Create

【描述】

创建定时任务。

【语法】

```
CVI_TIMER_HANDLE_T CVI_Timer_Create(int32_t grpHdl, CVI_TIMER_S *timerConf);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入
timerConf	定时器指针	输入

【返回值】

返回值	描述
非 NULL	成功
NULL	失败

【需求】

- 头文件：cvi_timer.h
- 库文件：libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.2.4 CVI_Timer_Reset

【描述】

重置定时任务。

【语法】

```
int32_t CVI_Timer_Reset(int32_t grpHdl, CVI_TIMER_HANDLE_T tmrHdle, struct timespec_↵  
↵ *timeVal, uint32_t timeLen);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入
tmrHdle	定时任务句柄	输入
timeVal	定时起始时间	输入
timeLen	定时时长，单位毫秒	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_timer.h
- 库文件：libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.2.5 CVI_Timer_SetTickValue

【描述】

设置定时任务组计时精度。

【语法】

```
int32_t CVI_Timer_SetTickValue(int32_t grpHdl, uint32_t u32TickVal_us);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入
u32TickVal_us	计时精度，单位毫秒	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_timer.h`
- 库文件: `libcvi_timer.a/libcvi_timer.so`

【注意】

无。

【举例】

无。

5.11.2.6 CVI_Timer_Destroy

【描述】

销毁定时任务。

【语法】

```
int32_t CVI_Timer_Destroy(int32_t grpHdl, CVI_TIMER_HANDLE_T tmrHdle);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入
tmrHdle	定时任务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_timer.h`
- 库文件: `libcvi_timer.a/libcvi_timer.so`

【注意】

创建定时任务之后调用。

【举例】

无。

5.11.2.7 CVI_Timer_CleanUp

【描述】

清空定时任务组中的所有定时任务。

【语法】

```
int32_t CVI_Timer_CleanUp(int32_t grpHdl);
```

【参数】

参数名称	描述	输入/输出
grpHdl	定时任务组	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_timer.h
- 库文件：libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.2.8 CVI_Timer_SetPeriodicAttr

【描述】

设置定时任务是否周期性。

【语法】

```
int32_t CVI_Timer_SetPeriodicAttr(CVI_TIMER_HANDLE_T tmrHdle, bool periodic);
```

【参数】

参数名称	描述	输入/输出
tmrHdle	定时任务句柄	输入
periodic	周期性标识	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_timer.h
- 库文件: libcvi_timer.a/libcvi_timer.so

【注意】

当 periodic 为 TRUE 时, 定时任务为周期性定时任务; 当 periodic 为 FALSE 时, 定时任务为单次定时任务。

【举例】

无。

5.11.2.9 CVI_Timer_GetPastTime

【描述】

获取定时任务计时时间。

【语法】

```
int32_t CVI_Timer_GetPastTime(CVI_TIMER_HANDLE_T tmrHdle, uint32_t *pu32Time);
```

【参数】

参数名称	描述	输入/输出
tmrHdle	定时任务句柄	输入
pu32Time	定时任务已计时时间, 单位毫秒 (ms)	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_timer.h
- 库文件: libcvi_timer.a/libcvi_timer.so

【注意】

无。

【举例】

无。

5.11.3 数据类型

相关数据类型定义如下：

- `CVI_TIMER_S`：定义定时任务结构体。
- `CVI_TIMER_HANDLE_T`：定义定时任务句柄。
- `CVI_TIMER_PROC_CALLBACK`：定义定时任务超时处理回调函数。

5.11.3.1 `CVI_TIMER_S`

【说明】

定义定时任务结构体。

【定义】

```
typedef struct cviTIMER_CONF {  
    struct timespec *now;          /**< timer create time point32_t */  
    long interval_ms;             /**< periodic timer timeout interval or one shot timer timeout interval */  
    bool periodic;                /**< periodic or ont-shot */  
    CVI_TIMER_PROC *timer_proc;   /**< time out call back */  
    void *clientData;  
} CVI_TIMER_S;
```

【成员】

成员名称	描述
now	定时任务计时起始时间
interval_ms	定时时长，单位为 ms
periodic	定时任务周期性标识
timer_proc	定时任务计时到达时间回调
clientData	定时任务私有数据

【注意事项】

无。

【相关数据类型及接口】

无。

5.11.3.2 CVI_TIMER_HANDLE_T

【说明】

定义定时任务句柄。

【定义】

```
typedef void *CVI_TIMER_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

5.11.3.3 CVI_TIMER_PROC_CALLBACK

【说明】

定义定时任务超时处理回调函数。

【定义】

```
typedef void CVI_TIMER_PROC_CALLBACK(void *client_data, struct timespec *nowP);
```

【成员】

成员名称	描述
nowP	定时任务计时起始时间
client_data	客户端数据

【注意事项】

无。

【相关数据类型及接口】

无。

5.12 UPGRADE

5.12.1 概述

upgrade 模块用于 ota 升级。

5.12.2 API 参考

该功能模块为用户提供以下 API:

- CVI_UPGRADE_Init : 初始化升级模块。
- CVI_UPGRADE_Deinit : 去初始化升级模块。
- CVI_UPGRADE_CheckPkg : 检查 ota 升级包。
- CVI_UPGRADE_DoUpgrade : 将 ota 文件写进 flash。
- CVI_UPGRADE_DoUpgradeViaSD : 将 ota 文件写进烧录文件。
- CVI_UPGRADE_RegisterEvent : 升级模块注册事件。

5.12.2.1 CVI_UPGRADE_Init

【描述】

初始化升级模块。

【语法】

```
int32_t CVI_UPGRADE_Init(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.2.2 CVI_UPGRADE_Deinit

【描述】

去初始化升级模块。

【语法】

```
int32_t CVI_UPGRADE_Deinit(void);
```

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.2.3 CVI_UPGRADE_CheckPkg

【描述】

检查升级包是否可用。

【语法】

```
int32_t CVI_UPGRADE_CheckPkg(const char *pszPkgUrl, const CVI_UPGRADE_DEV_INFO_S_
↪ *pstDevInfo, CVI_BOOL bCheckVer);
```

【参数】

参数名称	描述	输入/输出
pszPkgUrl	升级包路径指针	输入
pstDevInfo	设备信息结构体指针	输入
bCheckVer	是否检查软件版本一致	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.2.4 CVI_UPGRADE_DoUpgrade

【描述】

将 ota 文件写进 flash。

【语法】

```
int32_t CVI_UPGRADE_DoUpgrade(const char *pszPkgUrl);
```

【参数】

参数名称	描述	输入/输出
pszPkgUrl	升级包路径指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.2.5 CVI_UPGRADE_DoUpgradeViaSD

【描述】

将 ota 文件写入烧录文件。

【语法】

```
int32_t CVI_UPGRADE_DoUpgradeViaSD(const char *pszPkgUrl, const char *path);
```

【参数】

参数名称	描述	输入/输出
pszPkgUrl	升级包路径指针	输入
path	挂载路径指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.2.6 CVI_UPGRADE_RegisterEvent

【描述】

升级模块注册事件。

【语法】

```
void CVI_UPGRADE_RegisterEvent(void (*eventCb)(CVI_UPGRADE_EVENT_S *));
```

【参数】

参数名称	描述	输入/输出
eventCb	回调函数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_upgrade.h
- 库文件: libcvi_upgrade.a/libcvi_upgrade.so

【注意】

无。

【举例】

无。

5.12.3 数据类型

相关数据类型定义如下：

- CVI_UPGRADE_PKG_HEAD_S：定义升级包头结构体。
- CVI_UPGRADE_EVENT_S：定义升级模块事件结构体。
- CVI_UPGRADE_DEV_INFO_S：定义升级设备信息结构体。

5.12.3.1 CVI_UPGRADE_PKG_HEAD_S

【说明】

定义 ota 文件头信息的结构体。

【定义】

```
typedef struct cviUPGRADE_PKG_HEAD_S {
    uint32_t u32Magic;
    uint32_t u32Crc;          /* CRC number from HeadVer to end of image-data */
    uint32_t u32HeadVer;      /* Package head version: 0x00000001 */
    uint32_t u32PkgLen;       /* Package total length, including head/data */
    uint32_t reserved0;
    char szPkgModel[CVI_COMM_STR_LEN]; /* Package model, eg. cv1835_asic_wevb_0002a */
    char szPkgSoftVersion[CVI_COMM_STR_LEN]; /* Package version, eg. 1.0.0.0 */
    /*file name for partition, eg. for spinor(partition.xml), 0:fip_spl.bin, 1:yoc.bin */
    char PartitionFileName[CVI_UPGRADE_MAX_PART_CNT][32];
    char reserved1[704];
    uint32_t reserved2;
    int32_t s32PartitionCnt;
    /* Partition offset in upgrade package */
    uint32_t au32PartitionOffSet[CVI_UPGRADE_MAX_PART_CNT];
} CVI_UPGRADE_PKG_HEAD_S;
```

【成员】

成员名称	描述
u32Magic	ota 标识
u32Crc	crc 校验码
u32HeadVer	文件头版本信息
u32PkgLen	升级包大小
reserved0	预留区 0
szPkgModel	板端型号
szPkgSoftVersion	升级包版本号
PartitionFileName	分区文件名
reserved1	预留区 1
reserved2	预留区 2
s32PartitionCnt	分区个数
au32PartitionOffSet	每个分区偏移量

【注意事项】

无。

【相关数据类型及接口】

无。

5.12.3.2 CVI_UPGRADE_EVENT_S**【说明】**

定义升级模块事件结构体。

【定义】

```
typedef struct cviUPGRADE_EVENT_S {  
    uint32_t eventID;  
    void *argv;  
} CVI_UPGRADE_EVENT_S;
```

【成员】

成员名称	描述
eventID	事件 ID
argv	参数

【注意事项】

无。

【相关数据类型及接口】

无。

5.12.3.3 CVI_UPGRADE_DEV_INFO_S

【说明】

定义设备信息结构体。

【定义】

```
typedef struct cviUPGRADE_DEV_INFO_S {
    char szSoftVersion[CVI_COMM_STR_LEN]; /* Software version */
    char szModel[CVI_COMM_STR_LEN];      /* Product model */
} CVI_UPGRADE_DEV_INFO_S;
```

【成员】

成员名称	描述
szSoftVersion	软件版本
szModel	软件信息

【注意事项】

无。

【相关数据类型及接口】

无。

- mem_alloc: ION 内存分配。
- Rbuf: 又名 ringbuffer, 用于缓存外部写入的音视频数据。
- DTCF: 又名日期时间文件系统模块, 详情 DTCF 组件。
- Muxer: 文件封装器, 其中有使用第三方库 ffmpeg, 请事先了解 ffmpeg 编码相关知识。
- Record: 结合以上 4 个组件的录像组件, 用于不同的录像类型, 如普通录像, 缩时录像, 紧急录像。
- Record Service: 该组件是在 record 组件上再封装一层, 用来接受外部参数, 其专门用于对接 APP 层, 提供的是录制的视频服务。
- Audio Service: 该服务提供的是录制的音频服务。
- Moviemode 指的是 app 层的状态管理 (statemng) 的录像模式, 该模式最终调用这两个服务组件的接口。

6.2 基本概念

1. 录像: 负责将若干路码流录制成相应的录像文件, 包括从外部模块获取各路码流对应的录制文件名, 同时控制各路码流的启动, 停止以及切分等操作, 此外将录像内部的事件上报到应用层。
2. 录制码流: 一路录制码流录制首先需要启动码流包含的数据源的启动, 将包含的数据源封装到媒体文件中, 当到达数据切分点之后完成文件的切分。
3. 数据源: 一组数据源包含视频数据源, 音频数据源等。
4. 录像切分: 启动录像后, 可以按照给定的切分策略对录制码流进行切分, 例如按照时间切分, 就是指在录制过程中, 当文件时长达到指定值后, 就进行切分操作, 开始下一个文件的录制。
5. 录制类型: 从数据源的连续性角度, 录制类型包括普通录像、缩时录像两种类型:
 - 普通录像模式下, 录像启动一次数据源, 数据源以一定的帧率连续不断的输出编码数据到录像模块, 直到收到用户停止录像的命令。
 - 缩时录像模式下, 录像以一定的时间间隔周期性的请求数据源产生一帧数据, 并写入录像模块。

6.3 组件 API

讲述 Mem_alloc, Ringbuffer, Muxer, Recorder, Record Service 和 Audio Service 组件。

6.3.1 Mem_alloc

ION 内存分配, ION 内存专门用于音视频内存分配, 可以减少内存分配。

6.3.1.1 API 参考

该组件提供以下 API:

- `CVI_MEM_Allocate`: 用户分配 ION 内存, 该内存将支持 cache, 针对大核 ION
- `CVI_MEM_Free`: 用户释放 ION 内存, 针对大核 ION
- `CVI_MEM_AllocateVb`: 用户分配 ION 内存, 针对小核 ION
- `CVI_MEM_VbFree`: 用户释放 ION 内存, 针对小核 ION

6.3.1.1.1 CVI_MEM_Allocate

【描述】

用户分配 ION 内存, 该内存将支持 cache, 针对大核 ION; 与 `CVI_MEM_Free` 配套使用

【语法】

```
void *CVI_MEM_Allocate(size_t size, const char *name);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入
size	分配大小	输入

【返回值】

返回值	描述
void *	虚拟地址指针

【需求】

- 头文件: `cvi_mem.h`
- 库文件: `libcvi_mem_alloc.a/libcvi_mem_alloc.so`

【注意】

透过此 API 所取得的虚拟地址, 若有跟 HW DMA 共享。在 RISC-V 存取前, 需呼叫 `invalidate`; 而在 RISC-V 修改后, 若要让 HW DMA 读取, 则需做 `invalidate`

【举例】

无

【相关主题】

无

6.3.1.1.2 CVI_MEM_Free**【描述】**

用户释放 ION 内存，针对大核 ION；与 CVI_MEM_Allocate 配套使用

【语法】

```
void CVI_MEM_Free(void *vir_addr);
```

【参数】

参数名称	描述	输入/输出
vir_addr	虚拟地址指针	输入

【返回值】

无

【需求】

- 头文件：cvi_mem.h
- 库文件：libcvi_mem_alloc.a/libcvi_mem_alloc.so

【注意】**【举例】**

无

【相关主题】

无

6.3.1.1.3 CVI_MEM_AllocateVb**【描述】**

用户分配 ION 内存，针对小核 ION；与 CVI_MEM_VbFree 配套使用

【语法】

```
void *CVI_MEM_AllocateVb(size_t size);
```

【参数】

参数名称	描述	输入/输出
size	分配大小	输入

【返回值】

返回值	描述
void *	虚拟地址指针

【需求】

- 头文件: cvi_mem.h
- 库文件: libcvi_mem_alloc.a/libcvi_mem_alloc.so

【注意】**【举例】**

无

【相关主题】

无

6.3.1.1.4 CVI_MEM_VbFree

【描述】

用户释放 ION 内存, 针对小核 ION; 与 CVI_MEM_AllocateVb 配套使用

【语法】

```
void CVI_MEM_VbFree(void *vir_addr);
```

【参数】

参数名称	描述	输入/输出
vir_addr	虚拟地址指针	输入

【返回值】

无

【需求】

- 头文件: cvi_mem.h
- 库文件: libcvi_mem_alloc.a/libcvi_mem_alloc.so

【注意】**【举例】**

无

【相关主题】

无

6.3.2 Muxer

文件封装器，其中有使用第三方库 ffmpeg API，请事先了解 ffmpeg 编码相关知识。

6.3.2.1 API 参考

该组件提供以下 API：

- `CVI_MUXER_Create`：创建复用器
- `CVI_MUXER_Start`：启动复用器
- `CVI_MUXER_WritePacket`：写包进文件
- `CVI_MUXER_Stop`：停止复用器
- `CVI_MUXER_Destroy`：销毁复用器
- `CVI_MUXER_FlushPackets`：冲刷剩余包进文件

6.3.2.1.1 CVI_MUXER_Create

【描述】

创建复用器

【语法】

```
int32_t CVI_MUXER_Create(CVI_MUXER_ATTR_S attr, void **muxer);
```

【参数】

参数名称	描述	输入/输出
attr	复用器属性	输入
muxer	复用器	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_muxer.h
- 库文件：libcvi_muxer.a/libcvi_muxer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.1.2 CVI_MUXER_Start

【描述】

启动复用器

【语法】

```
int32_t CVI_MUXER_Start(void *muxer, const char *fname);
```

【参数】

参数名称	描述	输入/输出
fname	文件名	输入
muxer	复用器	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_muxer.h
- 库文件: libcvi_muxer.a/libcvi_muxer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.1.3 CVI_MUXER_WritePacket

【描述】

写包进文件

【语法】

```
int32_t CVI_MUXER_WritePacket(void *muxer, CVI_MUXER_FRAME_INFO_S *packet);
```

【参数】

参数名称	描述	输入/输出
packet	包数据	输入
muxer	复用器	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_muxer.h
- 库文件: libcvi_muxer.a/libcvi_muxer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.1.4 CVI_MUXER_Stop

【描述】

停止复用器

【语法】

```
void CVI_MUXER_Stop(void *muxer);
```

【参数】

参数名称	描述	输入/输出
muxer	复用器	输入

【返回值】

无

【需求】

- 头文件: `cvi_muxer.h`
- 库文件: `libcvi_muxer.a/libcvi_muxer.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.1.5 CVI_MUXER_Destroy

【描述】

销毁复用器

【语法】

```
void CVI_MUXER_Destroy(void *muxer);
```

【参数】

参数名称	描述	输入/输出
<code>muxer</code>	复用器	输入

【返回值】

无

【需求】

- 头文件: `cvi_muxer.h`
- 库文件: `libcvi_muxer.a/libcvi_muxer.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.1.6 CVI_MUXER_FlushPackets

【描述】

冲刷剩余包进文件

【语法】

```
void CVI_MUXER_FlushPackets(void *muxer, int32_t flag);
```

【参数】

参数名称	描述	输入/输出
flag	标志	输入
muxer	复用器	输入

【返回值】

无

【需求】

- 头文件: cvi_muxer.h
- 库文件: libcvi_muxer.a/libcvi_muxer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.2.2 数据类型

相关数据类型定义如下:

- CVI_MUXER_TRACK_VIDEO_CODEC_E : 定义视频编码类型
- CVI_MUXER_TRACK_AUDIO_CODEC_E : 定义音频编码类型
- CVI_MUXER_CODEC_VIDEO_S : 定义视频编码信息
- CVI_MUXER_CODEC_AUDIO_S : 定义音频编码信息
- CVI_MUXER_CODEC_SUBTITLE_S : 定义字幕编码信息
- CVI_MUXER_CODEC_THUMBNAIL_S : 定义缩略图编码信息
- CVI_MUXER_EVENT_E : 定义复用器事件
- CVI_MUXER_FRAME_TYPE_E : 复用器帧类型
- CVI_MUXER_FRAME_INFO_S : 定义帧信息

- CVI_MUXER_EVENT_CALLBACK：复用器回调函数
- CVI_MUXER_ATTR_S：复用器属性

6.3.2.2.1 CVI_MUXER_TRACK_VIDEO_CODEC_E

【说明】

定义视频编码类型。

【定义】

```
typedef enum cviTrack_VideoCodec_E
{
    CVI_MUXER_TRACK_VIDEO_CODEC_H264 = 96,
    CVI_MUXER_TRACK_VIDEO_CODEC_H265 = 98,
    CVI_MUXER_TRACK_VIDEO_CODEC_MJPEG = 102,
    CVI_MUXER_TRACK_VIDEO_CODEC_BUTT
} CVI_MUXER_TRACK_VIDEO_CODEC_E;
```

【成员】

成员名称	描述
CVI_MUXER_TRACK_VIDEO_CODEC_H264	H264 编码
CVI_MUXER_TRACK_VIDEO_CODEC_H265	H265 编码
CVI_MUXER_TRACK_VIDEO_CODEC_MJPEG	MJPEG 编码

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.2 CVI_MUXER_TRACK_AUDIO_CODEC_E

【说明】

定义音频编码类型。

【定义】

```
typedef enum cviTrack_AudioCodec_E
{
    CVI_MUXER_TRACK_AUDIO_CODEC_G711Mu = 0, /**< G.711 Mu */
    CVI_MUXER_TRACK_AUDIO_CODEC_G711A = 8, /**< G.711 A */
    CVI_MUXER_TRACK_AUDIO_CODEC_G726 = 97, /**< G.726 */
    CVI_MUXER_TRACK_AUDIO_CODEC_AMR = 101, /**< AMR encoder format */
    CVI_MUXER_TRACK_AUDIO_CODEC_ADPCM = 104, /**< ADPCM */
    CVI_MUXER_TRACK_AUDIO_CODEC_AAC = 105,
    CVI_MUXER_TRACK_AUDIO_CODEC_WAV = 108, /**< WAV encoder */
    CVI_MUXER_TRACK_AUDIO_CODEC_MP3 = 109,
```

(下页继续)

(续上页)

```
CVI_MUXER_TRACK_AUDIO_CODEC_BUTT
} CVI_MUXER_TRACK_AUDIO_CODEC_E;
```

【成员】

成员名称	描述
CVI_MUXER_TRACK_AUDIO_CODEC_G711MULAW	G711MULAW-law 编码
CVI_MUXER_TRACK_AUDIO_CODEC_G711A	G711A-law 编码
CVI_MUXER_TRACK_AUDIO_CODEC_G726	G726 编码
CVI_MUXER_TRACK_AUDIO_CODEC_AMR	AMR 编码
CVI_MUXER_TRACK_AUDIO_CODEC_ADPCM	ADPCM 编码
CVI_MUXER_TRACK_AUDIO_CODEC_AAC	AAC 编码
CVI_MUXER_TRACK_AUDIO_CODEC_VORBIS	VORBIS 编码
CVI_MUXER_TRACK_AUDIO_CODEC_MP3	MP3 编码

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.3 CVI_MUXER_CODEC_VIDEO_S

【说明】

定义视频编码类型。

【定义】

```
typedef struct cviCODEC_VIDEO_T{
    int32_t en;
    float framerate;
    CVI_MUXER_TRACK_VIDEO_CODEC_E codec;
    uint32_t w;
    uint32_t h;
}CVI_MUXER_CODEC_VIDEO_S;
```

【成员】

成员名称	描述
en	使能
framerate	帧率
codec	视频编码类型
w	视频宽度
h	视频高度

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.4 CVI_MUXER_CODEC_AUDIO_S**【说明】**

定义视频编码类型。

【定义】

```
typedef struct cviCODEC_AUDIO_T{  
    int32_t en;  
    CVI_MUXER_TRACK_AUDIO_CODEC_E codec;  
    uint32_t samplerate;  
    uint32_t chns;  
    float framerate;  
}CVI_MUXER_CODEC_AUDIO_S;
```

【成员】

成员名称	描述
en	使能
codec	音频编码类型
samplerate	采样率
chns	声道数
framerate	帧率

【注意事项】

AAC：帧大小 1024 个 sample，建设采样率为 44100Hz，那么帧率 = $44100/1024 = 43.07$ f/s

【相关数据类型及接口】

无。

6.3.2.2.5 CVI_MUXER_CODEC_SUBTITLE_S**【说明】**

定义字幕编码信息

【定义】

```
typedef struct cviCODEC_SUBTITLE_T{  
    int32_t en;  
    float framerate;  
    uint32_t timebase;  
}CVI_MUXER_CODEC_SUBTITLE_S;
```

【成员】

成员名称	描述
en	使能
framerate	帧率
timebase	时间基

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.6 CVI_MUXER_CODEC_THUMBNAIL_S

【说明】

定义缩略图编码信息

【定义】

```
typedef struct cviCODEC_THUMBNAIL_T {  
    int32_t en;  
    int32_t res;  
} CVI_MUXER_CODEC_THUMBNAIL_S;
```

【成员】

成员名称	描述
en	使能
res	保留

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.7 CVI_MUXER_EVENT_E

【说明】

定义复用器事件

【定义】

```
typedef enum CVI_MUXER_EVENT_E {  
    CVI_MUXER_OPEN_FILE_FAILED,  
    CVI_MUXER_SEND_FRAME_FAILED,  
    CVI_MUXER_SEND_FRAME_TIMEOUT,  
};
```

(下页继续)

(续上页)

```

    CVI_MUXER_PTS_JUMP,
    CVI_MUXER_STOP,
    CVI_MUXER_EVENT_BUTT
} CVI_MUXER_EVENT_E;

```

【成员】

成员名称	描述
CVI_MUXER_OPEN_FILE_FAILED	打开文件失败
CVI_MUXER_STOP	停止复用器
CVI_MUXER_SEND_FRAME_FAILED	帧写入失败
CVI_MUXER_SEND_FRAME_TIMEOUT	帧写入超时
CVI_MUXER_PTS_JUMP	pts 帧跳跃

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.8 CVI_MUXER_FRAME_TYPE_E**【说明】**

定义复用器帧类型

【定义】

```

typedef enum CVI_MUXER_FRAME_TYPE_E {
    CVI_MUXER_FRAME_TYPE_VIDEO,
    CVI_MUXER_FRAME_TYPE_AUDIO,
    CVI_MUXER_FRAME_TYPE_SUBTITLE,
    CVI_MUXER_FRAME_TYPE_THUMBNAI,
    CVI_MUXER_FRAME_TYPE_BUTT
} CVI_MUXER_FRAME_TYPE_E;

```

【成员】

成员名称	描述
CVI_MUXER_FRAME_TYPE_VIDEO	视频帧
CVI_MUXER_FRAME_TYPE_AUDIO	音频帧
CVI_MUXER_FRAME_TYPE_SUBTITLE	字幕帧
CVI_MUXER_FRAME_TYPE_THUMBNAI	缩略图帧

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.9 CVI_MUXER_FRAME_INFO_S

【说明】

定义帧信息

【定义】

```
typedef struct cviFRAME_INFO_T{
    uint32_t hmagic;
    CVI_MUXER_FRAME_TYPE_E type;
    int32_t isKey;
    int32_t gopInx;
    int64_t pts;
    uint64_t vpts;
    int32_t dataLen;
    int32_t extraLen;
    int32_t totalSize;
    uint32_t tmagic;
}CVI_MUXER_FRAME_INFO_S;
```

【成员】

成员名称	描述
hmagic	开始标志, 魔数 0x5a5a5a5a, 用于完整性校验
type	帧类型
isKey	是否是关键帧
gopInx	帧序号, I 帧为 0, P 帧累加 1
pts	pts, 从 0 开始累加 1
vpts	使用 CPU 时间计算的微秒数
dataLen	音视频、字幕帧长度
extraLen	缩略图长度, 无缩略图时为 0
totalSize	本帧总长度, 包含 frame info 长度
tmagic	结束标志, 魔数 0x5a5a5a5a, 用于完整性校验

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.10 CVI_MUXER_EVENT_CALLBACK

【说明】

定义复用器事件回调函数

【定义】

```
typedef int32_t (*CVI_MUXER_EVENT_CALLBACK)(CVI_MUXER_EVENT_E event_type,
↪const char *filename, void *p, void *extend);
```

【成员】

成员名称	描述
event_type	事件类型
filename	文件名
p	参数
extend	扩展

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.2.2.11 CVI_MUXER_ATTR_S**【说明】**

定义复用器属性

【定义】

```
typedef struct cviMUXER_ATTR_T{
    CVI_MUXER_CODEC_VIDEO_S stvideocodec;
    CVI_MUXER_CODEC_AUDIO_S staudiocodec;
    CVI_MUXER_CODEC_SUBTITLE_S stsubtitlecodec;
    CVI_MUXER_CODEC_THUMBNAIL_S stthumbnailcodec;
    char *devmod;
    int32_t alignflag;
    int32_t presize;
    CVI_MUXER_EVENT_CALLBACK pfncallback;
    void *pfnparam;
}CVI_MUXER_ATTR_S;
```

【成员】

成员名称	描述
stvideocodec	视频编码信息
staudiocodec	音频编码信息
stsubtitlecodec	字幕编码信息
stthumbnailcodec	缩略图编码信息
devmod	设备模型
alignflag	对齐标志
presize	预分配大小
pfncallback	复用器事件回调函数
pfnparam	回调函数参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3 Recorder

结合以上 4 个组件的录像组件，用于不同的录像类型，如普通录像，缩时录像，紧急录像。

6.3.3.1 API 参考

该组件提供以下 API：

- `CVI_RECORDER_SendFrame`：将编码得到的帧写进 ringbuffer
- `CVI_RECORDER_Start_NormalRec`：开启正常录像
- `CVI_RECORDER_Stop_NormalRec`：停止正常录像
- `CVI_RECORDER_Start_EventRec`：开启事件录像，比如紧急录像
- `CVI_RECORDER_ForceStop_EventRec`：手动停止事件录像
- `CVI_RECORDER_Stop_EventRecPost`：停止事件录像
- `CVI_RECORDER_Start_LapseRec`：开启缩时录像
- `CVI_RECORDER_Stop_LapseRec`：停止缩时录像
- `CVI_RECORDER_Destroy`：销毁录像组件
- `CVI_RECORDER_Create`：创建录像组件
- `CVI_RECORDER_Split`：录像文件切分
- `CVI_RECORDER_Timelapse_Is_SendVenc`：确认符合缩时录像的帧时间
- `CVI_RECORDER_GetUs`：获取现在的时间

6.3.3.1.1 CVI_RECORDER_SendFrame

【描述】

将编码得到的帧写进 ringbuffer

【语法】

```
int32_t CVI_RECORDER_SendFrame(void *recorder, CVI_RECORDER_FRAME_STREAM_S_
↪ *frame);
```

【参数】

参数名称	描述	输入/输出
frame	帧流，可由多个帧组成	输入
recorder	录像上下文	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.2 CVI_RECORDER_Start_NormalRec

【描述】

开启正常录像

【语法】

```
int32_t CVI_RECORDER_Start_NormalRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.3 CVI_RECORDER_Stop_NormalRec

【描述】

停止正常录像

【语法】

```
int32_t CVI_RECORDER_Stop_NormalRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.4 CVI_RECORDER_Start_EventRec

【描述】

开启事件录像，比如紧急录像

【语法】

```
int32_t CVI_RECORDER_Start_EventRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_recorder.h/filesync.h
- 库文件：libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.5 CVI_RECORDER_Stop_EventRecPost

【描述】

停止事件录像

【语法】

```
int32_t CVI_RECORDER_Stop_EventRecPost(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_recorder.h/filesync.h

- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.6 CVI_RECORDER_ForceStop_EventRec

【描述】

手动停止事件录像

【语法】

```
int32_t CVI_RECORDER_ForceStop_EventRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.7 CVI_RECORDER_Start_LapseRec

【描述】

开启缩时录像

【语法】

```
int32_t CVI_RECORDER_Start_LapseRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.8 CVI_RECORDER_Stop_LapseRec

【描述】

停止缩时录像

【语法】

```
int32_t CVI_RECORDER_Stop_LapseRec(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.9 CVI_RECORDER_Destroy**【描述】**

销毁录像组件

【语法】

```
void CVI_RECORDER_Destroy(void **recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

无

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.10 CVI_RECORDER_Create

【描述】

创建录像组件

【语法】

```
int32_t CVI_RECORDER_Create(void **recorder, CVI_RECORDER_ATTR_S *attr);
```

【参数】

参数名称	描述	输入/输出
attr	录像属性	输入
recorder	录像上下文	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_recorder.h/filesync.h
- 库文件: libcvi_recorder.a/libcvi_recorder.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.11 CVI_RECORDER_Split

【描述】

录像文件切分

【语法】

```
int32_t CVI_RECORDER_Split(void *recorder);
```

【参数】

参数名称	描述	输入/输出
recorder	录像上下文	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_recorder.h/filesync.h`
- 库文件: `libcvi_recorder.a/libcvi_recorder.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.12 CVI_RECORDER_Timelapse_Is_SendVenc

【描述】

确认符合缩时录像的帧时间

【语法】

```
int32_t CVI_RECORDER_Timelapse_Is_SendVenc(void *recorder);
```

【参数】

参数名称	描述	输入/输出
frame	帧流, 可由多个帧组成	输入
recorder	录像上下文	输出

【返回值】

返回值	描述
1	成功
0	失败

【需求】

- 头文件: `cvi_recorder.h/filesync.h`
- 库文件: `libcvi_recorder.a/libcvi_recorder.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.1.13 CVI_RECORDER_GetUs

【描述】

获取现在的时间

【语法】

```
uint64_t CVI_RECORDER_GetUs(void);
```

【参数】

无

【返回值】

返回值	描述
非 0	微秒数

【需求】

- 头文件: `cvi_recorder.h/filesync.h`
- 库文件: `libcvi_recorder.a/libcvi_recorder.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.3.2 数据类型

相关数据类型定义如下:

- `CVI_RECORDER_EVENT_E` : 定义录像事件
- `CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT_S` : 定义写帧超时结构体
- `CVI_RECORDER_GET_FILENAME_CALLBACK` : 定义获取文件名回调函数
- `CVI_RECORDER_EVENT_CALLBACK` : 定义录像事件回调函数
- `CVI_RECORDER_GET_SUBTITLE_CALLBACK` : 定义获取字幕回调函数
- `CVI_RECORDER_GET_MEM_BUFFER_STOP_CALLBACK` : 定义停止获取缓存回调函数
- `CVI_RECORDER_REQUEST_IDR_CALLBACK` : 定义请求 IDR 帧回调函数
- `CVI_RECORDER_STOP_CALLBACK` : 定义录像停止回调函数
- `CVI_RECORDER_GET_DIR_TYPE_CALLBACK` : 定义获取文件夹类型回调函数
- `CVI_RECORDER_TYPE_INDEX_E` : 定义录像类型索引
- `CVI_RECORDER_CALLBACK_TYPE_E` : 定义回调类型
- `CVI_RECORDER_CB_HANDLES_S` : 定义回调句柄
- `CVI_RECORDER_RBUF_TYPE_E` : 定义录像 ringbuffer 类型
- `CVI_RECORDER_RBUF_ATTR_S` : 定义录像 ringbuffer 属性
- `CVI_RECORDER_TRACK_SOURCE_TYPE_E` : 定义媒体文件轨类型
- `CVI_RECORDER_TYPE_E` : 定义录像类型
- `CVI_RECORDER_SPLIT_TYPE_E` : 定义切分类型
- `CVI_RECORDER_SPLIT_ATTR_S` : 定义切分属性
- `CVI_RECORDER_NORMAL_ATTR_S` : 定义普通录像属性
- `CVI_RECORDER_LAPSE_ATTR_S` : 定义缩时录像属性
- `CVI_RECORDER_TRACK_VideoSourceInfo_S` : 定义视频数据源信息
- `CVI_RECORDER_TRACK_AudioSourceInfo_S` : 定义音频数据源信息
- `CVI_RECORDER_TRACK_PrivateSourceInfo_S` : 定义私有数据源信息
- `CVI_RECORDER_TRACK_SOURCE_S` : 定义数据源信息
- `CVI_RECORDER_STREAM_ATTR_S` : 定义录像码流属性
- `CVI_RECORDER_ATTR_S` : 定义录像组件属性
- `CVI_RECORDER_FRAME_STREAM_S` : 定义编码码流
- `CVI_RECORDER_TRACK_MAX_CNT` : 定义最大轨道数量
- `CVI_FRAME_STREAM_SEGMENT_MAX_NUM` : 定义码流所含最大帧数

6.3.3.2.1 CVI_FRAME_STREAM_SEGMENT_MAX_NUM

【说明】

定义码流所含最大帧数

【定义】

```
#define CVI_FRAME_STREAM_SEGMENT_MAX_NUM (8)
```

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.2 CVI_RECORDER_TRACK_MAX_CNT

【说明】

定义最大轨道数量

【定义】

```
#define CVI_RECORDER_TRACK_MAX_CNT (CVI_RECORDER_TRACK_SOURCE_TYPE_↪BUTT)
```

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.3 CVI_RECORDER_EVENT_E

【说明】

定义录像事件

【定义】

```
typedef enum CVI_RECORDER_EVENT_E
{
    CVI_RECORDER_EVENT_START,
    CVI_RECORDER_EVENT_STOP,
    CVI_RECORDER_EVENT_STOP_FAILED,
    CVI_RECORDER_EVENT_SPLIT,
    CVI_RECORDER_EVENT_WRITE_FRAME_DROP,
    CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT,
    CVI_RECORDER_EVENT_WRITE_FRAME_FAILED,
    CVI_RECORDER_EVENT_OPEN_FILE_FAILED,
}
```

(下页继续)

(续上页)

```

CVI_RECORDER_EVENT_CLOSE_FILE_FAILED,
CVI_RECORDER_EVENT_SHORT_FILE,
CVI_RECORDER_EVENT_PIV_START,
CVI_RECORDER_EVENT_PIV_END,
CVI_RECORDER_EVENT_SYNC_DONE,
CVI_RECORDER_EVENT_SPLIT_START,
CVI_RECORDER_EVENT_START_EMR,
CVI_RECORDER_EVENT_END_EMR,
CVI_RECORDER_EVENT_FRAME_DROP,
CVI_RECORDER_EVENT_BUTT
} CVI_RECORDER_EVENT_E;

```

【成员】

成员名称	描述
CVI_RECORDER_EVENT_START	录像开启
CVI_RECORDER_EVENT_STOP	录像停止
CVI_RECORDER_EVENT_STOP_FAILED	录像停止失败
CVI_RECORDER_EVENT_SPLIT	录像文件切分
CVI_RECORDER_EVENT_WRITE_FRAME_DROP	丢写入的帧
CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT	写帧超时
CVI_RECORDER_EVENT_WRITE_FRAME_FAILED	写帧失败
CVI_RECORDER_EVENT_OPEN_FILE_FAILED	打开文件失败
CVI_RECORDER_EVENT_CLOSE_FILE_FAILED	关闭文件失败
CVI_RECORDER_EVENT_SHORT_FILE	短文件
CVI_RECORDER_EVENT_PIV_START	抓拍开始
CVI_RECORDER_EVENT_PIV_END	抓拍结束
CVI_RECORDER_EVENT_SYNC_DONE	文件同步
CVI_RECORDER_EVENT_SPLIT_START	录像切分开始
CVI_RECORDER_EVENT_START_EMR	开始紧急录像
CVI_RECORDER_EVENT_END_EMR	紧急录像结束
CVI_RECORDER_EVENT_FRAME_DROP	丢帧

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.4 CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT_S**【说明】**

定义写帧超时结构体

【定义】

```
typedef struct _cvi_RECORDER_EVENT_WRITE_FRAME_TIMEOUT_S{  
    int32_t timeout_ms;  
    void *param;  
} CVI_RECORDER_EVENT_WRITE_FRAME_TIMEOUT_S;
```

【成员】

成员名称	描述
timeout_ms	超时时间
param	参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.5 CVI_RECORDER_GET_FILENAME_CALLBACK

【说明】

定义获取文件名回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_GET_FILENAME_CALLBACK)(void *p, char *filename,  
→int32_t filename_len);
```

【成员】

成员名称	描述
p	参数
filename	文件名
filename_len	文件名长度

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.6 CVI_RECORDER_EVENT_CALLBACK

【说明】

定义录像事件回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_EVENT_CALLBACK)(CVI_RECORDER_EVENT_E event_  
→type, const char *filename, void *p);
```

【成员】

成员名称	描述
p	参数
filename	文件名
event_type	事件类型

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.7 CVI_RECORDER_GET_SUBTITLE_CALLBACK

【说明】

定义获取字幕回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_GET_SUBTITLE_CALLBACK)(void *p, char *str, int32_t_  
→str_len);
```

【成员】

成员名称	描述
p	参数
str	字符串
str_len	字符串长度

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.8 CVI_RECORDER_GET_MEM_BUFFER_STOP_CALLBACK

【说明】

定义停止获取缓存回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_GET_MEM_BUFFER_STOP_CALLBACK)(void *p);
```

【成员】

成员名称	描述
p	参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.9 CVI_RECORDER_REQUEST_IDR_CALLBACK

【说明】

定义请求 IDR 帧回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_REQUEST_IDR_CALLBACK)(void *p);
```

【成员】

成员名称	描述
p	参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.10 CVI_RECORDER_STOP_CALLBACK

【说明】

定义录像停止回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_STOP_CALLBACK)(void *p)
```

【成员】

成员名称	描述
p	参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.11 CVI_RECORDER_GET_DIR_TYPE_CALLBACK

【说明】

定义获取文件夹类型回调函数

【定义】

```
typedef int32_t (*CVI_RECORDER_GET_DIR_TYPE_CALLBACK)(int32_t id, int32_t base);
```

【成员】

成员名称	描述
id	录像 id
base	文件夹类型

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.12 CVI_RECORDER_TYPE_INDEX_E

【说明】

定义录像类型索引

【定义】

```
typedef enum CVI_RECORDER_TYPE_INDEX_E{  
    CVI_RECORDER_TYPE_NORMAL_INDEX = 0,  
    CVI_RECORDER_TYPE_LAPSE_INDEX = 0,  
    CVI_RECORDER_TYPE_EVENT_INDEX = 1,  
    CVI_RECORDER_TYPE_BUTT_INDEX = 2  
}CVI_RECORDER_TYPE_INDEX_E;
```

【成员】

成员名称	描述
CVI_RECORDER_TYPE_NORMAL_INDEX	普通录像索引
CVI_RECORDER_TYPE_LAPSE_INDEX	缩时录像索引
CVI_RECORDER_TYPE_EVENT_INDEX	事件录像索引

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.13 CVI_RECORDER_CALLBACK_TYPE_E

【说明】

定义回调类型

【定义】

```
typedef enum CVI_CALLBACK_TYPE{  
    CVI_RECORDER_CALLBACK_TYPE_NORMAL = 0,  
    CVI_RECORDER_CALLBACK_TYPE_LAPSE,  
    CVI_RECORDER_CALLBACK_TYPE_EVENT,  
    CVI_RECORDER_CALLBACK_TYPE_BUTT  
}CVI_RECORDER_CALLBACK_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORDER_CALLBACK_TYPE_NORMAL	普通录像回调
CVI_RECORDER_CALLBACK_TYPE_LAPSE	缩时录像回调
CVI_RECORDER_CALLBACK_TYPE_EVENT	事件录像回调

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.14 CVI_RECORDER_CB_HANDLES_S

【说明】

定义回调句柄

【定义】

```
typedef struct CVI_CALLBACK_HANDLES_S{
    CVI_RECORDER_GET_SUBTITLE_CALLBACK pfn_get_subtitle_cb;
    void *pfn_get_subtitle_cb_param;
    CVI_RECORDER_GET_FILENAME_CALLBACK pfn_get_filename;
    void *pfn_get_filename_param[CVI_RECORDER_CALLBACK_TYPE_BUTT];
    CVI_RECORDER_REQUEST_IDR_CALLBACK pfn_request_idr;
    void *pfn_request_idr_param;
    CVI_RECORDER_EVENT_CALLBACK pfn_event_cb[CVI_RECORDER_TYPE_BUTT_
→INDEX]; /*normal && lapse share*/
    void *pfn_event_cb_param;
    CVI_RECORDER_GET_MEM_BUFFER_STOP_CALLBACK pfn_mem_buffer_stop_cb;
    void *pfn_mem_buffer_stop_cb_param;
    CVI_RECORDER_STOP_CALLBACK pfn_rec_stop_cb;
    void *pfn_rec_stop_cb_param;
}CVI_RECORDER_CB_HANDLES_S;
```

【成员】

成员名称	描述
pfn_get_subtitle_cb	获取字幕回调函数指针
pfn_get_subtitle_cb_param	字幕回调函数参数
pfn_get_filename	获取文件名回调函数指针
pfn_get_filename_param	获取文件名回调函数参数
pfn_request_idr	请求 IDR 帧回调函数指针
pfn_request_idr_param	请求 IDR 帧回调函数参数
pfn_event_cb	录像事件回调函数指针
pfn_event_cb_param	录像事件回调函数参数
pfn_mem_buffer_stop_cb	停止获取缓存回调函数指针
pfn_mem_buffer_stop_cb_param	停止获取缓存回调函数参数
pfn_rec_stop_cb	录像停止回调函数指针
pfn_rec_stop_cb_param	录像停止回调函数参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.15 CVI_RECORDER_RBUF_TYPE_E

【说明】

定义录像 ringbuffer 类型

【定义】

```
typedef enum CVI_RECORDER_RBUF_TYPE_E{  
    CVI_RECORDER_RBUF_VIDEO = 0,  
    CVI_RECORDER_RBUF_AUDIO,  
    CVI_RECORDER_RBUF_SUBTITLE,  
    CVI_RECORDER_RBUF_BUTT  
}CVI_RECORDER_RBUF_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORDER_RBUF_VIDEO	视频类型
CVI_RECORDER_RBUF_AUDIO	音频类型
CVI_RECORDER_RBUF_SUBTITLE	字幕类型

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.16 CVI_RECORDER_RBUF_ATTR_S

【说明】

定义录像 ringbuffer 属性

【定义】

```
typedef struct CVI_RECORDER_RBUF_ATTR_S{  
    uint32_t size;  
    const char *name;  
}CVI_RECORDER_RBUF_ATTR_S;
```

【成员】

成员名称	描述
size	大小
name	名字

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.17 CVI_RECORDER_TRACK_SOURCE_TYPE_E

【说明】

定义媒体文件轨类型

【定义】

```
typedef enum cviTrack_SourceType_E {  
    CVI_RECORDER_TRACK_SOURCE_TYPE_VIDEO = 0,  
    CVI_RECORDER_TRACK_SOURCE_TYPE_AUDIO,  
    CVI_RECORDER_TRACK_SOURCE_TYPE_PRIV,  
    CVI_RECORDER_TRACK_SOURCE_TYPE_BUTT  
} CVI_RECORDER_TRACK_SOURCE_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORDER_TRACK_SOURCE_TYPE_VIDEO	视频轨
CVI_RECORDER_TRACK_SOURCE_TYPE_AUDIO	音频轨
CVI_RECORDER_TRACK_SOURCE_TYPE_PRIV	私有轨

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.18 CVI_RECORDER_TYPE_E

【说明】

定义录像类型

【定义】

```
typedef enum cviREC_TYPE_E {  
    CVI_RECORDER_TYPE_NORMAL = 0, /* normal record */  
    CVI_RECORDER_TYPE_LAPSE, /* time lapse record, record a frame by an fixed time_  
→interval */  
    CVI_RECORDER_TYPE_BUTT  
} CVI_RECORDER_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORDER_TYPE_NORMAL	普通录像
CVI_RECORDER_TYPE_LAPSE	缩时录像

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.19 CVI_RECORDER_SPLIT_TYPE_E**【说明】**

定义切分类型

【定义】

```
typedef enum cviREC_SPLIT_TYPE_E {  
    CVI_RECORDER_SPLIT_TYPE_NONE = 0, /* means split is disabled */  
    CVI_RECORDER_SPLIT_TYPE_TIME,    /* record split when time reaches */  
    CVI_RECORDER_SPLIT_TYPE_BUTT  
} CVI_RECORDER_SPLIT_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORDER_SPLIT_TYPE_NONE	禁止切分
CVI_RECORDER_SPLIT_TYPE_TIME	按照给定时间切分

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.20 CVI_RECORDER_SPLIT_ATTR_S**【说明】**

定义切分属性

【定义】

```
typedef struct cviREC_SPLIT_ATTR_S {  
    CVI_RECORDER_SPLIT_TYPE_E enSplitType; /* split type */  
    uint64_t u64SplitTimeLenMSec;          /* split time, unit in msecond(ms) */  
} CVI_RECORDER_SPLIT_ATTR_S;
```

【成员】

成员名称	描述
enSplitType	切分类型
u64SplitTimeLenMSec	切分时间

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.21 CVI_RECORDER_NORMAL_ATTR_S**【说明】**

定义普通录像属性

【定义】

```
typedef struct cviREC_NORMAL_ATTR_S {  
    uint32_t u32Rsv; /* reserve */  
} CVI_RECORDER_NORMAL_ATTR_S;
```

【成员】

成员名称	描述
u32Rsv	保留

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.22 CVI_RECORDER_LAPSE_ATTR_S**【说明】**

定义缩时录像属性

【定义】

```
typedef struct cviREC_LAPSE_ATTR_S {  
    uint32_t u32IntervalMs; /* lapse record time interval, unit in millisecond(ms) */  
    float fFramerate;  
} CVI_RECORDER_LAPSE_ATTR_S;
```

【成员】

成员名称	描述
u32IntervalMs	缩时间隔
fFramerate	帧率

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.23 CVI_RECORDER_TRACK_VideoSourceInfo_S

【说明】

定义视频数据源信息

【定义】

```
typedef struct cviTrack_VideoSourceInfo_S {
    CVI_Track_VideoCodec_E enCodecType;
    uint32_t u32Width;
    uint32_t u32Height;
    uint32_t u32BitRate;
    float fFrameRate;
    uint32_t u32Gop;
    float fSpeed;
} CVI_RECORDER_TRACK_VideoSourceInfo_S;
```

【成员】

成员名称	描述
enCodecType	视频编码类型
fFramerate	目的帧率
u32Width	视频宽度
u32Height	视频高度
u32BitRate	码率
u32Gop	GOP 值
fSpeed	播放速度

【注意事项】

fSpeed 决定了录制文件以几倍速播放，对于普通录像来说，fSpeed 等于目的帧率/原始帧率；对于缩时录像来说 fSpeed 等于目的帧率 * u32IntervalMs （缩时录像帧间隔）/1000

【相关数据类型及接口】

无。

6.3.3.2.24 CVI_RECORDER_TRACK_AudioSourceInfo_S

【说明】

定义音频数据源信息

【定义】

```
typedef struct cviTrack_AudioSourceInfo_S {  
    CVI_Track_AudioCodec_E enCodecType;  
    uint32_t u32ChnCnt;  
    uint32_t u32SampleRate;  
    uint32_t u32AvgBytesPerSec;  
    uint32_t u32SamplesPerFrame;  
    unsigned short u16SampleBitWidth;  
    float fFramerate;  
} CVI_RECORDER_TRACK_AudioSourceInfo_S;
```

【成员】

成员名称	描述
enCodecType	音频编码类型
fFramerate	目的帧率
u32ChnCnt	声道数
u32SampleRate	采样率
u32AvgBytesPerSec	平均每秒 BYTE 数
u32SamplesPerFrame	平均每帧采样点
u16SampleBitWidth	采样位宽

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.25 CVI_RECORDER_TRACK_PrivateSourceInfo_S

【说明】

定义私有数据源信息

【定义】

```
typedef struct cviTrack_PrivateSourceInfo_S  
{  
    uint32_t u32PrivateData;  
    uint32_t u32FrameRate;  
    uint32_t u32BytesPerSec;  
    int32_t bStrictSync;  
} CVI_RECORDER_TRACK_PrivateSourceInfo_S;
```


【成员】

成员名称	描述
u32PrivateData	私有数据，预留
fFramerate	私有帧率
u32BytesPerSec	私有数据每秒字节数
bStrictSync	私有数据帧是否严格同步视频帧标识

【注意事项】

当私有数据的帧率大于或等于视频帧率，且私有数据帧的时间戳与视频帧严格同步时，建议 bStrictSync 设置为 CVI_TRUE，除此之外建议设置为 CVI_FALSE

【相关数据类型及接口】

无。

6.3.3.2.26 CVI_RECORDER_TRACK_SOURCE_S**【说明】**

定义数据源信息

【定义】

```
typedef struct cviTrack_Source_S
{
    CVI_RECORDER_TRACK_SOURCE_TYPE_E enTrackType;
    int32_t enable;
    union
    {
        CVI_RECORDER_TRACK_VideoSourceInfo_S stVideoInfo;
        CVI_RECORDER_TRACK_AudioSourceInfo_S stAudioInfo;
        CVI_RECORDER_TRACK_PrivateSourceInfo_S stPrivInfo;
    } unTrackSourceAttr;
}CVI_RECORDER_TRACK_SOURCE_S;
```

【成员】

成员名称	描述
enTrackType	数据源类型
enable	使能
unTrackSourceAttr	数据源属性枚举

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.27 CVI_RECORDER_STREAM_ATTR_S

【说明】

定义录像码流属性

【定义】

```
typedef struct cviREC_STREAM_ATTR_S {
    uint32_t u32TrackCnt; /* track cnt */
    CVI_RECORDER_TRACK_SOURCE_S aHTrackSrcHandle[CVI_RECORDER_TRACK_MAX_
    CNT]; /* array of track source cnt */
} CVI_RECORDER_STREAM_ATTR_S;
```

【成员】

成员名称	描述
u32TrackCnt	数据源数
aHTrackSrcHandle	数据源句柄数组

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.28 CVI_RECORDER_ATTR_S

【说明】

定义录像组件属性

【定义】

```
typedef struct CVI_RECORDER_ATTR_S {
    CVI_RECORDER_STREAM_ATTR_S astStreamAttr;
    CVI_RECORDER_CB_HANDLES_S fncallback;
    CVI_RECORDER_TYPE_E enRecType; /* record type */
    union {
        CVI_RECORDER_NORMAL_ATTR_S stNormalRecAttr; /* normal record attribute */
        CVI_RECORDER_LAPSE_ATTR_S stLapseRecAttr; /* lapse record attribute */
    } unRecAttr;
    CVI_RECORDER_RBUF_ATTR_S stRbufAttr[CVI_RECORDER_RBUF_BUTT];
    CVI_RECORDER_SPLIT_ATTR_S stSplitAttr; /* record split attribute */
    int32_t enable_subtitle;
    int32_t enable_file_alignment;
    int32_t enable_emrfile_from_normfile;
    float subtitle_framerate;
    uint32_t u32PreRecTimeSec; /* pre record time */
    uint32_t u32PostRecTimeSec; /* post record time */
    int32_t s32MemRecPreSec;
    char *device_model;
```

(下页继续)

(续上页)

```

int32_t prealloc_size;
float short_file_ms;
int32_t id;
} CVI_RECORDER_ATTR_S;

```

【成员】

成员名称	描述
astStreamAttr	录像流属性
fncallback	回调函数句柄
enRecType	录像类型
unRecAttr	录像属性枚举
stRbufAttr	录像 ringbuffer 属性
stSplitAttr	录像切分属性
enable_subtitle	字幕使能
enable_file_alignment	文件对齐使能
enable_emrfile_from_normfile	普通录像文件转为紧急录像文件使能
subtitle_framerate	字幕帧率
u32PreRecTimeSec	录像预录时间（单位：秒）
u32PostRecTimeSec	录像后录时间（单位：秒）
s32MemRecPreSec	缓存大小
device_model	设备模型
prealloc_size	预分配大小
id	录像 id

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.3.2.29 CVI_RECORDER_FRAME_STREAM_S

【说明】

定义编码码流

【定义】

```

typedef struct CVI_FRAME_STREAM_S {
    CVI_MUXER_FRAME_TYPE_E type;
    bool vftype[CVI_FRAME_STREAM_SEGMENT_MAX_NUM];
    int32_t num;
    uint64_t vi_pts[CVI_FRAME_STREAM_SEGMENT_MAX_NUM];
    unsigned char *data[CVI_FRAME_STREAM_SEGMENT_MAX_NUM];
    size_t len[CVI_FRAME_STREAM_SEGMENT_MAX_NUM];
    unsigned char *thumbnail_data;
    size_t thumbnail_len;
} CVI_RECORDER_FRAME_STREAM_S;

```

【成员】

成员名称	描述
type	复用器帧类型
vftype	是否为 I 帧
num	数目
vi_pts	pts
data	数据
len	长度
thumbnail_data	缩略图数据
thumbnail_len	缩略图长度

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.4 Ringbuffer

6.3.4.1 API 参考

该组件提供以下 API:

- CVI_RBUF_Init : ringbuffer 初始化
- CVI_RBUF_DeInit : ringbuffer 反初始化
- CVI_RBUF_DataCnt : ringbuffer 中剩下的 frame 个数
- CVI_RBUF_Copy_In : 拷贝数据到 ringbuffer
- CVI_RBUF_Copy_Out : 从 ringbuffer 拷贝数据至目标地址
- CVI_RBUF_Copy_OutTmp : 将 ringbuffer 数据拷贝到目标地址, 用于文件切分
- CVI_RBUF_Unused : ringbuffer 未写入空间大小
- CVI_RBUF_Refresh_In : 刷新 ringbuffer 写入信息
- CVI_RBUF_Refresh_Out : 刷新 ringbuffer 读出信息
- CVI_RBUF_Refresh_OutTmp : 更新 ringbuffer 读出数组 outtmp
- CVI_RBUF_ShowLog : 输出 ringbuffer 信息
- CVI_RBUF_Get_Totalsize : ringbuffer 空间大小
- CVI_RBUF_Get_InSize : 已写入数据的大小
- CVI_RBUF_Reset : ringbuffer 重置

6.3.4.1.1 CVI_RBUF_Init

【描述】

ringbuffer 初始化

【语法】

```
int32_t CVI_RBUF_Init(void **rbuf, int32_t size, const char *name, int32_t outcnt);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入
size	分配大小	输入
outcnt	输出数组大小	输入
rbuf	ringbuffer	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.2 CVI_RBUF_DeInit

【描述】

ringbuffer 反初始化

【语法】

```
void CVI_RBUF_DeInit(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

无

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.3 CVI_RBUF_DataCnt

【描述】

ringbuffer 中剩下的帧数

【语法】

```
uint32_t CVI_RBUF_DataCnt(void *rbuf, int32_t type);
```

【参数】

参数名称	描述	输入/输出
type	读取的类型	输入
rbuf	ringbuffer	输入

【返回值】

返回值	描述
cnt	ringbuffer 中剩下的帧数

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.4 CVI_RBUF_Unused**【描述】**

ringbuffer 未使用空间大小

【语法】

```
uint32_t CVI_RBUF_Unused(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	失败
非 0	ringbuffer 未使用空间大小

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.5 CVI_RBUF_Copy_In

【描述】

拷贝数据到 ringbuffer

【语法】

```
int32_t CVI_RBUF_Copy_In(void *rbuf, void *src, int32_t len, int32_t off);
```

【参数】

参数名称	描述	输入/输出
src	数据源	输入
len	拷贝长度	输入
off	偏移量	输入
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.6 CVI_RBUF_Copy_Out

【描述】

从 ringbuffer 拷贝数据至目标地址

【语法】

```
int32_t CVI_RBUF_Copy_Out(void *rbuf, void *dst, int32_t len, int32_t off, int32_t type);
```

【参数】

参数名称	描述	输入/输出
dst	目标地址	输入
type	拷贝类型	输入
off	偏移量	输入
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvt_rbuf.h
- 库文件: libcvt_ringbuffer.a/libcvt_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.7 CVI_RBUF_Copy_OutTmp

【描述】

将 ringbuffer 数据拷贝到目标地址，用于文件切分

【语法】

```
int32_t CVI_RBUF_Copy_OutTmp(void *rbuf, void *dst, int32_t len, int32_t off, int32_t type);
```

【参数】

参数名称	描述	输入/输出
len	大小	输入
off	偏移量	输入
type	类型	输入
rbuf	ringbuffer	输入
dst	目标地址	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.8 CVI_RBUF_Refresh_In

【描述】

刷新 ringbuffer 写入信息

【语法】

```
void CVI_RBUF_Refresh_In(void *rbuf, int32_t off);
```

【参数】

参数名称	描述	输入/输出
off	偏移量	输入
rbuf	ringbuffer	输入

【返回值】

无

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.9 CVI_RBUF_Refresh_Out

【描述】

刷新 ringbuffer 读出信息

【语法】

```
void CVI_RBUF_Refresh_Out(void *rbuf, int32_t off, int32_t type);
```

【参数】

参数名称	描述	输入/输出
off	偏移量	输入
rbuf	ringbuffer	输入
type	类型	输入

【返回值】

无

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.10 CVI_RBUF_Refresh_OutTmp

【描述】

更新 ringbuffer 读出数组 outtmp

【语法】

```
void CVI_RBUF_Refresh_OutTmp(void *rbuf, int32_t off, int32_t type);
```

【参数】

参数名称	描述	输入/输出
off	偏移量	输入
type	类型	输入
rbuf	ringbuffer	输入

【返回值】

无

【需求】

- 头文件: cvl_rbuf.h
- 库文件: libcvl_ringbuffer.a/libcvl_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.11 CVI_RBUI_ShowLog

【描述】

输出 ringbuffer 信息

【语法】

```
void CVI_RBUI_ShowLog(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

无

【需求】

- 头文件: cvl_rbuf.h
- 库文件: libcvl_ringbuffer.a/libcvl_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.12 CVI_RBUF_Reset

【描述】

ringbuffer 重置

【语法】

```
int32_t CVI_RBUF_Reset(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.13 CVI_RBUF_Get_InSize

【描述】

已写入数据大小

【语法】

```
uint64_t CVI_RBUF_Get_InSize(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	失败
非 0	已写入数据大小

【需求】

- 头文件: cvi_rbuf.h
- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.1.14 CVI_RBUF_Get_Totalsize

【描述】

ringbuffer 空间大小

【语法】

```
int32_t CVI_RBUF_Get_Totalsize(void *rbuf);
```

【参数】

参数名称	描述	输入/输出
rbuf	ringbuffer	输入

【返回值】

返回值	描述
0	失败
非 0	空间大小

【需求】

- 头文件: cvi_rbuf.h

- 库文件: libcvi_ringbuffer.a/libcvi_ringbuffer.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.4.2 数据类型

相关数据类型定义如下:

- `CVI_RBUF_INFO_S`: 定义 ringbuffer 信息结构体。
- `RINGBUF_OUTPTR_CNT`: 定义输出种类数量。

6.3.4.2.1 RINGBUF_OUTPTR_CNT

【说明】

定义输出种类数量。

【定义】

```
#define RINGBUF_OUTPTR_CNT 4
```

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.4.2.2 CVI_RBUF_INFO_S

【说明】

定义 ringbuffer 信息结构体。

【定义】

```
typedef struct CVI_RBUF_INFO_S{  
    int32_t size;  
    uint64_t in;  
    uint64_t inCnt;  
    uint64_t out[RINGBUF_OUTPTR_CNT];  
    uint64_t outTmp[RINGBUF_OUTPTR_CNT];  
    uint64_t outCnt[RINGBUF_OUTPTR_CNT];  
    int32_t used;
```

(下页继续)

(续上页)

```
void *data;  
char name[64];  
}CVI_RBUF_INFO_S;
```

【成员】

成员名称	描述
size	ringbuffer 空间大小
in	已写入数据大小
inCnt	已写入数据块个数
out[RINGBUF_OUTPTR_CNT]	已读出数据大小数组
outTmp[RINGBUF_OUTPTR_CNT]	已读出数据大小数组，用于文件切分
outCnt[RINGBUF_OUTPTR_CNT]	已读出数据块个数数组
used	已使用数组大小
data	数据
name[64]	ringbuffer 名字

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5 Record_Service

该组件是在 record 组件上再封装一层，用来接受外部参数，其专门用于对接 APP 层，提供的是录制的视频服务。

6.3.5.1 API 参考

该组件提供以下 API：

- CVI_RECORD_SERVICE_Create：创建录像服务
- CVI_RECORD_SERVICE_Destroy：销毁录像服务
- CVI_RECORD_SERVICE_UpdateParam：更新录像服务参数
- CVI_RECORD_SERVICE_StartRecord：开启录像服务
- CVI_RECORD_SERVICE_StopRecord：暂停录像服务
- CVI_RECORD_SERVICE_StartTimelapseRecord：开启缩时录像服务
- CVI_RECORD_SERVICE_StopTimelapseRecord：停止缩时录像服
- CVI_RECORD_SERVICE_EventRecord：开启事件录像服务
- CVI_RECORD_SERVICE_StopEventRecord：停止事件录像服务
- CVI_RECORD_SERVICE_StartMute：开启静音服务

- CVI_RECORD_SERVICE_StopMute：停止静音服务
- CVI_RECORD_SERVICE_PivCapture：开启抓拍服务
- CVI_RECORD_SERVICE_WaitPivFinish：等待抓拍结束服务

6.3.5.1.1 CVI_RECORD_SERVICE_Create

【描述】

创建录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_Create(CVI_RECORD_SERVICE_HANDLE_T *hdl, CVI_RECORD_SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
param	录像服务参数	输入
hdl	录像服务句柄	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_record_service.h/cvi_command_record.h
- 库文件：libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.2 CVI_RECORD_SERVICE_Destroy

【描述】

销毁录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_Destroy(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.3 CVI_RECORD_SERVICE_UpdateParam

【描述】

更新录像服务参数

【语法】

```
int32_t CVI_RECORD_SERVICE_UpdateParam(CVI_RECORD_SERVICE_HANDLE_T hdl,  
→CVI_RECORD_SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
param	录像服务参数	输入
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.4 CVI_RECORD_SERVICE_StartRecord**【描述】**

开启录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StartRecord(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.5 CVI_RECORD_SERVICE_StopRecord

【描述】

暂停录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StopRecord(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.6 CVI_RECORD_SERVICE_StartTimelapseRecord

【描述】

开启缩时录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StartTimelapseRecord(CVI_RECORD_SERVICE_HANDLE_T  
↪hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.7 CVI_RECORD_SERVICE_StopTimelapseRecord

【描述】

停止缩时录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StopTimelapseRecord(CVI_RECORD_SERVICE_HANDLE_T  
→hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.8 CVI_RECORD_SERVICE_EventRecord

【描述】

开启事件录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_EventRecord(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.9 CVI_RECORD_SERVICE_StopEventRecord

【描述】

停止事件录像服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StopEventRecord(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.10 CVI_RECORD_SERVICE_StartMute

【描述】

开启静音服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StartMute(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.11 CVI_RECORD_SERVICE_StopMute**【描述】**

停止静音服务

【语法】

```
int32_t CVI_RECORD_SERVICE_StopMute(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.12 CVI_RECORD_SERVICE_PivCapture

【描述】

开启抓拍服务

【语法】

```
int32_t CVI_RECORD_SERVICE_PivCapture(CVI_RECORD_SERVICE_HANDLE_T hdl, char_
↪ *file_name);
```

【参数】

参数名称	描述	输入/输出
file_name	文件名	输入
hdl	录像服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.1.13 CVI_RECORD_SERVICE_WaitPivFinish

【描述】

等待抓拍结束服务

【语法】

```
void CVI_RECORD_SERVICE_WaitPivFinish(CVI_RECORD_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	录像服务句柄	输入

【返回值】

无

【需求】

- 头文件: cvi_record_service.h/cvi_command_record.h
- 库文件: libcvi_record_service.a/libcvi_record_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.5.2 数据类型

相关数据类型定义如下:

- CVI_RECORD_SERVICE_FILE_TYPE_E : 定义录像文件类型
- CVI_RECORD_SERVICE_VIDEO_CODEC_E : 定义视频编码类型
- CVI_RECORD_SERVICE_AUDIO_CODEC_E : 定义音频编码信息
- CVI_RECORD_SERVICE_VENC_BIND_MODE_E : 定义 VENC 绑定模式
- CVI_RECORD_SERVICE_PARAM_S : 定义录像服务参数
- CVI_RECORD_SERVICE_GPS_RMCINFO_S : 定义 GPS RMC 信息
- CVI_RECORD_SERVICE_GPS_INFO_S : 定义 gps 信息
- CVI_RECORD_SERVICE_HANDLE_T : 定义录像服务句柄

6.3.5.2.1 CVI_RECORD_SERVICE_HANDLE_T

【说明】

定义录像服务句柄

【定义】

```
typedef void *CVI_RECORD_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.2 CVI_RECORD_SERVICE_FILE_TYPE_E

【说明】

定义录像文件类型

【定义】

```
typedef enum CVI_RECORD_SERVICE_FILE_TYPE_E {  
    CVI_RECORD_SERVICE_FILE_TYPE_MP4 = 0,  
    CVI_RECORD_SERVICE_FILE_TYPE_MOV,  
    CVI_RECORD_SERVICE_FILE_TYPE_TS,  
    CVI_RECORD_SERVICE_FILE_TYPE_ES, // *.264 or *.265 file  
    CVI_RECORD_SERVICE_FILE_TYPE_NONE, // No saving  
    CVI_RECORD_SERVICE_FILE_TYPE_MAX  
} CVI_RECORD_SERVICE_FILE_TYPE_E;
```

【成员】

成员名称	描述
CVI_RECORD_SERVICE_FILE_TYPE_MP4	
CVI_RECORD_SERVICE_FILE_TYPE_MOV	
CVI_RECORD_SERVICE_FILE_TYPE_TS	
CVI_RECORD_SERVICE_FILE_TYPE_ES	基本流
CVI_RECORD_SERVICE_FILE_TYPE_NONE	不写入

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.3 CVI_RECORD_SERVICE_VIDEO_CODEC_E

【说明】

定义视频编码类型

【定义】

```
typedef enum CVI_RECORD_SERVICE_VIDEO_CODEC_E {  
    CVI_RECORD_SERVICE_VIDEO_CODEC_H264 = 0,  
    CVI_RECORD_SERVICE_VIDEO_CODEC_H265,  
    CVI_RECORD_SERVICE_VIDEO_CODEC_MAX,  
} CVI_RECORD_SERVICE_VIDEO_CODEC_E;
```

【成员】

成员名称	描述
CVI_RECORD_SERVICE_VIDEO_CODEC_H264	H264
CVI_RECORD_SERVICE_VIDEO_CODEC_H265	H265

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.4 CVI_RECORD_SERVICE_AUDIO_CODEC_E

【说明】

定义音频编码类型

【定义】

```
typedef enum CVI_RECORD_SERVICE_AUDIO_CODEC_E {  
    CVI_RECORD_SERVICE_AUDIO_CODEC_NONE,  
    CVI_RECORD_SERVICE_AUDIO_CODEC_PCM,  
    CVI_RECORD_SERVICE_AUDIO_CODEC_AAC, // not support yet  
    CVI_RECORD_SERVICE_AUDIO_CODEC_BUTT  
} CVI_RECORD_SERVICE_AUDIO_CODEC_E;
```

【成员】

成员名称	描述
CVI_RECORD_SERVICE_AUDIO_CODEC_NONE	NONE
CVI_RECORD_SERVICE_AUDIO_CODEC_PCM	PCM
CVI_RECORD_SERVICE_AUDIO_CODEC_AAC	AAC

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.5 CVI_RECORD_SERVICE_VENC_BIND_MODE_E**【说明】**

定义 VENC 绑定模式

【定义】

```
typedef enum CVI_RECORD_SERVICE_VENC_BIND_MODE_E {  
    CVI_RECORD_SERVICE_VENC_BIND_MODE_NONE,  
    CVI_RECORD_SERVICE_VENC_BIND_MODE_VPSS,  
    CVI_RECORD_SERVICE_VENC_BIND_MODE_VI, // not support yet  
    CVI_RECORD_SERVICE_VENC_BIND_MODE_BUTT  
} CVI_RECORD_SERVICE_VENC_BIND_MODE_E;
```

【成员】

成员名称	描述
CVI_RECORD_SERVICE_VENC_BIND_MODE_NONE	不绑定
CVI_RECORD_SERVICE_VENC_BIND_MODE_VI	绑定 VI, 目前不支持
CVI_RECORD_SERVICE_VENC_BIND_MODE_VPSS	绑定 VPSS

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.6 CVI_RECORD_SERVICE_PARAM_S**【说明】**

定义录像服务参数

【定义】

```
typedef struct CVI_RECORD_SERVICE_PARAM_S {  
    int32_t recorder_id;  
  
    bool enable_record_on_start;  
    bool enable_perf_on_start;  
    bool enable_debug_on_start;  
    bool enable_subtitle;  
  
    // Recorder  
    int32_t rec_mode;  
    uint32_t rec_width;
```

(下页继续)

(续上页)

```
uint32_t rec_height;
float framerate;
uint32_t gop;
int32_t bitrate_kbps;
int32_t recorder_file_type;
CVI_RECORD_SERVICE_VIDEO_CODEC_E recorder_video_codec;
CVI_RECORD_SERVICE_AUDIO_CODEC_E recorder_audio_codec;
uint64_t recorder_split_interval_ms;
#define CS_PARAM_MAX_FILENAME_LEN (128)
char recorder_save_dir_base[CS_PARAM_MAX_FILENAME_LEN];
bool audio_recorder_enable;
int32_t audio_sample_rate;
int32_t audio_channels;
int32_t audio_num_per_frame;
int32_t event_recorder_pre_recording_sec;
int32_t event_recorder_post_recording_sec;
float timelapse_recorder_framerate;
int32_t timelapse_recorder_gop_interval;
int32_t memory_buffer_sec;
int32_t pre_alloc_unit;
void *cont_recorder_event_cb;
void *event_recorder_event_cb;
void *timelapse_recorder_event_cb;
void *get_subtitle_cb;
void *generate_filename_cb;
void *get_rec_dir_type_cb;
void *get_gps_info_cb;

// PIV
uint32_t prealloclen;

int32_t vproc_chn_id_venc;
int32_t vproc_chn_id_thumbnail;
CVI_RECORD_SERVICE_VENC_BIND_MODE_E venc_bind_mode;

CVI_MAPI_VPROC_HANDLE_T rec_vproc;
CVI_MAPI_VPROC_HANDLE_T thumbnail_vproc;
CVI_MAPI_VENC_HANDLE_T rec_venc_hdl;
CVI_MAPI_VENC_HANDLE_T thumbnail_venc_hdl;
uint32_t thumbnail_bufsize;
CVI_MAPI_VENC_HANDLE_T piv_venc_hdl;
uint32_t piv_bufsize;
int32_t vproc_id_rec;

float short_file_ms;
char devmodel[32];
} CVI_RECORD_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
recorder_id	录像 id

下页继续

表 6.1 – 续上页

成员名称	描述
enable_record_on_start	设备开机录像使能
enable_perf_on_start	设备开机 perf 使能
enable_debug_on_start	设备开机 debug 使能
enable_subtitle	字幕使能
rec_mode	录像模式
rec_width	录像宽度
rec_height	录像高度
framerate	帧率
gop	gop 值
bitrate_kbps	码率
recorder_file_type	录像文件类型
recorder_video_codec	视频编码类型
recorder_audio_codec	音频编码类型
recorder_split_interval_ms	录像切分时间
recorder_save_dir_base	录像文件夹类型
audio_recorder_enable	音频使能
audio_sample_rate	采样率
audio_channels	声道数
audio_num_per_frame	每帧采样数
event_recorder_pre_recording_sec	事件录像预录时间
event_recorder_post_recording_sec	事件录像后录时间
timelapse_recorder_framerate	缩时录像帧率
timelapse_recorder_gop_interval	缩时录像间隔，例如 1s 代表 1s 原录像视频取一帧
memory_buffer_sec	缓存区秒数
pre_alloc_unit	预分配大小
cont_recorder_event_cb	普通录像回调函数指针
event_recorder_event_cb	事件录像回调函数指针
timelapse_recorder_event_cb	缩时录像回调函数指针
get_subtitle_cb	获取字幕回调函数指针
generate_filename_cb	生成文件名回调函数指针
get_rec_dir_type_cb	获取文件夹类型回调函数指针
get_gps_info_cb	获取 gps 信息回调函数指针
prealloclen	用于抓拍的预分配长度
vproc_chn_id_venc	vpss 绑定 venc 的通道 id
vproc_chn_id_thumbnail	vpss 绑定 venc 的缩略图通道 id
venc_bind_mode	venc 绑定模式
rec_vproc	录像 vpss 句柄
thumbnail_vproc	用于缩略图的录像 vpss 句柄
rec_venc_hdl	视频编码句柄
thumbnail_venc_hdl	用于缩略图的编码句柄
thumbnail_bufsize	用于缩略图的缓存大小
piv_venc_hdl	用于抓拍的编码句柄
piv_bufsize	用于抓拍的缓存大小
vproc_id_rec	用于抓拍的 vpss id
short_file_ms	短文件时间

下页继续

表 6.1 – 续上页

成员名称	描述
devmodel	设备模型

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.7 CVI_RECORD_SERVICE_GPS_RMCINFO_S

【说明】

定义 GPS RMC 信息

【定义】

```
typedef struct MEDIA_GPS_RMCINFO{
    //都是采集同一条RMC语句里的数据，GNRMC 或GPRMC
    uint32_t Hour;      //gps信息里输出的小时
    uint32_t Minute;    //gps信息里输出的分
    uint32_t Second;    //gps信息里输出的秒
    uint32_t Year;      //gps信息里输出的年
    uint32_t Month;     //gps信息里输出的月
    uint32_t Day;       //gps信息里输出的天
    char Status;        //gps信息里输出的状态
    char NSInd;         //gps信息里输出的NSInd
    char EWInd;         //gps信息里输出的EWInd
    char reserved;      //gps信息里输出的reserved
    double Latitude;    //gps信息里输出的纬度，
    →注意这是double型，RMC语句里，直接采集进去，不需要任何转化
    double Longitude;   //
    →gps信息里输出的经度，注意这是double型，RMC语句里，直接采集进去，不需要任何转化
    float Speed;        //gps信息里输出的速度
    float Angle;        //gps信息里输出的方向
    char ID[20];        //这个ID，读取RMC语法的ID,加密模组才有
    uint32_t GsensorX;  //重力加速X
    uint32_t GsensorY;  //重力加速Y
    uint32_t GsensorZ;  //重力加速Z
}CVI_RECORD_SERVICE_GPS_RMCINFO_S;
```

【成员】

上同

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.5.2.8 CVI_RECORD_SERVICE_GPS_INFO_S

【说明】

定义 gps 信息

【定义】

```
typedef struct MEDIA_GPS_INFO{  
    int32_t init;  
    CVI_RECORD_SERVICE_GPS_RMCINFO_S rmc_info;  
}CVI_RECORD_SERVICE_GPS_INFO_S;
```

【成员】

成员名称	描述
init	是否初始化
rmc_info	gps rmc 信息

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.6 Audio_Service

该服务提供的是录制的音频服务。

该组件提供音频输入服务

6.3.6.1 API 参考

该组件提供以下 API:

- CVI_AUDIO_SERVICR_ACAP_CallbackSet : 设置音频输入回调函数
- CVI_AUDIO_SERVICR_ACAP_CallbackUnset : 取消音频输入回调函数
- CVI_AUDIO_SERVICR_ACAP_AacCallbackSet : 设置音频输入 aac 回调函数
- CVI_AUDIO_SERVICR_ACAP_AacCallbackUnset : 取消音频输入 aac 回调函数
- CVI_AUDIO_SERVICR_ACAP_TaskStart : 开启音频输入任务
- CVI_AUDIO_SERVICR_ACAP_TaskStop : 停止音频输入任务

6.3.6.1.1 CVI_AUDIO_SERVICR_ACAP_CallbackSet

【描述】

设置音频输入回调函数

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_CallbackSet(const char *name, CVI_AUDIO_SERVICR_
↪ACAP_GET_FRAME_CALLBACK *cb, void *arg);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入
cb	获取音频回调函数	输入
arg	回调函数参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_audio_service.h
- 库文件: libcvi_audio_service.a/libcvi_audio_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.1.2 CVI_AUDIO_SERVICR_ACAP_CallbackUnset

【描述】

取消音频输入回调函数

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_CallbackUnset(const char *name);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_audio_service.h
- 库文件: libcvi_audio_service.a/libcvi_audio_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.1.3 CVI_AUDIO_SERVICR_ACAP_AacCallbackSet

【描述】

设置音频输入 aac 回调函数

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_AacCallbackSet(const char *name, CVI_AUDIO_
↪SERVICR_ACAP_GET_AAC_FRAME_CALLBACK *cb, void *arg);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入
cb	获取音频回调函数	输入
arg	回调函数参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_audio_service.h`
- 库文件: `libcvi_audio_service.a/libcvi_audio_service.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.1.4 CVI_AUDIO_SERVICR_ACAP_AacCallbackUnset

【描述】

取消音频输入 aac 回调函数

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_AacCallbackUnset(const char *name);
```

【参数】

参数名称	描述	输入/输出
name	名字	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_audio_service.h`
- 库文件: `libcvi_audio_service.a/libcvi_audio_service.so`

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.1.5 CVI_AUDIO_SERVICR_ACAP_TaskStart

【描述】

开启音频输入任务

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_TaskStart(CVI_MAPI_ACAP_HANDLE_T acap_handle,  
↪ CVI_MAPI_AENC_HANDLE_T aenc_handle);
```

【参数】

参数名称	描述	输入/输出
acap_handle	音频输入句柄	输入
aenc_handle	音频编码句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_audio_service.h
- 库文件: libcvi_audio_service.a/libcvi_audio_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.1.6 CVI_AUDIO_SERVICR_ACAP_TaskStop

【描述】

停止音频输入任务

【语法】

```
int32_t CVI_AUDIO_SERVICR_ACAP_TaskStop();
```

【参数】

无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvt_audio_service.h
- 库文件: libcvt_audio_service.a/libcvt_audio_service.so

【注意】

无

【举例】

无

【相关主题】

无

6.3.6.2 数据类型

相关数据类型定义如下:

- `CVI_AUDIO_SERVICR_ACAP_GET_FRAME_CALLBACK`: 定义获取音频帧回调函数
- `CVI_AUDIO_SERVICR_ACAP_GET_AAC_FRAME_CALLBACK`: 定义获取 aac 音频帧回调函数

6.3.6.2.1 CVI_AUDIO_SERVICR_ACAP_GET_FRAME_CALLBACK

【说明】

定义获取音频帧回调函数

【定义】

```
typedef void (CVI_AUDIO_SERVICR_ACAP_GET_FRAME_CALLBACK) (const AUDIO_  
↪FRAME_S *frame, const AEC_FRAME_S *aec_frame, void *arg);
```

【成员】

成员名称	描述
frame	音频帧
aec_frame	音频编码帧
arg	参数

【注意事项】

无。

【相关数据类型及接口】

无。

6.3.6.2.2 CVI_AUDIO_SERVICR_ACAP_GET_AAC_FRAME_CALLBACK

【说明】

定义获取 aac 音频帧回调函数

【定义】

```
typedef void (CVI_AUDIO_SERVICR_ACAP_GET_AAC_FRAME_CALLBACK) (const AUDIO_
↪STREAM_S *stream, void *arg);
```

【成员】

成员名称	描述
stream	音频流
arg	参数

【注意事项】

无。

【相关数据类型及接口】

无。

7 播放器模块

7.1 概述

播放器模块为用户提供本地媒体文件回放功能，可以播放视频和音频文件。它包含解封装功能与播控功能，解封装功能使用对媒体文件的解析，如把 MP4 解析成 ES 流；播控功能包括播放控制、内部数据管理等。此外，本模块提供了回调注册接口，让用户必须注册回调接口以接收文件播完的消息。可以接受播放进度回调消息以及播放错误消息。播放器模块在整个系统中的位置如图 7-1 所示：

- 本地媒体文件经过解复用器组件，缩略图组件提取缩略图，并展示在 ui 界面上。
- 点击缩略图，调用 playback Service，最终音视频输出。
- Playback Service 是基于 player 组件封装，其目的是专门对接 APP 层。
- MAPI 是多媒体软件接口。

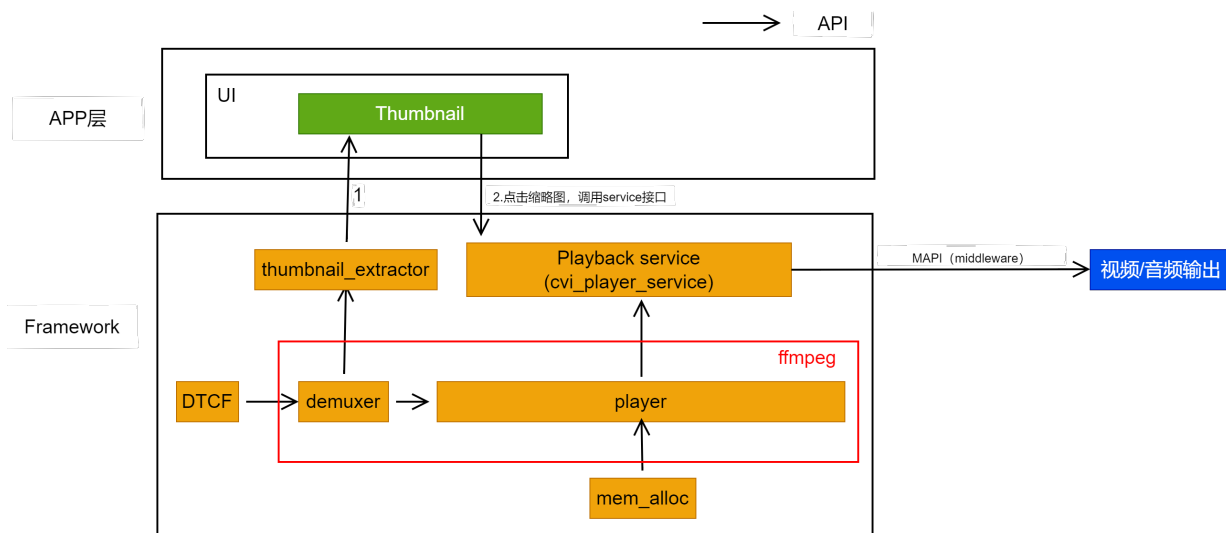


图 7.1: 播放器模块上下文

注意:

- 播放器使用第三方库 ffmpeg 进行解复用和解码，请事先了解 ffmpeg。

7.2 播放器模块处理流程

Player 播放接口流程，使用的是 Player 接口，即 CVI_PLAYER，其调用逻辑大致如下：

- 打开网络视频流 CVI_PLAYER_Init
- 播放器创建 CVI_PLAYER_Create
- 设置数据源 CVI_PLAYER_SetDataSource
- 获取媒体信息 CVI_PLAYER_GetMediaInfo
- 初始化系统
- 设置 AO 处理函数 CVI_PLAYER_SetCustomArgAOHandler
- 设置 VO 处理函数 CVI_PLAYER_SetCustomArgVOHandler
- 设置事件处理函数 CVI_PLAYER_SetCustomArgEventHandler
- 设置视频解码器处理函数 CVI_PLAYER_SetVideoCustomArgDecodeHandler，注意音频使用软解码，视频只支持 H264，H265，MJPEG 格式硬解码，其余格式走软解码。
- 播放 CVI_PLAYER_Play
- 处理各种事件，如暂停，播放，恢复，快进等事件
- 退出并销毁 CVI_PLAYER_SERVICE_Destroy

7.2.1 获取媒体信息分析

1.Player Service 接口：CVI_PLAYER_SERVICE_GetMediaInfo

2.Player 接口：CVI_PLAYER_GetMediaInfo

3. 调用层次关系：

- Player Service -> player -> Demuxer → ffmpeg
- 主要函数：
 - avformat_open_input 解复用。
 - av_guess_format 获取输出格式，将解复用后得到的 video、audio 流信息放到数组中。

7.2.2 解码分析

解码流程如图 7-2 所示：

- Player Service 接口：CVI_PLAYER_SERVICE_Play
- Player 接口：CVI_PLAYER_Play
- 调用层次关系：Player Service -> Player -> Demuxer -> decode → ffmpeg

- 处理：解复用 (demuxer) -> 根据不同的流 (stream) 创建不同的解码器 (decoder) -> 发送未解码的包 (packet) 到解码器中解码-> 获取解码后的帧 (frame) → 将 video 或者 audio 发送给 vo 或者 ao 进行播放
- 主要函数:
 - avcodec_alloc_context3 根据特定的 CODEC 分配一个 AVCodecContext 结构体
 - avcodec_parameters_to_context 将 AVCodecParameters 结构体中码流参数拷贝到 AVCodecContext 结构体中
 - avcodec_find_decoder 查找解码器
 - avcodec_open2 打开解码器
 - av_frame_alloc 申请内存
 - avcodec_send_packet 发送未解码的包
 - avcodec_receive_frame 获取解码后的帧数据

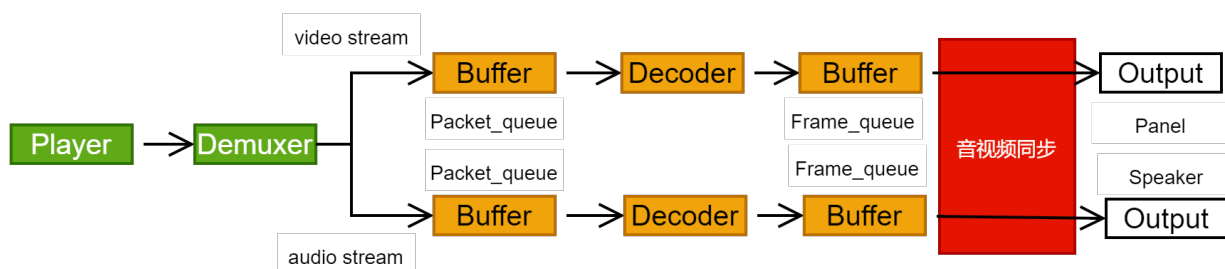


图 7.2: 解码流程

注意:

- avcodec_send_packet 和 avcodec_receive_frame 属于软解码过程，硬解码调用多媒体软件解码接口。

7.3 播放器状态图

播放器状态图如图 7-3 所示：

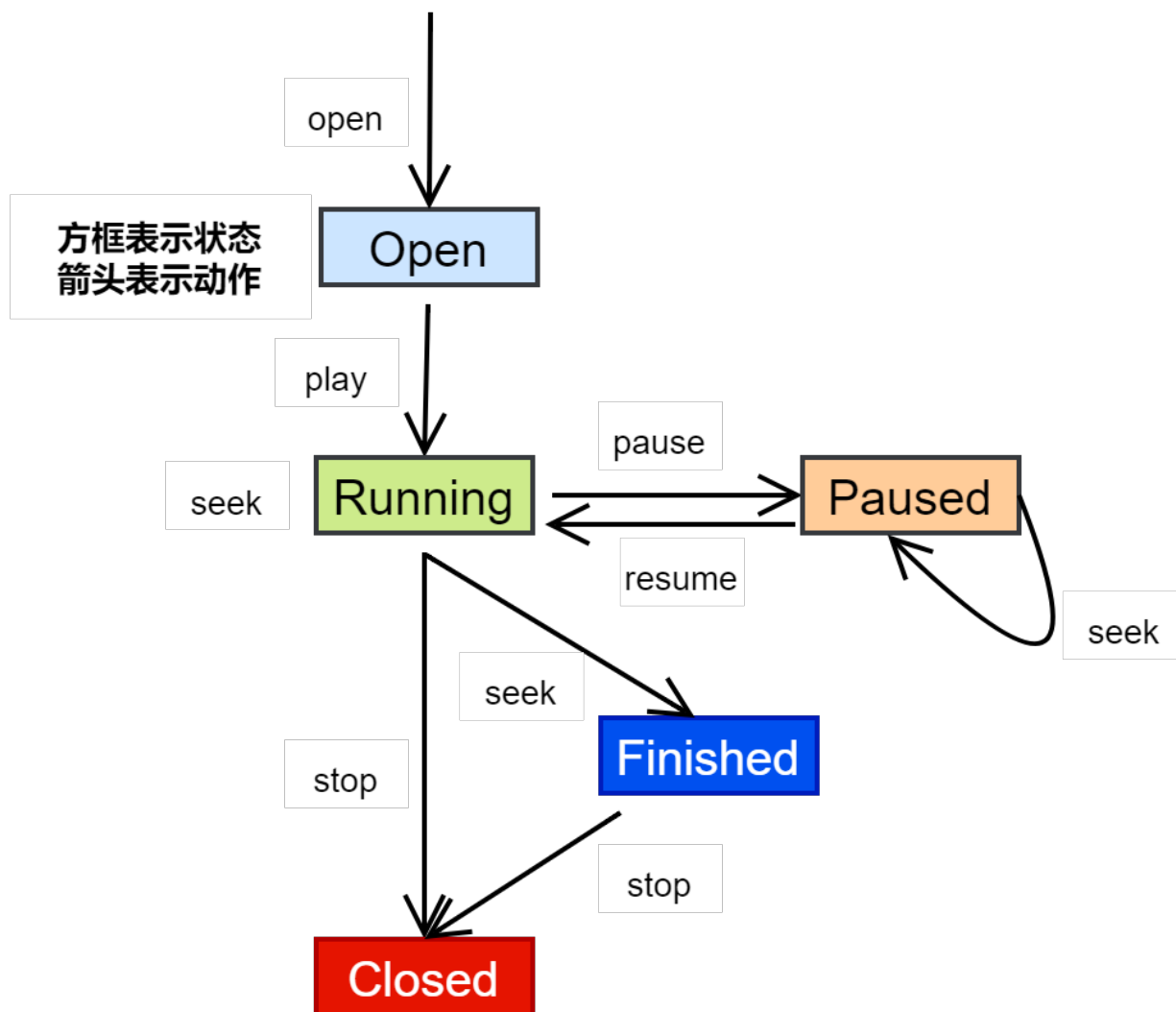


图 7.3: 播放器状态图

7.4 组件 API

讲述组件 Demuxer, Thumbnail_extractor, Player 和 Playback service。

7.4.1 Demuxer

Demuxer（解复用器）组件主要是对媒体文件进行解封装，解封装得到 stream(packt)。它的实现是用 C++ 语言，所以阅读此章节需要具备一定的 C++ 语言基础，这里介绍该组件的两个类，即基类 Demuxer 和派生类 FFmpegDemuxer 及 API 参考。

7.4.1.1 基类 Demuxer

```
namespace cvi_demuxer {
template <typename T>
class Demuxer
{
public:
    virtual ~Demuxer() = default; //析构函数

    virtual int32_t read(T *data) = 0; //读取数据

    virtual int32_t open() //打开解复用器
    {
        opened = true;
        return 0;
    }

    virtual void close() //关闭解复用器
    {
        opened = false;
    }

    virtual void pause() //暂停解复用器，用于播放器暂停
    {
        paused = true;
    }

    virtual void resume() //恢复解复用器，用于恢复播放
    {
        paused = false;
    }

    bool isOpened() const //解复用器是否打开
    {
        return opened;
    }

    bool isPaused() const //解复用器是否停止
    {
        return paused;
    }

protected:
    std::atomic<bool> opened{false}; //初始化
    std::atomic<bool> paused{false};
};

} // namespace cvi_demuxer
```

7.4.1.2 派生类 FFmpegDemuxer

```

namespace cvi_demuxer {

class FFmpegDemuxer final : public Demuxer<AVPacket>
{
public:
    ~FFmpegDemuxer();//析构函数

    virtual int32_t open() override;//打开解复用器
    int32_t openTs();//打开解复用器, 包括ts文件, 现在播放器都是使用这个接口打开解复用器
    virtual void close() override;//关闭解复用器
    virtual int32_t read(AVPacket *packet) override;//获取音视频帧的压缩数据
    virtual void pause() override;//暂停解复用器
    virtual void resume() override;//恢复解复用器
    int32_t getNumberOfStream() const;//获取解复用后的基本流数
    int32_t getStreamIndex(int32_t media_type) const;//获取基本流索引
    AVCodecID getVideoCodecId() const;//获取解码器id
    AVStream *getAvStream(int32_t index) const;//获取基本流
    void setInputFormat(AVInputFormat *input_format);//设置输入格式
    char *getInput() const;//获取输入
    void setInput(const std::string &input);//设置输入
    int32_t seek(int64_t target, int64_t offset = 0, int32_t flag = 0);//进行seek操作
    bool isRealTime() const;//是否是实时传输
    double getMaxFrameDuration() const;//获取最大帧持续时间
    void dumpInputInfo() const;//打印输入文件信息
    int64_t getStartTime() const;//获取开始时间
    int32_t getMediaInfo(CviDemuxerMediaInfo &info) const;//获取媒体信息
    int32_t getfilenameextern();//获取文件扩展名
    double getfileinfotime();//获取文件持续时间
    char* getcreationtime();//获取文件创建时间
    int32_t readandcreationtime(AVPacket *packet, char *creationtime);//
    → 获取音视频帧的压缩数据并设置创建时间

private:
    int32_t openInput();//打开输入文件
    int32_t openInputts();//打开输入文件, 包括ts文件, 现在播放器都是使用这个接口打开文件
    void findStreams();//获取基本流的stream_index
    void saveStreamsParameters();//保存视频流基本参数
    void updateStreamInfo();//更新mjpeg信息
    void freeInput();//释放输入
    void updateInputFormatIfNeed(const std::string &input_extension);//更新输入格式
    int32_t tspsparse(CVI_DEMUXER_STREAM_INFO_S *streaminfo);//解析ts流信息
    bool getmjpegstreaminfo();//获取mjpeg流的宽和高

    uint32_t Ue(unsigned char *pBuff, uint32_t nLen, uint32_t &nStartBit);//
    → 无符号指数哥伦布熵解码
    int32_t Se(unsigned char *pBuff, uint32_t nLen, uint32_t &nStartBit);//有符号指数哥伦布熵解码
    unsigned long u(uint32_t BitCount, unsigned char * buf, uint32_t &nStartBit);//
    → 读进连续若干个比特, 并将它们解释为无符号整数
    void de_emulation_prevention(unsigned char* buf, uint32_t* buf_size);//剔除防竞争码0x03
    //解析h264 sps信息, 获取宽高和帧率
    int32_t h264_decode_sps(unsigned char * buf, uint32_t nLen, int32_t &width, int32_t &height,
    → float &fps);

```

(下页继续)

(续上页)

```
private:
    AVFormatContext *av_context{nullptr};
    AVInputFormat *input_format{nullptr};
    char *input{nullptr};
    AVCodecID video_codec_id{AV_CODEC_ID_NONE};
    int32_t stream_index[AVMEDIA_TYPE_NB]{0};
};

} // namespace cvi_demuxer
```

7.4.1.3 API 参考

该组件提供以下 API:

- CVI_DEMUXER_Create : 解复用器创建
- CVI_DEMUXER_Destroy : 解复用器销毁
- CVI_DEMUXER_Open : 打开解复用器
- CVI_DEMUXER_Close : 关闭解复用器
- CVI_DEMUXER_Pause : 暂停解复用器
- CVI_DEMUXER_Resume : 恢复解复用器
- CVI_DEMUXER_SetInput : 设置输入源
- CVI_DEMUXER_Read : 解复用器读取数据
- CVI_DEMUXER_Seek : 解复用器进行跳转
- CVI_DEMUXER_GetMediaInfo : 获取媒体信息

7.4.1.3.1 CVI_DEMUXER_Create

【描述】

解复用器创建

【语法】

```
int32_t CVI_DEMUXER_Create(CVI_DEMUXER_HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvd_demuxer.h/packet.h/media_info.h
- 库文件: libcvd_demuxer.a/libcvd_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.2 CVI_DEMUXER_Destroy

【描述】

解复用器销毁

【语法】

```
int32_t CVI_DEMUXER_Destroy(CVI_DEMUXER_HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvd_demuxer.h/packet.h/media_info.h
- 库文件: libcvd_demuxer.a/libcvd_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.3 CVI_DEMUXER_Open**【描述】**

解复用器打开

【语法】

```
int32_t CVI_DEMUXER_Open(CVI_DEMUXER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.4 CVI_DEMUXER_Close**【描述】**

解复用器关闭

【语法】

```
int32_t CVI_DEMUXER_Close(CVI_DEMUXER_HANDLE_T handle);
```


【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.5 CVI_DEMUXER_Pause

【描述】

解复用器暂停

【语法】

```
int32_t CVI_DEMUXER_Pause(CVI_DEMUXER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h

- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.6 CVI_DEMUXER_Resume

【描述】

解复用器恢复

【语法】

```
int32_t CVI_DEMUXER_Resume(CVI_DEMUXER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.7 CVI_DEMUXER_SetInput

【描述】

设置输入

【语法】

```
int32_t CVI_DEMUXER_SetInput(CVI_DEMUXER_HANDLE_T handle, const char *input);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入
input	文件路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.8 CVI_DEMUXER_Read

【描述】

解复用器读取数据

【语法】

```
int32_t CVI_DEMUXER_Read(CVI_DEMUXER_HANDLE_T handle, CVI_DEMUXER_PACKET_S *packet);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入
packet	解复用后的包	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.9 CVI_DEMUXER_Seek

【描述】

解复用器进行跳转

【语法】

```
int32_t CVI_DEMUXER_Seek(CVI_DEMUXER_HANDLE_T handle, const int64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入
time_in_ms	需要跳转的时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h

- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.3.10 CVI_DEMUXER_GetMediaInfo

【描述】

获取媒体信息

【语法】

```
int32_t CVI_DEMUXER_GetMediaInfo(CVI_DEMUXER_HANDLE_T handle, CVI_DEMUXER_
↳ MEDIA_INFO_S *info);
```

【参数】

参数名称	描述	输入/输出
handle	解复用器句柄	输入
info	解复用后的文件信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_demuxer.h/packet.h/media_info.h
- 库文件: libcvi_demuxer.a/libcvi_demuxer.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.1.4 数据类型

相关数据类型定义如下：

- CVI_DEMUXER_STREAM_RESOLUTION_S：定义流分辨率
- CVI_DEMUXER_MEDIA_INFO_S：定义媒体信息
- CVI_DEMUXER_STREAM_INFO_S：定义码流信息。
- CVI_DEMUXER_PACKET_S：定义解复用后的包信息。
- CVI_DEMUXER_HANDLE_T：定义解复用句柄。

7.4.1.4.1 CVI_DEMUXER_HANDLE_T

【说明】

定义解复用句柄。

【定义】

```
typedef void* CVI_DEMUXER_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.1.4.2 CVI_DEMUXER_STREAM_RESOLUTION_S

【说明】

流分辨率

【定义】

```
typedef struct CviDemuxerStreamResolution {  
    int32_t stream_index;  
    uint32_t width;  
    uint32_t height;  
    char codec[16];  
} CVI_DEMUXER_STREAM_RESOLUTION_S;
```

【成员】

成员名称	描述
width	视频宽
height	视频高
codec	解码器

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.1.4.3 CVI_DEMUXER_STREAM_INFO_S

【说明】

码流信息。

【定义】

```
typedef struct CviDemuxerstreaminfo {
    int32_t videonum;
    int32_t videoden;
    int32_t audionum;
    int32_t audioden;
    int32_t videowidth;
    int32_t videoheight;
    float frame_rate;
    int64_t duration;
} CVI_DEMUXER_STREAM_INFO_S;
```

【成员】

成员名称	描述
videonum	视频分子，通常是帧率
videoden	视频分母，通常为 1
audionum	音频分子，通常是 1
audioden	音频分母，通常是采样率
videowidth	视频宽
videoheight	视频高
frame_rate	帧率
duration	时长

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.1.4.4 CVI_DEMUXER_PACKET_S

【说明】

解复用后的包信息。

【定义】

```
typedef struct CviDemuxerPacket
{
    uint8_t *data;
    int32_t size;
    int64_t pts;
    double duration;
    long creationtime;
    int32_t errorcode[4]; // File initialization, thumbnail, Creation time, duration
} CVI_DEMUXER_PACKET_S;
```

【成员】

成员名称	描述
data	数据
size	包大小
pts	pts
duration	时长
creationtime	创建时间
errorcode	错误内容

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.1.4.5 CVI_DEMUXER_MEDIA_INFO_S

【说明】

媒体信息。

【定义】

```
typedef struct CviDemuxerMediaInfo {
    char file_name[64];
    char format[16];
    int32_t width;
    int32_t height;
    uint64_t file_size;
    double start_time_sec;
    double duration_sec;
    double audio_duration_sec;
    double video_duration_sec;
```

(下页继续)

(续上页)

```

    CVI_DEMUXER_STREAM_RESOLUTION_S stream_resolution[CVI_DEMUXER_STREAM_
↪MAX_NUM];
    int32_t used_video_stream_index;
    float frame_rate;
    uint64_t bit_rate;
    uint32_t audio_channel_layout;
    uint32_t sample_rate;
    int32_t used_audio_stream_index;
    char video_codec[16];
    char audio_codec[16];
} CVI_DEMUXER_MEDIA_INFO_S;

```

【成员】

成员名称	描述
file_name	文件名称
format	格式, video:h264/h265 audio:pcm/aac
width	视频宽
height	视频高
file_size	文件大小
start_time_sec	起始时间
duration_sec	文件时长
audio_duration_sec	audio 时长
video_duration_sec	video 时长
stream_resolution	流分辨率
used_video_stream_index	已使用 video stream id 号
frame_rate	帧率
bit_rate	码率
audio_channel_layout	udio 通道数
sample_rate	采样率
video_codec	视频解码器
used_audio_stream_index	已使用 audio stream id 号
audio_codec	音频解码器

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.2 Thumbnail_extractor

该组件用于文件列表显示小图，通常用于文件列表九宫格。通常在 app 层上调用。主要从录像文件中拿缩略图 track 中的数据，把媒体文件中的缩略图数据拿出来提供给上层用于缩略图显示，后续点击该缩略图即可播放相应的文件。

7.4.2.1 API 参考

该组件提供以下 API：

- `CVI_THUMBNAI_L_EXTRACTOR_Create`：创建缩略图提取器
- `CVI_THUMBNAI_L_EXTRACTOR_Destroy`：销毁缩略图提取器
- `CVI_THUMBNAI_L_EXTRACTOR_GetThumbnail`：获取缩略图
- `CVI_THUMBNAI_L_EXTRACTOR_ClearPacket`：释放包数据

7.4.2.1.1 CVI_THUMBNAI_L_EXTRACTOR_Create

【描述】

创建缩略图提取器

【语法】

```
int32_t CVI_THUMBNAI_L_EXTRACTOR_Create(CVI_THUMBNAI_L_EXTRACTOR_HANDLE_
↳T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	缩略图提取句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_thumbnail_extractor.h
- 库文件：libcvi_thumbnail_extractor.a/libcvi_thumbnail_extractor.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.2.1.2 CVI_THUMBNAIL_EXTRACTOR_Destroy

【描述】

销毁缩略图提取器

【语法】

```
int32_t CVI_THUMBNAIL_EXTRACTOR_Destroy(CVI_THUMBNAIL_EXTRACTOR_
↪HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	缩略图提取句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_thumbnail_extractor.h
- 库文件: libcvi_thumbnail_extractor.a/libcvi_thumbnail_extractor.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.2.1.3 CVI_THUMBNAIL_EXTRACTOR_GetThumbnail

【描述】

获取缩略图

【语法】

```
int32_t CVI_THUMBNAIL_EXTRACTOR_GetThumbnail(CVI_THUMBNAIL_EXTRACTOR_
↪HANDLE_T handle, const char *input, CVI_THUMBNAIL_PACKET_S *thumbnail);
```

【参数】

参数名称	描述	输入/输出
handle	缩略图提取句柄	输入
input	文件名	输入
thumbnail	缩略图	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_thumbnail_extractor.h
- 库文件: libcvi_thumbnail_extractor.a/libcvi_thumbnail_extractor.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.2.1.4 CVI_THUMBNAIL_EXTRACTOR_ClearPacket

【描述】

释放包数据

【语法】

```
int32_t CVI_THUMBNAIL_EXTRACTOR_ClearPacket(CVI_THUMBNAIL_PACKET_S *packet);
```

【参数】

参数名称	描述	输入/输出
packet	缩略图	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_thumbnail_extractor.h`
- 库文件: `libcvi_thumbnail_extractor.a/libcvi_thumbnail_extractor.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.2.2 数据类型

相关数据类型定义如下:

- `CVI_THUMBNAIL_PACKET_S`: 定义解复用后的缩略图包信息。
- `CVI_THUMBNAIL_EXTRACTOR_HANDLE_T`: 定义缩略图句柄。

7.4.2.2.1 CVI_THUMBNAIL_EXTRACTOR_HANDLE_T

【说明】

定义缩略图句柄

【定义】

```
typedef void* CVI_THUMBNAIL_EXTRACTOR_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.2.2 CVI_THUMBNAIL_PACKET_S

【说明】

定义解复用后的缩略图包信息。

【定义】

```
typedef CVI_DEMUXER_PACKET_S CVI_THUMBNAIL_PACKET_S;
```

【成员】

参考CVI_DEMUXER_PACKET_S

【注意事项】

无

【相关数据类型及接口】

无。

7.4.3 Player

Player 的内部实现是用 C++ 语言，是参考 ffmpeg 的 ffplay 工具，这里介绍 include 和 src 文件夹下各个子文件夹的大概作用，以及给予用户使用的 API。

7.4.3.1 目录结构

```
/player
├── /include
│   ├── /
│   └── clock: 时钟类, 音频流和视频流都有各自的时钟, 另外还有一个外部时钟; 该播放器音视频同步默认是以音频时钟为主
├── /decoder: 解码器, 层次decoder → av_decoder → video_decoder/audio_decoder/subtitle_decoder
│   ├── /event: 播放事件类型
│   ├── /frame: 定义内部解码帧结构体
│   ├── /packet: 定义内部解码包结构体
│   ├── /queue: 包含包队列与帧队列
│   ├── /service: 定义事件服务
│   └── /stream: 基本流, 层次base_stream → av_stream → video_stream/audio_streamr/
├── subtitle_stream
│   ├── /thread: 内部线程
│   ├── /utils: 工具, 包含数据搬运, 定时器
│   ├── /types.hpp: 定义类型, 比如错误类型, 音视频同步类型
│   ├── /player: 基于事件服务的类, 调用以上类
│   └── cvi_player.h: 用户API
└── /src
    ├── /clock
    ├── /decoder
    ├── /packet
    ├── /queue
    └── /stream
```

(下页继续)

(续上页)

```
| |—— /utils  
| |—— /player  
| |—— cvi_player.cpp  
|—— Makefile
```

注意:

- 各文件的调用逻辑可参考[解码流程](#)。

7.4.3.2 API 参考

该组件提供以下 API:

- CVI_PLAYER_Init : 创建复用器
- CVI_PLAYER_Deinit : 启动复用器
- CVI_PLAYER_Create : 写包进文件
- CVI_PLAYER_Destroy : 停止复用器
- CVI_PLAYER_SetDataSource : 销毁复用器
- CVI_PLAYER_GetDataSource : 获取数据源
- CVI_PLAYER_LightOpen : 打开解复用器
- CVI_PLAYER_Play : 播放
- CVI_PLAYER_Stop : 停止播放器
- CVI_PLAYER_Pause : 暂停播放器
- CVI_PLAYER_Resume : 恢复播放
- CVI_PLAYER_Seek : 跳转
- CVI_PLAYER_TPlay : 快进
- CVI_PLAYER_SetAudioParameters : 设置音频参数
- CVI_PLAYER_SetVideoParameters : 设置视频参数
- CVI_PLAYER_SetAOHandler : 设置音频输出句柄
- CVI_PLAYER_SetCustomArgAOHandler : 自定义设置音频输出句柄
- CVI_PLAYER_SetVOHandler : 设置视频输出句柄
- CVI_PLAYER_SetCustomArgVOHandler : 自定义设置视频输出句柄
- CVI_PLAYER_SetEventHandler : 设置播放事件句柄
- CVI_PLAYER_SetCustomArgEventHandler : 自定义设置播放事件句柄
- CVI_PLAYER_SaveImage : 保存软解码后的视频帧
- CVI_PLAYER_GetMediaInfo : 获取媒体信息

- `CVI_PLAYER_GetPlayInfo` : 获取播放器信息
- `CVI_PLAYER_GetVideoFrame` : 获取视频帧
- `CVI_PLAYER_GetVideoPacket` : 获取视频包
- `CVI_PLAYER_GetVideoExtraPacket` : 获取视频额外包
- `CVI_PLAYER_SetVideoDecodeHandler` : 设置视频解码句柄
- `CVI_PLAYER_SetVideoCustomArgDecodeHandler` : 自定义设置视频解码句柄
- `CVI_PLAYER_SetAudioDecodeHandler` 设置音频解码句柄
- `CVI_PLAYER_SetAudioCustomArgDecodeHandler` : 自定义设置音频解码句柄
- `CVI_PLAYER_PacketContainSps` : 包是否包含 sps
- `CVI_PLAYER_SeekPause` : 在暂停状态下跳转
- `CVI_PLAYER_SeekFlage` : 获取跳转标志
- `CVI_PLAYER_SeekTime` : 跳转到某个时间
- `CVI_PLAYER_PlayerSeep` : 快进快退倍数播放
- `CVI_PLAYER_GetForWardBackWardStatus` : 获取倍数播放状态

7.4.3.2.1 CVI_PLAYER_Init

【描述】

打开网络视频流

【语法】

```
int32_t CVI_PLAYER_Init();
```

【参数】

无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: 目录结构所有头文件
- 库文件: libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.2 CVI_PLAYER_Deinit

【描述】

关闭网络视频流

【语法】

```
int32_t CVI_PLAYER_Deinit();
```

【参数】

无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.3 CVI_PLAYER_Create

【描述】

创建播放器

【语法】

```
int32_t CVI_PLAYER_Create(CVI_PLAYER_HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.4 CVI_PLAYER_Destroy

【描述】

销毁播放器

【语法】

```
int32_t CVI_PLAYER_Destroy(CVI_PLAYER_HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放器句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件

- 库文件: libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.5 CVI_PLAYER_SetDataSource

【描述】

设置数据源

【语法】

```
int32_t CVI_PLAYER_SetDataSource(CVI_PLAYER_HANDLE_T handle, const char *data_  
→source);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
data_source	数据源	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: 目录结构所有头文件
- 库文件: libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.6 CVI_PLAYER_GetDataSource

【描述】

获取数据源

【语法】

```
int32_t CVI_PLAYER_GetDataSource(CVI_PLAYER_HANDLE_T handle, char *data_source);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
data_source	数据源	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.7 CVI_PLAYER_LightOpen

【描述】

打开解复用器

【语法】

```
int32_t CVI_PLAYER_LightOpen(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.8 CVI_PLAYER_Play

【描述】

播放

【语法】

```
int32_t CVI_PLAYER_Play(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.9 CVI_PLAYER_Stop

【描述】

停止播放器

【语法】

```
int32_t CVI_PLAYER_Stop(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.10 CVI_PLAYER_Pause

【描述】

暂停播放器

【语法】

```
int32_t CVI_PLAYER_Pause(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.11 CVI_PLAYER_Resume

【描述】

恢复播放

【语法】

```
int32_t CVI_PLAYER_Resume(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件

- 库文件: libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.12 CVI_PLAYER_Seek

【描述】

跳转

【语法】

```
int32_t CVI_PLAYER_Seek(CVI_PLAYER_HANDLE_T handle, int64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
time_in_ms	时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: 目录结构所有头文件
- 库文件: libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.13 CVI_PLAYER_TPlay

【描述】

快进

【语法】

```
int32_t CVI_PLAYER_TPlay(CVI_PLAYER_HANDLE_T handle, double speed);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
speed	快进速度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.14 CVI_PLAYER_SetAudioParameters

【描述】

设置音频参数

【语法】

```
int32_t CVI_PLAYER_SetAudioParameters(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪AUDIO_PARAMETERS parameters);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
parameters	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.15 CVI_PLAYER_SetVideoParameters

【描述】

设置视频参数

【语法】

```
int32_t CVI_PLAYER_SetVideoParameters(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪ VIDEO_PARAMETERS parameters);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
parameters	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.16 CVI_PLAYER_SetAOHandler

【描述】

设置音频输出句柄

【语法】

```
int32_t CVI_PLAYER_SetAOHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪OUTPUT_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	音频输出句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.17 CVI_PLAYER_SetCustomArgAOHandler

【描述】

自定义设置音频输出句柄

【语法】

```
int32_t CVI_PLAYER_SetCustomArgAOHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_CUSTOM_ARG_OUTPUT_HANDLER handler, void *custom_arg);;
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	音频输出句柄	输入
custom_arg	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.18 CVI_PLAYER_SetVOHandler

【描述】

设置视频输出句柄

【语法】

```
int32_t CVI_PLAYER_SetVOHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_OUTPUT_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	视频输出句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.19 CVI_PLAYER_SetCustomArgVOHandler

【描述】

自定义设置视频输出句柄

【语法】

```
int32_t CVI_PLAYER_SetCustomArgVOHandler(CVI_PLAYER_HANDLE_T handle, CVI_
↪PLAYER_CUSTOM_ARG_OUTPUT_HANDLER handler, void *custom_arg);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	音频输出句柄	输入
custom_arg	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.20 CVI_PLAYER_SetEventHandler

【描述】

设置播放事件句柄

【语法】

```
int32_t CVI_PLAYER_SetEventHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪EVENT_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	输出句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.21 CVI_PLAYER_SetCustomArgEventHandler

【描述】

自定义设置播放事件句柄

【语法】

```
int32_t CVI_PLAYER_SetCustomArgEventHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_CUSTOM_ARG_EVENT_HANDLER handler, void *custom_arg);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	输出句柄	输入
custom_arg	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.22 CVI_PLAYER_SaveImage

【描述】

保存软解码后的视频帧

【语法】

```
int32_t CVI_PLAYER_SaveImage(CVI_PLAYER_HANDLE_T handle, const char *file_path);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
file_path	文件路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.23 CVI_PLAYER_GetMediaInfo

【描述】

获取媒体信息

【语法】

```
int32_t CVI_PLAYER_GetMediaInfo(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪MEDIA_INFO *info);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
info	媒体信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.24 CVI_PLAYER_GetPlayInfo

【描述】

获取播放器信息

【语法】

```
int32_t CVI_PLAYER_GetPlayInfo(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_PLAY_
↪INFO *info);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
info	播放器信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.25 CVI_PLAYER_GetVideoFrame

【描述】

获取视频帧

【语法】

```
int32_t CVI_PLAYER_GetVideoFrame(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↳FRAME *frame);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
frame	视频帧	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.26 CVI_PLAYER_GetVideoPacket

【描述】

获取视频包

【语法】

```
int32_t CVI_PLAYER_GetVideoPacket(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↳PACKET_S *packet);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
packet	视频包	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.27 CVI_PLAYER_GetVideoExtraPacket**【描述】**

获取视频额外包

【语法】

```
int32_t CVI_PLAYER_GetVideoExtraPacket(CVI_PLAYER_HANDLE_T handle, CVI_
⇨PLAYER_PACKET *packet);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
packet	视频包	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.28 CVI_PLAYER_SetVideoDecodeHandler

【描述】

设置视频解码句柄

【语法】

```
int32_t CVI_PLAYER_SetVideoDecodeHandler(CVI_PLAYER_HANDLE_T handle, CVI_
↳PLAYER_DECODE_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	解码句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.29 CVI_PLAYER_SetVideoCustomArgDecodeHandler

【描述】

自定义设置视频解码句柄

【语法】

```
int32_t CVI_PLAYER_SetVideoCustomArgDecodeHandler(CVI_PLAYER_HANDLE_T handle,
↪ CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER handler, void *custom_arg);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	解码句柄	输入
custom_arg	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.30 CVI_PLAYER_SetiAudioDecodeHandler

【描述】

设置音频解码句柄

【语法】

```
int32_t CVI_PLAYER_SetiAudioDecodeHandler(CVI_PLAYER_HANDLE_T handle, CVI_
↪PLAYER_DECODE_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	解码句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.31 CVI_PLAYER_SetAudioCustomArgDecodeHandler

【描述】

打开网络视频流

【语法】

```
int32_t CVI_PLAYER_SetAudioCustomArgDecodeHandler(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER_S handler, void *custom_arg);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
handler	解码句柄	输入
custom_arg	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.32 CVI_PLAYER_PacketContainSps

【描述】

包是否包含 sps

【语法】

```
bool CVI_PLAYER_PacketContainSps(CVI_PLAYER_HANDLE_T handle, CVI_PLAYER_
↪PACKET_S *packet);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
packet	视频包	输入

【返回值】

返回值	描述
true	成功
false	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.33 CVI_PLAYER_SeekPause

【描述】

在暂停状态下跳转

【语法】

```
int32_t CVI_PLAYER_SeekPause(CVI_PLAYER_HANDLE_T handle, int64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
time_in_ms	时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.34 CVI_PLAYER_SeekFlage

【描述】

获取 seek 标志

【语法】

```
int32_t CVI_PLAYER_SeekFlage();
```

【参数】

无

【返回值】

返回值	描述
非 0	标志

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.35 CVI_PLAYER_SeekTime

【描述】

跳转到某个时间

【语法】

```
int32_t CVI_PLAYER_SeekTime();
```

【参数】

无

【返回值】

返回值	描述
非 0	当前 seek 时间

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.36 CVI_PLAYER_PlayerSeep

【描述】

快进快退倍数播放

【语法】

```
int32_t CVI_PLAYER_PlayerSeep(CVI_PLAYER_HANDLE_T handle, int32_t speed, int32_t  
↪ backforward);
```

【参数】

参数名称	描述	输入/输出
handle	播放句柄	输入
speed	速度	输入
backforward	快退	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.2.37 CVI_PLAYER_GetForWardBackWardStatus

【描述】

获取倍数播放状态

【语法】

```
int32_t CVI_PLAYER_GetForWardBackWardStatus(CVI_PLAYER_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	复用器属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：目录结构所有头文件
- 库文件：libcvi_player.a/libcvi_player.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.3.3 数据类型

相关数据类型定义如下：

- `CVI_PLAYER_PACKET_S`：定义播放包。
- `CVI_PLAYER_MEDIA_INFO_S`：定义播放媒体信息。
- `CVI_PLAYER_HANDLE_T`：定义播放句柄。
- `CVI_PLAYER_OUTPUT_HANDLER`：定义播放输出处理函数。
- `CVI_PLAYER_CUSTOM_ARG_OUTPUT_HANDLER`：扩展自定义播放输出处理函数。
- `CVI_PLAYER_EVENT_HANDLER`：定义播放事件处理函数。
- `CVI_PLAYER_CUSTOM_ARG_EVENT_HANDLER`：扩展自定义播放事件处理函数。
- `CVI_PLAYER_DECODE_HANDLER_S`：定义播放解码处理函数结构体。
- `CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER_S`：扩展自定义播放解码处理函数结构体。
- `CVI_PLAYER_FRAME_S`：定义播放帧信息
- `CVI_PLAYER_EVENT_TYPE_E`：枚举播放事件类型
- `CVI_PLAYER_EVENT_S`：定义播放事件结构体

7.4.3.3.1 CVI_PLAYER_EVENT_TYPE_E

【说明】

枚举播放事件类型

【定义】

```
typedef enum CviPlayerEventType
{
    CVI_PLAYER_EVENT_OPEN_FAILED,
    CVI_PLAYER_EVENT_PLAY,
    CVI_PLAYER_EVENT_PLAY_FINISHED,
    CVI_PLAYER_EVENT_PLAY_PROGRESS,
    CVI_PLAYER_EVENT_PAUSE,
    CVI_PLAYER_EVENT_RESUME
} CVI_PLAYER_EVENT_TYPE_E;
```

【成员】

成员名称	描述
CVI_PLAYER_EVENT_OPEN_FAILED	播放打开失败
CVI_PLAYER_EVENT_PLAY	开始播放
CVI_PLAYER_EVENT_PLAY_FINISHED	播放完成
CVI_PLAYER_EVENT_PLAY_PROGRESS	播放进行中
CVI_PLAYER_EVENT_PAUSE	播放暂停
CVI_PLAYER_EVENT_RESUME	播放恢复

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.2 CVI_PLAYER_EVENT_S

【说明】

定义播放事件结构体。

【定义】

```
typedef struct CviPlayerEvent
{
    CVI_PLAYER_EVENT_TYPE_E type;
} CVI_PLAYER_EVENT_S;
```

【成员】

成员名称	描述
type	播放事件类型

【注意事项】

无。

7.4.3.3 CVI_PLAYER_PACKET_S**【说明】**

定义播放包。

【定义】

```
typedef CVI_DEMUXER_PACKET_S CVI_PLAYER_PACKET_S;
```

【成员】

参考 CVI_DEMUXER_PACKET_S

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.4 CVI_PLAYER_MEDIA_INFO_S**【说明】**

定义播放媒体信息。

【定义】

```
typedef CVI_DEMUXER_MEDIA_INFO_S CVI_PLAYER_MEDIA_INFO_S;
```

【成员】

参考 CVI_DEMUXER_MEDIA_INFO_S

【注意事项】

无。

7.4.3.3.5 CVI_PLAYER_HANDLE_T**【说明】**

定义播放句柄。

【定义】

```
typedef void* CVI_PLAYER_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.6 CVI_PLAYER_FRAME_S

【说明】

播放帧信息

【定义】

```
typedef struct CviPlayerFrame
{
    uint8_t **data; // pointer to AVFrame data
    int32_t *linesize; // pointer to AVFrame linesize
    int32_t width;
    int32_t height;
    int32_t packet_size;
    int64_t pts;
} CVI_PLAYER_FRAME_S;
```

【成员】

成员名称	描述
data	帧数据
linesize	frame linesize 值，对齐后
width	帧宽
height	帧高
packet_size	包大小
pts	pts 时间

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.7 CVI_PLAYER_OUTPUT_HANDLER

【说明】

定义播放输出处理函数。

【定义】

```
typedef void (*CVI_PLAYER_OUTPUT_HANDLER)(CVI_PLAYER_FRAME_S *);
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.8 CVI_PLAYER_CUSTOM_ARG_OUTPUT_HANDLER

【说明】

扩展自定义播放输出处理函数。

【定义】

```
typedef void (*CVI_PLAYER_CUSTOM_ARG_OUTPUT_HANDLER)(void *,  
CVI_PLAYER_FRAME_S *);
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.9 CVI_PLAYER_EVENT_HANDLER

【说明】

定义播放事件处理函数。

【定义】

```
typedef void (*CVI_PLAYER_EVENT_HANDLER)(CVI_PLAYER_EVENT_S *);
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.10 CVI_PLAYER_CUSTOM_ARG_EVENT_HANDLER**【说明】**

扩展自定义播放事件处理函数。

【定义】

```
typedef void (*CVI_PLAYER_CUSTOM_ARG_EVENT_HANDLER)(void *,
    CVI_PLAYER_EVENT_S *);
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.11 CVI_PLAYER_DECODE_HANDLER_S**【说明】**

定义播放解码处理函数结构体。

【定义】

```
typedef struct {
    int32_t (*get_frame)(CVI_PLAYER_FRAME_S *);
    int32_t (*decode_packet)(CVI_PLAYER_PACKET_S *);
} CVI_PLAYER_DECODE_HANDLER_S;
```

【成员】

成员名称	描述
get_frame	获取 frame 回调函数
decode_packet	解码回调函数

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.3.3.12 CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER_S**【说明】**

扩展自定义播放解码处理函数结构体。

【定义】

```
typedef struct {  
    int32_t (*get_frame)(void *, CVI_PLAYER_FRAME_S *);  
    int32_t (*decode_packet)(void *, CVI_PLAYER_PACKET_S *);  
} CVI_PLAYER_CUSTOM_ARG_DECODE_HANDLER_S;
```

【成员】

成员名称	描述
get_frame	获取 frame 回调函数
decode_packet	解码回调函数

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4 Playback_Service

该组件是在 player 组件的基础上封装一层，其专门用于对接 app 层。

7.4.4.1 API 参考

该组件提供以下 API：

- CVI_PLAYER_SERVICE_GetDefaultParam：获取回放服务参数
- CVI_PLAYER_SERVICE_Create：创建回放服务
- CVI_PLAYER_SERVICE_Destroy：销毁回放服务
- CVI_PLAYER_SERVICE_SetInput：设置输入
- CVI_PLAYER_SERVICE_GetMediaInfo：获取媒体信息。
- CVI_PLAYER_SERVICE_GetPlayInfo：获取播放信息
- CVI_PLAYER_SERVICE_Play：播放
- CVI_PLAYER_SERVICE_PlayerAndSeek：直接跳到某个时间点开始播放

- CVI_PLAYER_SERVICE_Stop : 停止播放
- CVI_PLAYER_SERVICE_Pause : 暂停播放
- CVI_PLAYER_SERVICE_Seek : 跳转操作
- CVI_PLAYER_SERVICE_Resize : 调整输出大小
- CVI_PLAYER_SERVICE_MoveTo : 设置输出位置
- CVI_PLAYER_SERVICE_ToggleFullscreen : 切换全屏
- CVI_PLAYER_SERVICE_GetSignals : 获取信号
- CVI_PLAYER_SERVICE_GetSlots : 获取信号槽
- CVI_PLAYER_SERVICE_SeekTime : 获取跳转时间
- CVI_PLAYER_SERVICE_SeekFlage : 获取跳转标志
- CVI_PLAYER_SERVICE_SeekPause : 在暂停的基础上进行跳转
- CVI_PLAYER_SERVICE_TouchSeekPause : 跳到某个时间点暂停
- CVI_PLAYER_SERVICE_PlayerSeep : 快进
- CVI_PLAYER_SERVICE_PlayerSeepBack : 快退
- CVI_PLAYER_SERVICE_GetFileMediaInfo : 获取文件媒体信息
- CVI_PLAYER_SERVICE_SetEventHandler : 设置回放服务事件处理句柄

7.4.4.1.1 CVI_PLAYER_SERVICE_GetDefaultParam

【描述】

获取回放服务参数

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetDefaultParam(CVI_PLAYER_SERVICE_PARAM_S_
↪*param);
```

【参数】

参数名称	描述	输入/输出
param	回放服务参数	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.2 CVI_PLAYER_SERVICE_Create

【描述】

创建回放服务

【语法】

```
int32_t CVI_PLAYER_SERVICE_Create(CVI_PLAYER_SERVICE_HANDLE_T *handle, CVI_PLAYER_SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
param	回放服务参数	输入
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.3 CVI_PLAYER_SERVICE_Destroy

【描述】

销毁回放服务

【语法】

```
int32_t CVI_PLAYER_SERVICE_Destroy(CVI_PLAYER_SERVICE_HANDLE_T *handle);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.4 CVI_PLAYER_SERVICE_SetInput

【描述】

设置输入

【语法】

```
int32_t CVI_PLAYER_SERVICE_SetInput(CVI_PLAYER_SERVICE_HANDLE_T handle, const_↪char *input);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
input	输入	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.5 CVI_PLAYER_SERVICE_GetMediaInfo

【描述】

获取媒体信息

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetMediaInfo(CVI_PLAYER_SERVICE_HANDLE_T handle,  
→CVI_PLAYER_MEDIA_INFO *info);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
info	媒体信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.6 CVI_PLAYER_SERVICE_GetPlayInfo

【描述】

获取播放信息

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetPlayInfo(CVI_PLAYER_SERVICE_HANDLE_T handle, CVI_PLAYER_PLAY_INFO *info);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
info	媒体信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.7 CVI_PLAYER_SERVICE_Play

【描述】

播放

【语法】

```
int32_t CVI_PLAYER_SERVICE_Play(CVI_PLAYER_SERVICE_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.8 CVI_PLAYER_SERVICE_PlayerAndSeek

【描述】

获取回放服务参数

【语法】

```
int32_t CVI_PLAYER_SERVICE_PlayerAndSeek(CVI_PLAYER_SERVICE_HANDLE_T handle,  
→int32_t t64_seektime);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
seektime	跳转时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.9 CVI_PLAYER_SERVICE_Stop**【描述】**

停止播放

【语法】

```
int32_t CVI_PLAYER_SERVICE_Stop(CVI_PLAYER_SERVICE_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.10 CVI_PLAYER_SERVICE_Pause

【描述】

暂停播放

【语法】

```
int32_t CVI_PLAYER_SERVICE_Pause(CVI_PLAYER_SERVICE_HANDLE_T handle);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.11 CVI_PLAYER_SERVICE_Seek

【描述】

跳转操作

【语法】

```
int32_t CVI_PLAYER_SERVICE_Seek(CVI_PLAYER_SERVICE_HANDLE_T handle, int32_t  
↪ t64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
time_in_ms	跳转时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.12 CVI_PLAYER_SERVICE_SetEventHandler

【描述】

设置回放服务事件处理句柄

【语法】

```
int32_t CVI_PLAYER_SERVICE_SetEventHandler(CVI_PLAYER_SERVICE_HANDLE_T handle, CVI_PLAYER_SERVICE_EVENT_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
handler	事件处理句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.13 CVI_PLAYER_SERVICE_Resize

【描述】

调整输出大小

【语法】

```
int32_t CVI_PLAYER_SERVICE_Resize(CVI_PLAYER_SERVICE_HANDLE_T handle, uint32_t width, uint32_t height);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
width	宽度	输入
height	高度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.14 CVI_PLAYER_SERVICE_MoveTo

【描述】

设置输出位置

【语法】

```
int32_t CVI_PLAYER_SERVICE_MoveTo(CVI_PLAYER_SERVICE_HANDLE_T handle,  
↪ uint32_t x, uint32_t y);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
x	x 坐标	输入
y	y 坐标	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.15 CVI_PLAYER_SERVICE_ToggleFullscreen

【描述】

切换全屏

【语法】

```
int32_t CVI_PLAYER_SERVICE_ToggleFullscreen(CVI_PLAYER_SERVICE_HANDLE_T  
↪ handle);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.16 CVI_PLAYER_SERVICE_GetSignals

【描述】

获取信号

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetSignals(CVI_PLAYER_SERVICE_HANDLE_T handle,  
↪ CVI_PLAYER_SERVICE_SIGNALS **signals);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
signals	信号	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.17 CVI_PLAYER_SERVICE_GetSlots

【描述】

获取信号槽

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetSlots(CVI_PLAYER_SERVICE_HANDLE_T handle, CVI_PLAYER_SERVICE_SLOTS **slots);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
slots	信号槽	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.18 CVI_PLAYER_SERVICE_SeekTime

【描述】

获取跳转时间

【语法】

```
int32_t CVI_PLAYER_SERVICE_SeekTime();
```

【参数】

无

【返回值】

返回值	描述
非 0	跳转时间

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.19 CVI_PLAYER_SERVICE_SeekFlage

【描述】

获取跳转标志

【语法】

```
int32_t CVI_PLAYER_SERVICE_SeekFlage();
```

【参数】

无

【返回值】

返回值	描述
	跳转标志

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.20 CVI_PLAYER_SERVICE_SeekPause

【描述】

在播放暂停状态下跳转

【语法】

```
int32_t CVI_PLAYER_SERVICE_SeekPause(CVI_PLAYER_SERVICE_HANDLE_T handle,
↪int32_t64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
time_in_ms	跳转时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.21 CVI_PLAYER_SERVICE_TouchSeekPause

【描述】

跳到某个时间点暂停

【语法】

```
int32_t CVI_PLAYER_SERVICE_TouchSeekPause(CVI_PLAYER_SERVICE_HANDLE_T  
↪handle, int32_t64_t time_in_ms);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
time_in_ms	跳转时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.22 CVI_PLAYER_SERVICE_PlayerSeep

【描述】

快进

【语法】

```
int32_t CVI_PLAYER_SERVICE_PlayerSeep(CVI_PLAYER_SERVICE_HANDLE_T handle,  
↪int32_t speeds);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
speeds	速度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_player_service.h
- 库文件: libcvi_player_service.a/libcvi_player_service.so

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.23 CVI_PLAYER_SERVICE_PlayerSeepBack

【描述】

快退

【语法】

```
int32_t CVI_PLAYER_SERVICE_PlayerSeepBack(CVI_PLAYER_SERVICE_HANDLE_T handle,  
↔ int32_t speeds);
```

【参数】

参数名称	描述	输入/输出
handle	回放服务句柄	输入
speeds	速度	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.1.24 CVI_PLAYER_SERVICE_GetFileMediaInfo

【描述】

获取文件媒体信息

【语法】

```
int32_t CVI_PLAYER_SERVICE_GetFileMediaInfo(char *filepatch);
```

【参数】

参数名称	描述	输入/输出
filepatch	文件路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_player_service.h`
- 库文件: `libcvi_player_service.a/libcvi_player_service.so`

【注意】

无

【举例】

无

【相关主题】

无

7.4.4.2 数据类型

相关数据类型定义如下:

- CVI_PLAYER_SERVICE_PARAM_S : 定义回放服务参数
- CVI_PLAYER_SERVICE_EVENT_TYPE_E : 回放服务事件枚举
- CVI_PLAYER_SERVICE_EVENT_S : 定义回放服务事件
- CVI_PLAYER_SERVICE_EVENT_HANDLER : 定义回放服务事件处理函数
- CVI_PLAYER_SERVICE_SIGNALS_S : 定义回放服务信号结构体
- CVI_PLAYER_SERVICE_SLOTS_S : 定义回放服务信号槽结构体
- CVI_PLAYER_SERVICE_HANDLE_T : 定义回放服务句柄

7.4.4.2.1 CVI_PLAYER_SERVICE_PARAM_S

【说明】

定义回放服务参数

【定义】

```
typedef struct {  
    int32_t chn_id;  
    bool repeat;  
    // handle  
    CVI_MAPI_DISP_HANDLE_T disp;  
    CVI_MAPI_AO_HANDLE_T ao;  
    // display  
    int32_t disp_id;  
    PIXEL_FORMAT_E disp_fmt;  
    ROTATION_E disp_rotate;  
    ASPECT_RATIO_E disp_aspect_ratio;  
    uint32_t x;  
    uint32_t y;  
    uint32_t width; // 0 for screen width  
    uint32_t height; // 0 for screen height  
} CVI_PLAYER_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
chn_id	通道
repeat	重复使能
disp	视频输出句柄
ao	音频输出句柄
disp_id	展示 id
disp_fmt	像素格式
disp_aspect_ratio	宽高比
x	起点 x 坐标
y	起点 y 坐标
width	宽
height	高

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.2 CVI_PLAYER_SERVICE_EVENT_TYPE_E

【说明】

回放服务事件枚举

【定义】

```
typedef enum
{
    CVI_PLAYER_SERVICE_EVENT_UNKNOWN,
    CVI_PLAYER_SERVICE_EVENT_OPEN_FAILED,
    CVI_PLAYER_SERVICE_EVENT_PLAY,
    CVI_PLAYER_SERVICE_EVENT_PLAY_FINISHED,
    CVI_PLAYER_SERVICE_EVENT_PLAY_PROGRESS,
    CVI_PLAYER_SERVICE_EVENT_PAUSE,
    CVI_PLAYER_SERVICE_EVENT_RESUME,
    CVI_PLAYER_SERVICE_EVENT_RECOVER_START,
    CVI_PLAYER_SERVICE_EVENT_RECOVER_PROGRESS,
    CVI_PLAYER_SERVICE_EVENT_RECOVER_FAILED,
    CVI_PLAYER_SERVICE_EVENT_RECOVER_FINISHED,
} CVI_PLAYER_SERVICE_EVENT_TYPE_E;
```

【成员】

成员名称	描述
CVI_PLAYER_SERVICE_EVENT_UNKNOW	未知事件
CVI_PLAYER_SERVICE_EVENT_OPEN	打开文件失败
CVI_PLAYER_SERVICE_EVENT_PLAY	开始播放
CVI_PLAYER_SERVICE_EVENT_PLAY_STOP	播放结束
CVI_PLAYER_SERVICE_EVENT_PLAY_PAUSE	播放进行中
CVI_PLAYER_SERVICE_EVENT_PAUSE	暂停
CVI_PLAYER_SERVICE_EVENT_RESUME	暂停恢复
CVI_PLAYER_SERVICE_EVENT_RECOVER_BEGIN	文件修复开始
CVI_PLAYER_SERVICE_EVENT_RECOVER_OK	文件修复成功
CVI_PLAYER_SERVICE_EVENT_RECOVER_FAIL	文件修复失败
CVI_PLAYER_SERVICE_EVENT_RECOVER_FINISH	文件修复完成

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.3 CVI_PLAYER_SERVICE_EVENT_S

【说明】

定义回放服务事件

【定义】

```
typedef struct
{
    CVI_PLAYER_SERVICE_EVENT_TYPE_E type;
    double value;
} CVI_PLAYER_SERVICE_EVENT_S;
```

【成员】

成员名称	描述
type	回放服务事件类型
value	值

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.4 CVI_PLAYER_SERVICE_EVENT_HANDLER

【说明】

定义回放服务事件处理函数

【定义】

```
typedef void (*CVI_PLAYER_SERVICE_EVENT_HANDLER)(CVI_PLAYER_SERVICE_
↪HANDLE_T,CVI_PLAYER_SERVICE_EVENT_S *);
```

【成员】

成员名称	描述
CVI_PLAYER_SERVICE_HANDLE_T	回放服务句柄
CVI_PLAYER_SERVICE_EVENT_S	回放服务事件

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.5 CVI_PLAYER_SERVICE_SLOTS_S

【说明】

定义回放服务信号槽结构体

【定义】

```
typedef struct {
    CVI_SLOT_S set_input;
    CVI_SLOT_S play;
    CVI_SLOT_S stop;
    CVI_SLOT_S pause;
    CVI_SLOT_S seek;
    CVI_SLOT_S resize;
    CVI_SLOT_S toggle_fullscreen;
    CVI_SLOT_S get_play_info;
} CVI_PLAYER_SERVICE_SLOTS_S;
```

【成员】

成员名称	描述
set_input	设置输入信号槽
play	播放信号槽
stop	停止信号槽
pause	暂停信号槽
seek	跳转信号槽
resize	跳转输出大小信号槽
toggle_fullscreen	全屏信号槽
get_play_info	获取播放信息信号槽

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.6 CVI_PLAYER_SERVICE_SIGNALS_S

【说明】

定义回放服务信号结构体

【定义】

```
typedef struct {  
    CVI_SIGNAL_S play;  
    CVI_SIGNAL_S pause;  
    CVI_SIGNAL_S resume;  
    CVI_SIGNAL_S finish;  
} CVI_PLAYER_SERVICE_SIGNALS_S;
```

【成员】

成员名称	描述
play	播放信号
pause	暂停信号
resume	恢复信号
finish	结束信号

【注意事项】

无。

【相关数据类型及接口】

无。

7.4.4.2.7 CVI_PLAYER_SERVICE_HANDLE_T

【说明】

定义回放服务句柄。

【定义】

```
typedef void* CVI_PLAYER_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

8 文件修复

8.1 概述

文件修复模块主要的功能是当录制媒体文件时异常断档，例如断电、sd 卡异常拔出，此时 MOV/MP4 封装格式未完全书写完毕，因此文件处于一个坏档状态，即播放器也无法正常播放该文件。当再次挂载 sd 卡时，文件修复模块自动检测文档，会对坏档文档进行修复。整体的流程如图 8-1 所示：

- 当前文件修复支持两种录像音视频搭配方式，分别为 MOV 封装（mov+pcm）和 MP4（mp4+aac）。
- 现有的媒体文件生成方式是，创建媒体文件，媒体文件在写入过程中，由于存放媒体流信息的索引部分（moov）与存放媒体流数据的部分（mdat）的大小都会随着媒体流数据的增加而增加，所以现有媒体库在写入媒体文件时，都会将 moov 缓存在内存中。只将媒体流数据 mdat 的部分写入硬盘，等所有数据写入完毕后，再将文件头部写入硬盘。也就是在 mdat 持续写入的同时，发生断电，此时 moov 并未写入到 mdat 的尾部。而播放文件需要从 moov 中读取 mdat 的媒体数据。坏档文件与正常文件对比图如图 8-2 所示



图 8.1: 文件修复流程图

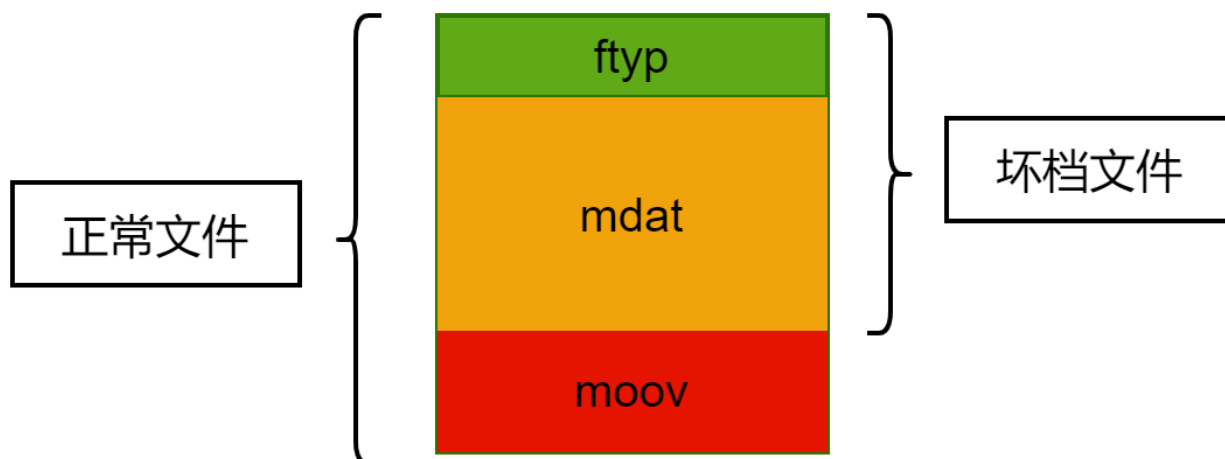


图 8.2: 正常 vs 坏档文件内容

注意:

- 建议先了解 mov 和 mp4 标准协议，再看 mov 及 mp4 内部封装做法，这样对文件修复 Moov 部分理解有很大帮助。

8.2 实现逻辑

修复模块是解析 mdat 数据，然后移植模板 bin 档的 moov box 至坏档文件的 mdat box 的尾部，最后用解析出来的数据更新 moov 各子 box 内容，坏档文件的修复过程如图 8-3 所示。

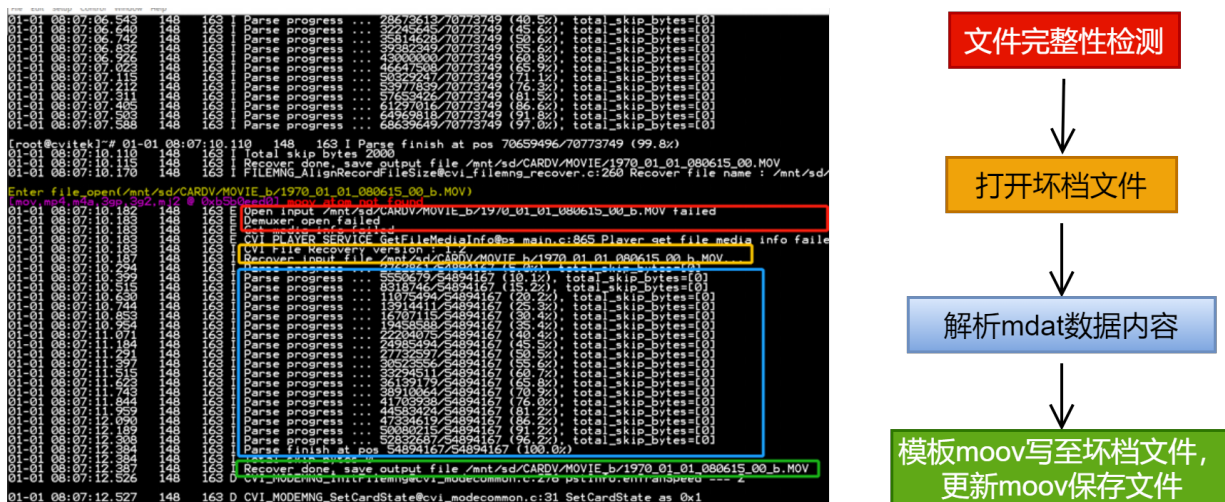


图 8.3: 坏档文件的修复过程

8.2.1 解析 mdat

如图 8-4 所示，mdat 主要包括缩略图包、字幕包、视频包和音频包数据信息。解析 mdat 就是将各个包解析出来，用于计算 moov 中的信息，例如计算文件时长、文件信息，宽高、帧率、通道数、采样率等信息。

- ThumbnailPacket: 当前判断是否为缩略图的依据为文件开头两个字节为 0xFF, 0xD8，找到头后依次读取数据，直到读取到 jpeg 尾部字段 0xFF, 0xD9，则认为这部分数据为缩略图数据，记录当前的包长度。
- AudioPacket (PCM): 当前判断是否为音频数据的依据为文件开头四个字节为 0xFF, 0xF1, 0x60, 0x40。pcm 的包长度为 1280 字节的整数倍。
- VideoPacket: 是由 startcode 和 NALUs (Network Abstraction Layer Unit) 组成，当前判断是否为视频数据的依据为是否有 NALU header。例如对于 H264 编码，forbidden bit 须为 0，Nalu Type 为 1/5/7/8。
- SubtitlePacket: 设置的 subtitlepacket 长度固定为 202Bytes。当第一个字节为 00 时，第二个字节数据为 C8 (200) 时，则认为此为字幕包，然后赋值固定长度为 202。此外对于 MP4 封装，还应判断第三，第四个字节为 0x23, 0x41 或 0x67, 0x73。

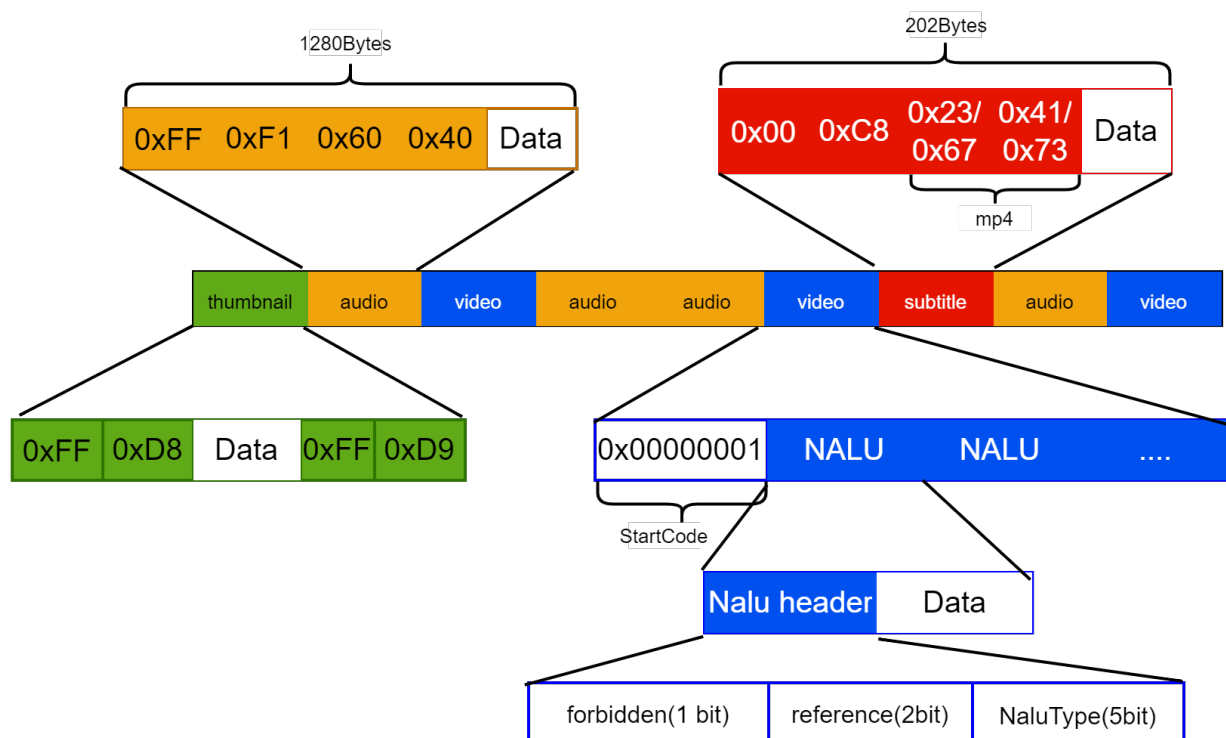


图 8.4: mdat 内容排布

8.2.2 更新 moov

Moov box 文件结构如图 8-5 所示，要更新的主要内容红色方框标记，总结如下：

- Packets offset(start position in file)
- Packets size
- Duration:
 - Video & Audio: Packet 数量 * 1/fps
 - Subtitle: Packet 数量 * 1s
 - 更新 Duration to mvhd(file)/tkhd(track)
- Video width/height
 - 从 Video Packet 里面找出 Sps，解析 Sps 取得影像宽高；更新宽高到 tkhd/stsd。

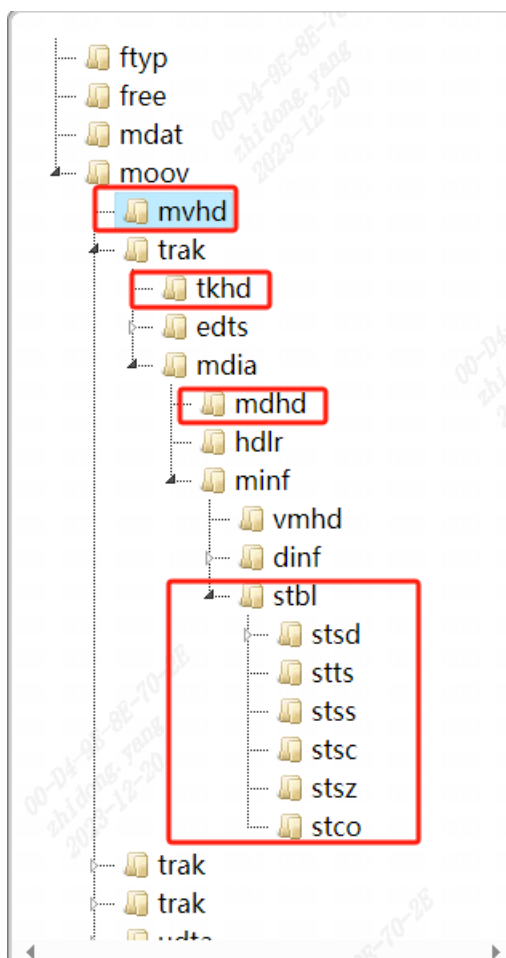


图 8.5: moov 文件结构

8.3 API 参考

这里介绍给予用户使用的 API。

8.3.1 目录结构

文件修复的内部实现是用 C++ 语言。为了方便理解，这里介绍 include 和 src 文件夹下各个子文件的作用以及各文件夹的逻辑调用关系。

```
/file_recover
├── /include
│   ├── event.h: 文件修复事件类型，例如修复成功和失败
│   └── cvi_file_rocover.h: 用户API
├── /src
│   ├── file.hpp: 定义文件类，包含文件打开，关闭和搜寻等成员函数
│   ├── event_handler.hpp: 定义事件处理类，用于处理文件修复事件类型 (event.
│   └── container.hpp: 定义文件容器基类
```

(下页继续)

(续上页)

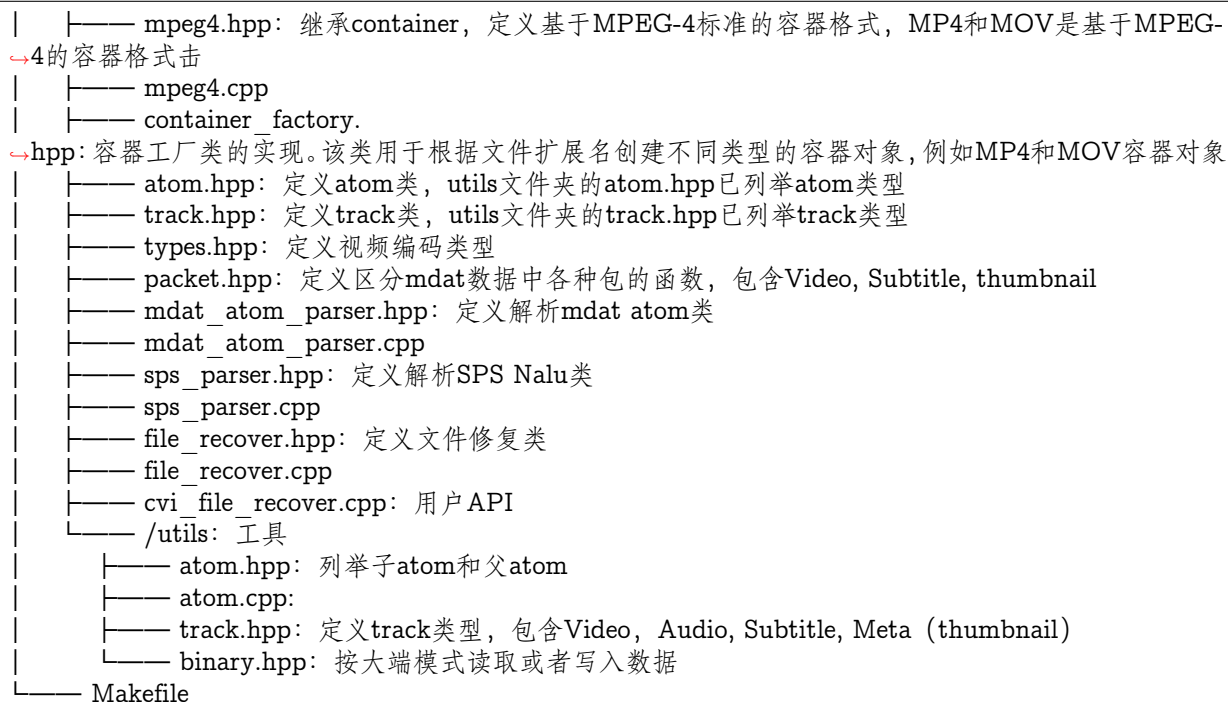


图 8-6 表述文件修复各头文件.hpp 之间逻辑调用关系。

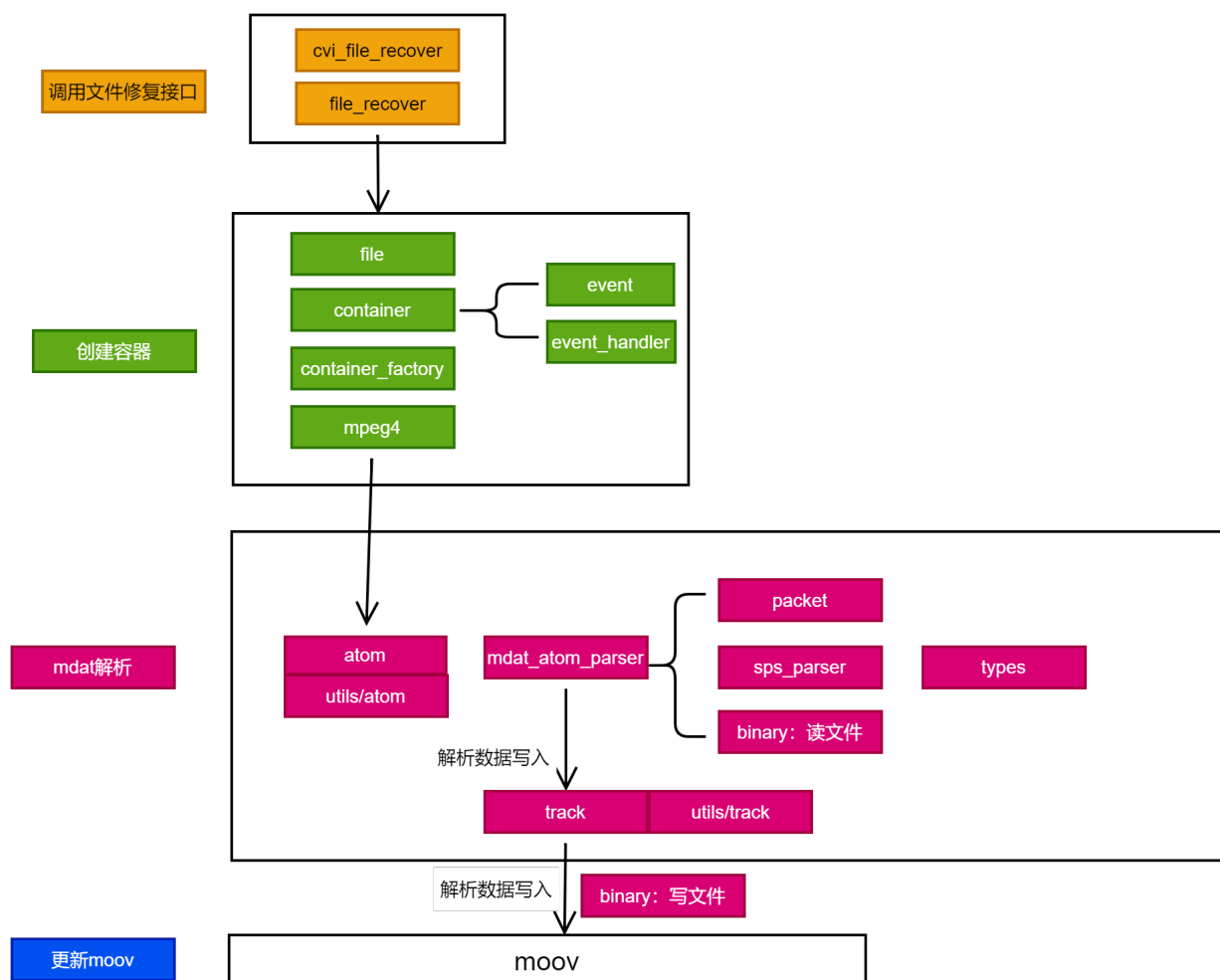


图 8.6: 文件修复各头文件.hpp 之间逻辑调用关系

8.3.2 API

该组件提供以下 API:

- CVI_FILE_RECOVER_Create : 创建文件修复
- CVI_FILE_RECOVER_Destroy : 销毁文件修复
- CVI_FILE_RECOVER_Open : 打开文件修复
- CVI_FILE_RECOVER_Check : 检查文件的完整性
- CVI_FILE_RECOVER_Dump : 以 atom tree 形式打印各 atom 大小
- CVI_FILE_RECOVER_Recover : 开启文件修复
- CVI_FILE_RECOVER_RecoverAsync : 创建线程, 异步开启文件修复
- CVI_FILE_RECOVER_RecoverJoin : 以阻塞的方式等待文件修复线程结束
- CVI_FILE_RECOVER_Close : 关闭文件修复
- CVI_FILE_RECOVER_SetEventHandler : 设置文件修复处理事件函数

- `CVI_FILE_RECOVER_SetCustomArgEventHandler`：自定义设置文件修复处理事件函数
- `CVI_FILE_RECOVER_PreallocateState`：设置预分配标志

8.3.2.1 `CVI_FILE_RECOVER_Create`

【描述】

创建文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_Create(CVI_FILE_RECOVER_HANDLE *handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：`cvi_file_recover.h/event.h`
- 库文件：`libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.2 CVI_FILE_RECOVER_Destroy

【描述】

销毁文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_Destroy(CVI_FILE_RECOVER_HANDLE *handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_file_recover.h/event.h
- 库文件: libcvi_file_recover.a/libcvi_file_recover.so

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.3 CVI_FILE_RECOVER_Open

【描述】

打开文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_Open(CVI_FILE_RECOVER_HANDLE handle, const char* input_↵file_path);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
input_file_path	文件路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.4 CVI_FILE_RECOVER_Check

【描述】

检查文件的完整性

【语法】

```
int32_t CVI_FILE_RECOVER_Check(CVI_FILE_RECOVER_HANDLE handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.5 CVI_FILE_RECOVER_Dump

【描述】

以 atom tree 形式打印各 atom 大小

【语法】

```
int32_t CVI_FILE_RECOVER_Dump(CVI_FILE_RECOVER_HANDLE handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_file_recover.h/event.h
- 库文件: libcvi_file_recover.a/libcvi_file_recover.so

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.6 CVI_FILE_RECOVER_Recover

【描述】

开始文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_Recover(CVI_FILE_RECOVER_HANDLE handle, const char* _  
↪ output_file_path, const char* device_model, bool has_create_time);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
output_file_path	输出路径	输入
device_model	设备模型	输入
has_create_time	是否有创建时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvl_file_recover.h/event.h
- 库文件: libcvl_file_recover.a/libcvl_file_recover.so

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.7 CVI_FILE_RECOVER_RecoverAsync

【描述】

异步开启文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_RecoverAsync(CVI_FILE_RECOVER_HANDLE handle, const_
↪char* output_file_path, const char* device_model, bool has_create_time);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
output_file_path	输出路径	输入
device_model	设备模型	输入
has_create_time	是否有创建时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.8 CVI_FILE_RECOVER_RecoverJoin

【描述】

以阻塞的方式等待文件修复线程结束

【语法】

```
int32_t CVI_FILE_RECOVER_RecoverJoin(CVI_FILE_RECOVER_HANDLE handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.9 CVI_FILE_RECOVER_Close**【描述】**

关闭文件修复

【语法】

```
int32_t CVI_FILE_RECOVER_Close(CVI_FILE_RECOVER_HANDLE handle);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_file_recover.h/event.h
- 库文件: libcvi_file_recover.a/libcvi_file_recover.so

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.10 CVI_FILE_RECOVER_SetEventHandler**【描述】**

设置文件修复处理事件函数

【语法】

```
int32_t CVI_FILE_RECOVER_SetEventHandler(CVI_FILE_RECOVER_HANDLE handle,CVI_
↪FILE_RECOVER_EVENT_HANDLER handler);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
handler	文件修复处理事件函数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.11 CVI_FILE_RECOVER_SetCustomArgEventHandler

【描述】

自定义设置文件修复处理事件函数

【语法】

```
int32_t CVI_FILE_RECOVER_SetCustomArgEventHandler(CVI_FILE_RECOVER_HANDLE_
↪ handle, CVI_FILE_RECOVER_CUSTOM_ARG_EVENT_HANDLER handler, void* custom_
↪ arg);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
handler	文件修复处理事件函数	输入
custom_arg	函数参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.2.12 CVI_FILE_RECOVER_PreallocateState

【描述】

设置预分配标志

【语法】

```
void CVI_FILE_RECOVER_PreallocateState(CVI_FILE_RECOVER_HANDLE handle, bool_
↪PreallocFlage);
```

【参数】

参数名称	描述	输入/输出
handle	文件修复句柄	输入
PreallocFlage	预分配标志	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_file_recover.h/event.h`
- 库文件: `libcvi_file_recover.a/libcvi_file_recover.so`

【注意】

无

【举例】

无

【相关主题】

无

8.3.3 数据结构

相关数据类型定义如下:

- `CVI_FILE_RECOVER_EVENT_HANDLER`: 文件修复事件处理函数
- `CVI_FILE_RECOVER_CUSTOM_ARG_EVENT_HANDLER`: 自定义文件修复事件处理函数
- `CVI_FILE_RECOVER_EVENT_TYPE_E`: 文件修复事件
- `CVI_FILE_RECOVER_EVENT_S`: 文件修复事件结构体
- `CVI_FILE_RECOVER_HANDLE_T`: 文件修复句柄

8.3.3.1 `CVI_FILE_RECOVER_HANDLE_T`

【说明】

文件修复句柄

【定义】

```
typedef void* CVI_FILE_RECOVER_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

8.3.3.2 `CVI_FILE_RECOVER_EVENT_TYPE_E`

【说明】

文件修复事件

【定义】

```
typedef enum CviFileRecoverEventType
{
    CVI_FILE_RECOVER_EVENT_OPEN_FAILED,
    CVI_FILE_RECOVER_EVENT_RECOVER_START,
    CVI_FILE_RECOVER_EVENT_RECOVER_PROGRESS,
    CVI_FILE_RECOVER_EVENT_RECOVER_FAILED,
    CVI_FILE_RECOVER_EVENT_RECOVER_FINISHED,
} CVI_FILE_RECOVER_EVENT_TYPE_E;
```

【成员】

成员名称	描述
CVI_FILE_RECOVER_EVENT_OPEN_FAILED	打开文件失败
CVI_FILE_RECOVER_EVENT_RECOVER_START	文件修复开始
CVI_FILE_RECOVER_EVENT_RECOVER_PROGRESS	文件修复进行中
CVI_FILE_RECOVER_EVENT_RECOVER_FAILED	文件修复失败
CVI_FILE_RECOVER_EVENT_RECOVER_COMPLETED	文件修复完成

【注意事项】

无。

【相关数据类型及接口】

无。

8.3.3.3 CVI_FILE_RECOVER_EVENT_S

【说明】

文件修复结构体

【定义】

```
typedef struct CviFileRecoverEvent
{
    CVI_FILE_RECOVER_EVENT_TYPE_E type;
    double value;
} CVI_FILE_RECOVER_EVENT_S;
```

【成员】

成员名称	描述
type	文件修复事件
value	值

【注意事项】

无。

【相关数据类型及接口】

无。

8.3.3.4 CVI_FILE_RECOVER_EVENT_HANDLER

【说明】

文件修复事件处理函数

【定义】

```
typedef void (*CVI_FILE_RECOVER_EVENT_HANDLER)(CVI_FILE_RECOVER_EVENT_S*);
```

【成员】

成员名称	描述
CVI_FILE_RECOVER_EVENT_S	文件修复事件类型

【注意事项】

无。

【相关数据类型及接口】

无。

8.3.3.5 CVI_FILE_RECOVER_CUSTOM_ARG_EVENT_HANDLER**【说明】**

自定义文件修复事件处理函数

【定义】

```
typedef void (*CVI_FILE_RECOVER_CUSTOM_ARG_EVENT_HANDLER)(void*, CVI_FILE_
↪RECOVER_EVENT_S*);
```

【成员】

成员名称	描述
CVI_FILE_RECOVER_EVENT_S	文件修复事件类型
void*	自定义参数

【注意事项】

无。

【相关数据类型及接口】

无。

9 服务组件

服务组件专门对接 APP 层，处理 APP 层传下的参数，位置如图 2-1 Framework 通用软件 所示。录像模块介绍了Record_Service 和Audio_Service，播放器模块介绍了Playback_Service，下面介绍以下服务组件：

- Rtsp_Service: rtsp 流媒体传输
- Storage_Service: 存储管理服务
- Photo_Service: 拍照服务
- Liveview_Service: 预览服务

注意：

- menuconfig 中 Services options 选项中提供了这些服务组件的开关选项。

9.1 Rtsp_Service

网络点播服务 Rtsp_Service 用于点播，运行于服务端，支持 RTSP 标准协议。

基本概念：

- RTSP (Real Time Streaming Protocol): 实时流协议，用于协议信令交互；与 HTTP 协议类似，是 TCP/IP 协议体系中的一个应用层协议
- RTP (Real-time Transport Protocol): 实时传输协议，RTP 协议详细说明了在互联网上传递音频和视频的标准数据包格式，它使用 TCP 或 UDP 完成数据传输，RTSP 在体系结构上位于 RTP 和 RTCP 之上，详情见图 9-1。

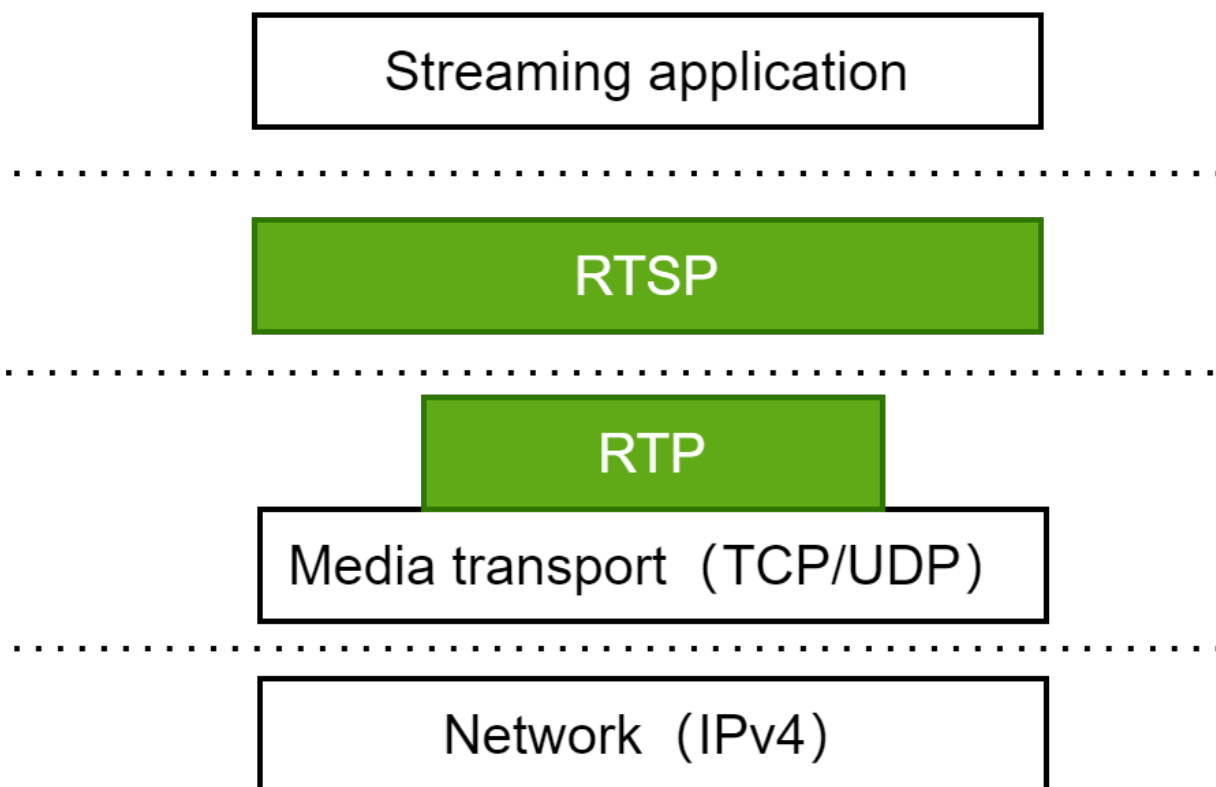


图 9.1: 流媒体框架网络体系

网络点播 Rtsp_Service 组件软件上下文依赖关系如图 9-2 所示。对下依赖媒体流源数据的获取，对上依赖 APP 层的调用。同时对于客户端，依赖支持标准 RTSP 协议的 RTSPClient，例如：VLC 点播客户端。

- 服务端监听客户端的请求，支持的请求包括：OPTION、DESCRIBE、SETUP、PLAY、GETPARAMETER、TEARDOWN。
- 当服务端收到客户端的请求后，根据请求的内容作出相应的回应。如果请求为 rtsp 点播，则将请求 URL 中指明的媒体数据按照一定的规则，并且根据客户端的请求解析出的打包方式通过 Socket 发送给客户端。例如 PLAY 请求，服务端会将媒体流按照 RTP 格式打包，然后发送到客户端，客户端接受数据并解析，然后在 VLC 播放。

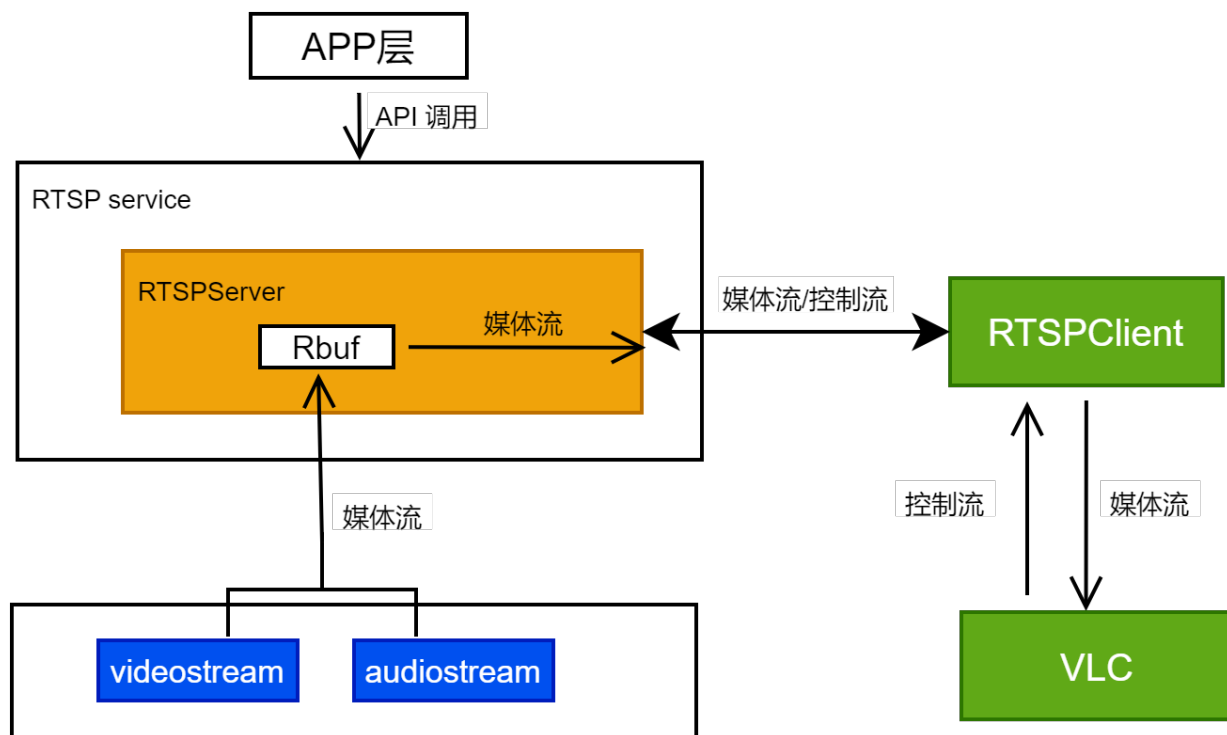


图 9.2: 网络点播 RTSPService 组件软件上下文依赖关系

9.1.1 API 参考

该组件提供以下 API:

- CVI_RTSP_SERVICE_Create : 创建 RTSP 服务
- CVI_RTSP_SERVICE_Destroy : 销毁 RTSP 服务
- CVI_RTSP_SERVICE_UpdateParam : 更新 RTSP 服务参数
- CVI_RTSP_SERVICE_StartMute : 静音
- CVI_RTSP_SERVICE_StopMute : 停止静音
- CVI_RTSP_SERVICE_StartStop : 开始或暂停视频传输

9.1.1.1 CVI_RTSP_SERVICE_Create

【描述】

创建 RTSP 服务

【语法】

```
int32_t CVI_RTSP_SERVICE_Create(CVI_RTSP_SERVICE_HANDLE_T *hdl, CVI_RTSP_SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
hdl	rtsp 服务句柄	输入
param	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvt_rtsp_service.h
- 库文件: libcvt_rtsp_service.a/libcvt_rtsp_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.1.1.2 CVI_RTSP_SERVICE_Destroy

【描述】

销毁 RTSP 服务

【语法】

```
int32_t CVI_RTSP_SERVICE_Destroy(CVI_RTSP_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	rtsp 服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_rtsp_service.h`
- 库文件: `libcvi_rtsp_service.a/libcvi_rtsp_service.so`

【注意】

无

【举例】

无

【相关主题】

无

9.1.1.3 CVI_RTSP_SERVICE_UpdateParam

【描述】

更新 RTSP 服务参数

【语法】

```
int32_t CVI_RTSP_SERVICE_UpdateParam(CVI_RTSP_SERVICE_HANDLE_T hdl, CVI_RTSP_SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
hdl	rtsp 服务句柄	输入
param	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_rtsp_service.h`
- 库文件: `libcvi_rtsp_service.a/libcvi_rtsp_service.so`

【注意】

无

【举例】

无

【相关主题】

无

9.1.1.4 CVI_RTSP_SERVICE_StartMute

【描述】

静音

【语法】

```
int32_t CVI_RTSP_SERVICE_StartMute(CVI_RTSP_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	rtsp 服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rtsp_service.h
- 库文件: libcvi_rtsp_service.a/libcvi_rtsp_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.1.1.5 CVI_RTSP_SERVICE_StopMute

【描述】

停止静音

【语法】

```
int32_t CVI_RTSP_SERVICE_StopMute(CVI_RTSP_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	rtsp 服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rtsp_service.h
- 库文件: libcvi_rtsp_service.a/libcvi_rtsp_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.1.1.6 CVI_RTSP_SERVICE_StartStop

【描述】

开始或暂停视频传输

【语法】

```
int32_t CVI_RTSP_SERVICE_StartStop(uint32_t value, char *name);
```

【参数】

参数名称	描述	输入/输出
value	使能	输入
name	rtsp 名字	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_rtsp_service.h
- 库文件: libcvi_rtsp_service.a/libcvi_rtsp_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.1.2 数据类型

相关数据类型定义如下:

- `CVI_RTSP_SERVICE_PARAM_S`: rtsp 服务参数
- `CVI_RTSP_SERVICE_VIDEO_CODEC_E`: 视频编码枚举
- `CVI_RTSP_SERVICE_AUDIO_CODEC_E`: 音频编码枚举
- `CVI_RTSP_SERVICE_CALLBACK`: rtsp 回调函数
- `CVI_RTSP_SERVICE_HANDLE_T`: rtsp 服务句柄
- `MAX_RTSP_STREAM_NAME_LEN`: 最长 rtsp 流名字

9.1.2.1 MAX_RTSP_STREAM_NAME_LEN

【说明】

最长 rtsp 流名字

【定义】

```
#define MAX_RTSP_STREAM_NAME_LEN (32);
```

【注意事项】

无。

【相关数据类型及接口】

无。

9.1.2.2 CVI_RTSP_SERVICE_PARAM_S

【说明】

rtsp 服务参数

【定义】

```
typedef struct {  
    int32_t recorder_id;  
    char rtsp_name[MAX_RTSP_STREAM_NAME_LEN];  
    int32_t max_conn;  
    int32_t timeout;  
    int32_t port;  
    CVI_RTSP_SERVICE_VIDEO_CODEC_E video_codec;  
    CVI_RTSP_SERVICE_AUDIO_CODEC_E audio_codec;
```

(下页继续)

(续上页)

```

CVI_RTSP_SERVICE_CALLBACK *rtsp_play;
void *rtsp_play_arg;
CVI_RTSP_SERVICE_CALLBACK *rtsp_teardown;
void *rtsp_teardown_arg;

uint32_t width;
uint32_t height;
float framerate;
int32_t bitrate_kbps;
int32_t audio_sample_rate;
int32_t audio_channels;
int32_t audio_pernum;

int32_t chn_id;
CVI_MAPI_VPROC_HANDLE_T vproc;
CVI_MAPI_VENC_HANDLE_T venc_hdl;
CVI_MAPI_ACAP_HANDLE_T acap_hdl;
CVI_MAPI_AENC_HANDLE_T aenc_hdl;
} CVI_RTSP_SERVICE_PARAM_S;

```

【成员】

成员名称	描述
recorder_id	录像 id
rtsp_name	rtsp 名字
max_conn	最大连接数
timeout	超时时间，单位为秒
port	端口号
video_codec	视频编码
audio_codec	音频编码
rtsp_play	rtsp 播放回调函数
rtsp_play_arg	rtsp 播放回调函数参数
rtsp_teardown	rtsp 断开连接回调函数
rtsp_teardown_arg	rtsp 断开连接回调函数参数
width	视频宽度
height	视频高度
framerate	帧率
bitrate_kbps	码率
audio_sample_rate	采样率
audio_channels	声道数
audio_pernum	每帧采样数
chn_id	通道 id
vproc	vpss 句柄
venc_hdl	视频编码句柄
acap_hdl	音频输入句柄
aenc_hdl	音频编码句柄

【注意事项】

无。

【相关数据类型及接口】

无。

9.1.2.3 CVI_RTSP_SERVICE_VIDEO_CODEC_E

【说明】

视频编码枚举

【定义】

```
typedef enum {
    CVI_RTSP_SERVICE_VIDEO_CODEC_H264 = 0,
    CVI_RTSP_SERVICE_VIDEO_CODEC_H265,
    CVI_RTSP_SERVICE_VIDEO_CODEC_JPEG
} CVI_RTSP_SERVICE_VIDEO_CODEC_E;
```

【成员】

成员名称	描述
CVI_RTSP_SERVICE_VIDEO_CODEC_H264	H264
CVI_RTSP_SERVICE_VIDEO_CODEC_H265	H265
CVI_RTSP_SERVICE_VIDEO_CODEC_JPEG	JPEG

【注意事项】

无。

【相关数据类型及接口】

无。

9.1.2.4 CVI_RTSP_SERVICE_AUDIO_CODEC_E

【说明】

音频编码枚举

【定义】

```
typedef enum {
    CVI_RTSP_SERVICE_AUDIO_CODEC_NONE,
    CVI_RTSP_SERVICE_AUDIO_CODEC_PCM,
    CVI_RTSP_SERVICE_AUDIO_CODEC_AAC
} CVI_RTSP_SERVICE_AUDIO_CODEC_E;
```

【成员】

成员名称	描述
CVI_RTSP_SERVICE_AUDIO_CODEC_PCM	PCM
CVI_RTSP_SERVICE_AUDIO_CODEC_AAC	AAC

【注意事项】

无。

【相关数据类型及接口】

无。

9.1.2.5 CVI_RTSP_SERVICE_CALLBACK

【说明】

rtsp 回调函数

【定义】

```
typedef void(CVI_RTSP_SERVICE_CALLBACK) (int32_t references, void *arg);
```

【成员】

成员名称	描述
references	会话值
arg	参数

【注意事项】

无。

【相关数据类型及接口】

无。

9.1.2.6 CVI_RTSP_SERVICE_HANDLE_T

【说明】

rtsp 服务句柄。

【定义】

```
typedef void *CVI_RTSP_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.2 Storage_Service

存储管理服务与 sd 卡有关。

9.2.1 API 参考

该组件提供以下 API:

- `CVI_STORAGE_SERVICE_Create` : 创建存储管理服务
- `CVI_STORAGE_SERVICE_Destroy` : 销毁存储管理服务
- `CVI_STORAGE_SERVICE_GetFsInfo` : 获取文件系统信息
- `CVI_STORAGE_SERVICE_GetDevInfo` : 获取设备信息
- `CVI_STORAGE_SERVICE_Format` : sd 卡格式化
- `CVI_STORAGE_SERVICE_Mount` : 挂载 sd 卡
- `CVI_STORAGE_SERVICE_Umount` : 卸载 sd 卡

9.2.1.1 `CVI_STORAGE_SERVICE_Create`

【描述】

创建存储管理服务

【语法】

```
int32_t CVI_STORAGE_SERVICE_Create(CVI_STORAGE_HANDLE_T *hdl, CVI_STORAGE_SERVICE_PARAM_S *params);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入
params	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_storage.h`
- 库文件: `libcvi_storage.a/libcvi_storage.so`

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.2 CVI_STORAGE_SERVICE_Destroy

【描述】

销毁存储管理服务

【语法】

```
int32_t CVI_STORAGE_SERVICE_Destroy(CVI_STORAGE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h
- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.3 CVI_STORAGE_SERVICE_GetFsInfo

【描述】

获取文件系统信息

【语法】

```
int32_t CVI_STORAGE_SERVICE_GetFsInfo(CVI_STORAGE_HANDLE_T hdl, STG_FS_
↪INFO_S *pstInfo);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入
pstInfo	文件系统信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h
- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.4 CVI_STORAGE_SERVICE_GetDevInfo

【描述】

获取设备信息

【语法】

```
int32_t CVI_STORAGE_SERVICE_GetDevInfo(CVI_STORAGE_HANDLE_T hdl, CVI_STG_
↪DEV_INFO_S *pstInfo);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入
pstInfo	设备信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h
- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.5 CVI_STORAGE_SERVICE_Format

【描述】

sd 卡格式化

【语法】

```
int32_t CVI_STORAGE_SERVICE_Format(CVI_STORAGE_HANDLE_T hdl, char *labelname);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入
labelname	标签名	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h

- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.6 CVI_STORAGE_SERVICE_Mount

【描述】

挂载 sd 卡

【语法】

```
int32_t CVI_STORAGE_SERVICE_Mount(CVI_STORAGE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h
- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.1.7 CVI_STORAGE_SERVICE_Umount

【描述】

卸载 sd 卡

【语法】

```
int32_t CVI_STORAGE_SERVICE_Umount(CVI_STORAGE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	存储管理服务	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_storage.h
- 库文件: libcvi_storage.a/libcvi_storage.so

【注意】

无

【举例】

无

【相关主题】

无

9.2.2 数据类型

相关数据类型定义如下:

- CVI_STORAGE_SERVICE_PARAM_S : 存储服务参数
- CVI_STORAGE_SERVICE_EVENT_CALLBACK : 存储事件处理回调函数
- CVI_STORAGE_SERVICE_HANDLE_T : 存储服务句柄
- STORAGE_LABEL_LEN : 标签长度

9.2.2.1 STORAGE_LABEL_LEN

【说明】

标签长度

【定义】

```
#define STORAGE_LABEL_LEN (64)
```

【注意事项】

无。

【相关数据类型及接口】

无。

9.2.2.2 CVI_STORAGE_SERVICE_PARAM_S

【说明】

存储服务参数

【定义】

```
typedef struct CVI_STORAGE_SERVICE_PARAM_S {  
    STG_DEVINFO_S devinfo;  
    char          labelname[STORAGE_LABEL_LEN];  
    CVI_STORAGE_SERVICE_EVENT_CALLBACK          storage_event_callback;  
} CVI_STORAGE_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
devinfo	设备信息
labelname	标签
storage_event_callback	存储事件处理回调函数

【注意事项】

无。

【相关数据类型及接口】

无。

9.2.2.3 CVI_STORAGE_SERVICE_EVENT_CALLBACK

【说明】

存储事件处理回调函数

【定义】

```
typedef int32_t (*CVI_STORAGE_SERVICE_EVENT_CALLBACK)(CVI_STG_STATE_E state);
```

【成员】

成员名称	描述
state	存储状态

【注意事项】

无。

【相关数据类型及接口】

无。

9.2.2.4 CVI_STORAGE_SERVICE_HANDLE_T

【说明】

存储服务句柄

【定义】

```
typedef void *CVI_STORAGE_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.3 Photo_Service

拍照服务指的是支持抓拍不同分辨率的图片，最小分辨率为 VGA，最大分辨率为 12M。

9.3.1 API 参考

该组件提供以下 API:

- `CVI_PHOTO_SERVICE_Create`: 创建拍照服务
- `CVI_PHOTO_SERVICE_Destroy`: 销毁拍照服务
- `CVI_PHOTO_SERVICE_WaitPivFinish`: 等待抓拍结束
- `CVI_PHOTO_SERVICE_PivCapture`: 抓拍

9.3.1.1 `CVI_PHOTO_SERVICE_Create`

【描述】

创建拍照服务

【语法】

```
int32_t CVI_PHOTO_SERVICE_Create(PHOTO_SERVICE_HANDLE_T *hdl, CVI_PHOTO_
↳SERVICE_PARAM_S *param);
```

【参数】

参数名称	描述	输入/输出
hdl	拍照服务句柄	输入
param	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_photo_service.h`
- 库文件: `libcvi_photo_service.a/libcvi_photo_service.so`

【注意】

无

【举例】

无

【相关主题】

无

9.3.1.2 CVI_PHOTO_SERVICE_Destroy

【描述】

销毁拍照服务

【语法】

```
int32_t CVI_PHOTO_SERVICE_Destroy(PHOTO_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	拍照服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_photo_service.h
- 库文件: libcvi_photo_service.a/libcvi_photo_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.3.1.3 CVI_PHOTO_SERVICE_PivCapture

【描述】

抓拍

【语法】

```
int32_t CVI_PHOTO_SERVICE_PivCapture(PHOTO_SERVICE_HANDLE_T hdl, char *file_  
↪ name);
```

【参数】

参数名称	描述	输入/输出
hdl	拍照服务句柄	输入
file_name	文件名	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_photo_service.h
- 库文件: libcvi_photo_service.a/libcvi_photo_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.3.1.4 CVI_PHOTO_SERVICE_WaitPivFinish**【描述】**

等待抓拍结束

【语法】

```
void CVI_PHOTO_SERVICE_WaitPivFinish(PHOTO_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	拍照服务句柄	输入

【返回值】

无

【需求】

- 头文件: cvi_photo_service.h
- 库文件: libcvi_photo_service.a/libcvi_photo_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.3.2 数据类型

相关数据类型定义如下：

- CVI_PHOTO_SERVICE_PARAM_S：拍照服务参数
- CVI_PHOTO_SERVICE_HANDLE_T：拍照服务句柄

9.3.2.1 CVI_PHOTO_SERVICE_PARAM_S

【说明】

拍照服务参数

【定义】

```
typedef struct _cvi_PHOTO_SERVICE_PARAM_S {  
    int32_t photo_id;  
    uint32_t prealloclen;  
  
    CVI_MAPI_VENC_HANDLE_T photo_venc_hdl;  
    uint32_t photo_bufsize;  
  
    CVI_MAPI_VPROC_HANDLE_T thumbnail_vproc;  
    int32_t vproc_chn_id_thumbnail;  
    CVI_MAPI_VENC_HANDLE_T thumbnail_venc_hdl;  
    uint32_t thumbnail_bufsize;  
  
    CVI_MAPI_VPROC_HANDLE_T src_vproc;  
    int32_t src_vproc_chn_id;  
  
    CVI_MAPI_VPROC_HANDLE_T scale_vproc;  
    int32_t scale_vproc_chn_id;  
  
    void *cont_photo_event_cb;  
} CVI_PHOTO_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
photo_id	sensor id
prealloclen	预分配大小
photo_venc_hdl	拍照编码句柄
photo_bufsize	用于编码的缓存大小
thumbnail_vproc	缩略图 vpss 句柄
vproc_chn_id_thumbnail	缩略图 vpss 通道 id
thumbnail_venc_hdl	缩略图编码句柄
thumbnail_bufsize	用于缩略图编码的缓存大小
src_vproc	源 vpss 句柄
src_vproc_chn_id	源 vpss 通道 id
scale_vproc	缩放 vpss 句柄
scale_vproc_chn_id	缩放 vpss 通道 i
cont_photo_event_cb	拍照事件处理函数

【注意事项】

无。

【相关数据类型及接口】

无。

9.3.2.2 CVI_PHOTO_SERVICE_HANDLE_T

【说明】

拍照服务句柄

【定义】

```
typedef void *CVI_PHOTO_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.4 Liveview_Service

预览服务支持镜头角度移动和视角放大缩小，目前只支持单镜头。

9.4.1 API 参考

该组件提供以下 API：

- `CVI_LIVEVIEW_SERVICE_Create`：创建预览服务
- `CVI_LIVEVIEW_SERVICE_Destroy`：销毁预览服务
- `CVI_LIVEVIEW_SERVICE_GetParam`：获取窗口参数

9.4.1.1 `CVI_LIVEVIEW_SERVICE_Create`

【描述】

创建预览服务

【语法】

```
int32_t CVI_LIVEVIEW_SERVICE_Create(CVI_LIVEVIEW_HANDLE_T *hdl, CVI_LIVEVIEW_SERVICE_PARAM_S *params);
```

【参数】

参数名称	描述	输入/输出
hdl	预览服务句柄	输入
params	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_liveview.h
- 库文件：libcvi_liveview.a/libcvi_liveview.so

【注意】

无

【举例】

无

【相关主题】

无

9.4.1.2 CVI_LIVEVIEW_SERVICE_Destroy**【描述】**

销毁预览服务

【语法】

```
int32_t CVI_LIVEVIEW_SERVICE_Destroy(CVI_LIVEVIEW_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	预览服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_liveview.h
- 库文件: libcvi_liveview.a/libcvi_liveview.so

【注意】

无

【举例】

无

【相关主题】

无

9.4.1.3 CVI_LIVEVIEW_SERVICE_GetParam**【描述】**

获取窗口参数

【语法】

```
int32_t CVI_LIVEVIEW_SERVICE_GetParam(CVI_LIVEVIEW_HANDLE_T hdl, int32_t wndId,  
↪ CVI_LIVEVIEW_SERVICE_WNDATTR_S *WndParam);
```

【参数】

参数名称	描述	输入/输出
hdl	预览服务句柄	输入
WndParam	窗口参数	输出
wndId	窗口值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_liveview.h`
- 库文件: `libcvi_liveview.a/libcvi_liveview.so`

【注意】

无

【举例】

无

【相关主题】

无

9.4.2 数据类型

相关数据类型定义如下:

- `CVI_LIVEVIEW_SERVICE_WNDATTR_S`: 定义窗口属性
- `CVI_LIVEVIEW_SERVICE_ATTR_S`: 预览服务属性
- `CVI_LIVEVIEW_SERVICE_PARAM_S`: 预览服务参数
- `CVI_LIVEVIEW_SERVICE_HANDLE_T`: 预览服务句柄
- `CVI_DISP_MAX_WND_NUM`: 最大窗口数量

9.4.2.1 CVI_DISP_MAX_WND_NUM

【说明】

最大窗口数量

【定义】

```
#define CVI_DISP_MAX_WND_NUM (16)
```

【注意事项】

无。

【相关数据类型及接口】

无。

9.4.2.2 CVI_LIVEVIEW_SERVICE_WNDATTR_S

【说明】

定义窗口属性

【定义】

```
typedef struct CVI_LIVEVIEW_SERVICE_WNDATTR_S {  
    bool        WndEnable;  
    bool        UsedCrop;  
    bool        SmallWndEnable;  
    uint32_t    BindVprocId;  
    uint32_t    BindVprocChnId;  
    uint32_t    WndX;  
    uint32_t    WndY;  
    uint32_t    WndWidth;  
    uint32_t    WndHeight;  
    uint32_t    WndsWidth;  
    uint32_t    WndsHeight;  
    uint32_t    WndsX;  
    uint32_t    WndsY;  
    uint32_t    OneStep;  
    uint32_t    WndMirror;  
    uint32_t    WndFilp;  
    int32_t     yStep;  
} CVI_LIVEVIEW_SERVICE_WNDATTR_S;
```

【成员】

成员名称	描述
WndEnable	使能窗口
UsedCrop	是否已裁剪
SmallWndEnable	使能小窗口
BindVprocId	绑定的 vpss id
BindVprocChnId	绑定的 vpss 通道 id
WndX	窗口 x 坐标
WndY	窗口 y 坐标
WndWidth	窗口宽度
WndHeight	窗口高度
WndsWidth	小窗口宽度
WndsHeight	小窗口高度
WndsX	小窗口 x 坐标
WndsY	小窗口 y 坐标
OneStep	步长
WndMirror	窗口镜面翻转
WndFilp	窗口翻转
yStep	y 步长

【注意事项】

无。

【相关数据类型及接口】

无。

9.4.2.3 CVI_LIVEVIEW_SERVICE_ATTR_S

【说明】

预览服务属性

【定义】

```
typedef struct CVI_LIVEVIEW_SERVICE_ATTR_S {
    CVI_LIVEVIEW_SERVICE_WNDATTR_S wnd_attr;
    CVI_MAPI_VPROC_HANDLE_T vproc_hdl;
} CVI_LIVEVIEW_SERVICE_ATTR_S;
```

【成员】

成员名称	描述
wnd_attr	窗口属性
vproc_hdl	vpss 句柄

【注意事项】

无。

【相关数据类型及接口】

无。

9.4.2.4 CVI_LIVEVIEW_SERVICE_PARAM_S

【说明】

预览服务参数

【定义】

```
typedef struct CVI_LIVEVIEW_SERVICE_PARAM_S {  
    uint32_t WndCnt;  
    VPSS_GRP vproc_id;  
    VB_POOL hVbPool;  
    CVI_LIVEVIEW_SERVICE_ATTR_S LiveviewService[CVI_DISP_MAX_WND_NUM];  
} CVI_LIVEVIEW_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
WndCnt	窗口数量
vproc_id	vpss id
hVbPool	VB 池
LiveviewService	预览服务属性数组

【注意事项】

无。

【相关数据类型及接口】

无。

9.4.2.5 CVI_LIVEVIEW_SERVICE_HANDLE_T

【说明】

预览服务句柄

【定义】

```
typedef void *CVI_LIVEVIEW_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.5 ADAS_Service

ADAS 服务提供车辆检测功能，包含前车启动提醒，保持车距提醒，车距过近警告，车道偏移提醒及标注车辆和车道线。

9.5.1 API 参考

该组件提供以下 API：

- `CVI_ADAS_SERVICE_Create`：创建 ADAS 服务
- `CVI_ADAS_SERVICE_Destroy`：销毁 ADAS 服务
- `CVI_ADAS_SERVICE_SetState`：设置 ADAS 任务状态

9.5.1.1 CVI_ADAS_SERVICE_Create

【描述】

创建 ADAS 服务

【语法】

```
int32_t CVI_ADAS_SERVICE_Create(CVI_ADAS_SERVICE_HANDLE_T *hdl, CVI_ADAS_SERVICE_PARAM_S *ADASParam);
```

【参数】

参数名称	描述	输入/输出
hdl	ADAS 服务句柄	输入
ADASParam	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_adas_service.h
- 库文件：libcvi_adas_service.a/libcvi_adas_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.5.1.2 CVI_ADAS_SERVICE_Destroy

【描述】

销毁 ADAS 服务

【语法】

```
int32_t CVI_ADAS_SERVICE_Destroy(CVI_ADAS_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	ADAS 服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_adas_service.h
- 库文件: libcvi_adas_service.a/libcvi_adas_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.5.1.3 CVI_ADAS_SERVICE_SetState

【描述】

设置 ADAS 任务状态

【语法】

```
void CVI_ADAS_SERVICE_SetState(int32_t id, int32_t en);
```

【参数】

参数名称	描述	输入/输出
id	camid	输入
en	状态参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_adas_service.h`
- 库文件: `libcvi_adas_service.a/libcvi_adas_service.so`

【注意】

无

【举例】

无

【相关主题】

无

9.5.2 数据类型

相关数据类型定义如下:

- `CVI_ADAS_SERVICE_VOICE_CALLBACK`: 定义语音提醒回调函数
- `CVI_ADAS_SERVICE_LABEL_CALLBACK`: 定义车辆及车道线标注回调函数
- `CVI_ADAS_SERVICE_CALLBACK_S`: ADAS 服务回调函数结构体
- `CVI_ADAS_SERVICE_VPROC_ATTR_S`: ADAS 服务 vpss 属性
- `CVI_ADAS_SERVICE_MODEL_ATTR_S`: ADAS 服务模型属性
- `CVI_ADAS_SERVICE_HANDLE_T`: ADAS 服务句柄
- `CVI_ADAS_SERVICE_ATTR_S`: ADAS 服务属性
- `CVI_ADAS_SERVICE_PARAM_S`: ADAS 服务参数
- `CVI_ADAS_SERVICE_CTX_S`: ADAS 服务上下文
- `CVI_ADAS_SERVICE_CMD_E`: ADAS 服务命令

9.5.2.1 CVI_ADAS_SERVICE_VOICE_CALLBACK

【说明】

定义语音提醒回调函数

【定义】

```
typedef int32_t (*CVI_ADAS_SERVICE_VOICE_CALLBACK)(int32_t index);
```

【成员】

成员名称	描述
index	命令参数

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.2 CVI_ADAS_SERVICE_LABEL_CALLBACK

【说明】

定义车辆及车道线标注回调函数

【定义】

```
typedef int32_t (*CVI_ADAS_SERVICE_LABEL_CALLBACK)(int32_t camid, int32_t index, uint32_t count, char* coordinates);
```

【成员】

成员名称	描述
index	命令参数
camid	sensor_id
count	数量
coordinates	坐标数组

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.3 CVI_ADAS_SERVICE_CALLBACK_S

【说明】

ADAS 服务回调函数结构体

【定义】

```
typedef struct _CVI_ADAS_CALLBACK_S {  
    void *pfnVoiceCb;  
    void *pfnLabelCb;  
} CVI_ADAS_SERVICE_CALLBACK_S;
```

【成员】

成员名称	描述
pfnVoiceCb	语音提醒回调函数
pfnLabelCb	车辆及车道线标注回调函数

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.4 CVI_ADAS_SERVICE_VPROC_ATTR_S

【说明】

ADAS 服务 vpss 属性

【定义】

```
typedef struct _CVI_ADAS_SERVICE_VPROC_S {  
    CVI_MAPI_VPROC_HANDLE_T vprocHandle;  
    int32_t vprocId;  
    uint32_t vprocChnId;  
} CVI_ADAS_SERVICE_VPROC_ATTR_S;
```

【成员】

成员名称	描述
vprocHandle	vpss 句柄
vprocId	vpss ID
vprocChnId	vpss 通道 ID

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.5 CVI_ADAS_SERVICE_MODEL_ATTR_S

【说明】

ADAS 服务模型属性

【定义】

```
typedef struct _CVI_ADAS_SERVICE_MODEL_ATTR_S{  
    float  fps;  
    int32_t width;  
    int32_t height;  
    char CarModelPath[128];  
    char LaneModelPath[128];  
} CVI_ADAS_SERVICE_MODEL_ATTR_S;
```

【成员】

成员名称	描述
fps	模型帧率
width	用于模型内部的 VB 的宽度
height	用于模型内部的 VB 的高度
CarModelPath	车辆检测模型路径
LaneModelPath	车道线检测模型路径

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.6 CVI_ADAS_SERVICE_HANDLE_T

【说明】

ADAS 服务句柄

【定义】

```
typedef void *CVI_ADAS_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.7 CVI_ADAS_SERVICE_ATTR_S

【说明】

ADAS 服务属性

【定义】

```
typedef struct _CVI_ADAS_SERVICE_ATTR_S {  
    CVI_ADAS_SERVICE_HANDLE_T    ADASHdl;  
    CVI_ADAS_SERVICE_CALLBACK_S    stADASCallback;  
    CVI_ADAS_SERVICE_VPROC_ATTR_S    stVprocAttr;  
    CVI_ADAS_SERVICE_MODEL_ATTR_S    stADASModelAttr;  
} CVI_ADAS_SERVICE_ATTR_S;
```

【成员】

成员名称	描述
ADASHdl	ADAS 服务句柄
stADASCallback	ADAS 服务回调函数结构体
stVprocAttr	ADAS 服务 vpss 属性
stADASModelAttr	ADAS 服务模型属性

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.8 CVI_ADAS_SERVICE_PARAM_S

【说明】

ADAS 服务参数

【定义】

```
typedef struct _CVI_ADAS_SERVICE_PARAM_S {  
    int32_t    camid;  
    CVI_ADAS_SERVICE_MODEL_ATTR_S    stADASModelParam;  
    CVI_ADAS_SERVICE_VPROC_ATTR_S    stVPSSParam;  
    void    *adas_voice_event_cb;  
    void    *adas_label_event_cb;  
} CVI_ADAS_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
camid	sensor id
stVPSSParam	ADAS 服务 vpss 属性
stADASModelAttr	ADAS 服务模型属性
adas_voice_event_cb	语音提醒回调函数
adas_label_event_cb	车辆及车道线标注回调函数

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.9 CVI_ADAS_SERVICE_CTX_S

【说明】

ADAS 服务上下文

【定义】

```
typedef struct _CVI_ADAS_SERVICE_CTX_S{  
    int32_t          id;  
    int32_t          state;  
    void             *attr;  
    int32_t          tdl_grp_id;  
    cvi_osal_task_handle_t    adas_task;  
    pthread_mutex_t  adas_mutex;  
    cvitdl_app_handle_t    app_handle;  
    cvitdl_handle_t    tdl_handle;  
}CVI_ADAS_SERVICE_CTX_S, *ADAS_CONTEXT_HANDLE_S;
```

【成员】

成员名称	描述
id	sensor id
state	任务状态
attr	ADAS 服务属性
tdl_grp_id	模型内部创建的 vpss id
adas_task	adas 任务
adas_mutex	adas 任务互斥量
app_handle	模型内部 app 句柄
tdl_handle	tdl 句柄

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2.10 CVI_ADAS_SERVICE_CMD_E

【说明】

ADAS 服务命令

【定义】

```
typedef enum _CVI_ADAS_SERVICE_CMD_E
{
    CVI_ADAS_SERVICE_NORMAL = 0,
    CVI_ADAS_SERVICE_CAR_MOVING,
    CVI_ADAS_SERVICE_CAR_CLOSING,
    CVI_ADAS_SERVICE_CAR_COLLISION,
    CVI_ADAS_SERVICE_CAR_LANE,
    CVI_ADAS_SERVICE_LABEL_CAR,
    CVI_ADAS_SERVICE_LABEL_LANE,
    CVI_ADAS_SERVICE_BUTT
} CVI_ADAS_SERVICE_CMD_E;
```

【成员】

成员名称	描述
CVI_ADAS_SERVICE_NORMAL	行驶正常
CVI_ADAS_SERVICE_CAR_MOVING	前车启动
CVI_ADAS_SERVICE_CAR_CLOSING	保持车距
CVI_ADAS_SERVICE_CAR_COLLISION	碰撞警告
CVI_ADAS_SERVICE_LABEL_CAR	车辆标注
CVI_ADAS_SERVICE_LABEL_LANE	车道线标注

【注意事项】

无。

【相关数据类型及接口】

无。

9.6 Speech_Service

语音识别服务提供语音控制功能，包括切换前后路，拍照，录像及开关 wifi。

9.6.1 API 参考

该组件提供以下 API：

- `CVI_SPEECH_SERVICE_Create`：创建语音识别服务
- `CVI_SPEECH_SERVICE_Destroy`：销毁语音识别服务
- `CVI_SPEECH_SERVICE_StartSpeech`：开始语音识别服务
- `CVI_SPEECH_SERVICE_StopSpeech`：暂停语音识别服务
- `CVI_SPEECH_SERVICE_Register`：注册语音识别回调函数

9.6.1.1 CVI_SPEECH_SERVICE_Create

【描述】

创建语音识别服务

【语法】

```
int32_t CVI_SPEECH_SERVICE_Create(CVI_SPEECH_HANDLE_S *hdl, CVI_SPEECH_
↪SERVICE_PARAM_S *params);
```

【参数】

参数名称	描述	输入/输出
hdl	语音识别服务句柄	输入
params	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：cvi_speech.h
- 库文件：libcvi_speech.a/libcvi_speech.so

【注意】

无

【举例】

无

【相关主题】

无

9.6.1.2 CVI_SPEECH_SERVICE_Destroy

【描述】

销毁语音识别服务

【语法】

```
int32_t CVI_SPEECH_SERVICE_Destroy(CVI_SPEECH_HANDLE_S hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	语音识别服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_speech.h
- 库文件: libcvi_speech.a/libcvi_speech.so

【注意】

无

【举例】

无

【相关主题】

无

9.6.1.3 CVI_SPEECH_SERVICE_StartSpeech

【描述】

开始语音识别服务

【语法】

```
int32_t CVI_SPEECH_SERVICE_StartSpeech(CVI_SPEECH_HANDLE_S hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	语音识别服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_speech.h
- 库文件: libcvi_speech.a/libcvi_speech.so

【注意】

无

【举例】

无

【相关主题】

无

9.6.1.4 CVI_SPEECH_SERVICE_StopSpeech

【描述】

暂停语音识别服务

【语法】

```
int32_t CVI_SPEECH_SERVICE_StopSpeech(CVI_SPEECH_HANDLE_S hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	语音识别服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_speech.h
- 库文件: libcvi_speech.a/libcvi_speech.so

【注意】

无

【举例】

无

【相关主题】

无

9.6.1.5 CVI_SPEECH_SERVICE_Register

【描述】

注册语音识别回调函数

【语法】

```
void CVI_SPEECH_SERVICE_Register(FUNC_SPEECH_HANDLE pfunction);
```

【参数】

参数名称	描述	输入/输出
pfunction	回调函数	输入

【返回值】

无

【需求】

- 头文件: cvi_speech.h
- 库文件: libcvi_speech.a/libcvi_speech.so

【注意】

无

【举例】

无

【相关主题】

无

9.6.2 数据类型

相关数据类型定义如下：

- `FUNC_SPEECH_HANDLE`：定义语音识别回调函数
- `CVI_SPEECH_HANDLE_S`：语音识别服务句柄
- `CVI_SPEECH_SERVICE_PARAM_S`：语音识别服务参数

9.6.2.1 `FUNC_SPEECH_HANDLE`

【说明】

定义语音识别回调函数

【定义】

```
typedef int32_t (*FUNC_SPEECH_HANDLE)( int32_t index );
```

【成员】

成员名称	描述
index	命令参数

【注意事项】

无。

【相关数据类型及接口】

无。

9.6.2.2 `CVI_SPEECH_HANDLE_S`

【说明】

语音识别服务句柄

【定义】

```
typedef void *CVI_SPEECH_HANDLE_S;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.6.2.3 CVI_SPEECH_SERVICE_PARAM_S

【说明】

语音识别服务参数

【定义】

```
typedef struct CVI_SPEECH_SERVICE_PARAM_S {  
    uint32_t enable;  
    uint32_t SampleRate;  
    uint32_t BitWidth;  
    uint32_t AiNumPerFrm;  
    char ModelPath[128];  
} CVI_SPEECH_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
enable	使能
SampleRate	采样率
BitWidth	位宽
AiNumPerFrm	每帧采样数
ModelPath	语音识别模型路径

【注意事项】

无。

【相关数据类型及接口】

无。

9.7 QRCode_Service

QRCode 服务提供二维码识别。

9.7.1 API 参考

该组件提供以下 API:

- `CVI_QRCode_Service_Create` : 创建二维码扫描服务
- `CVI_QRCode_Service_Destroy` : 销毁二维码扫描服务

9.7.1.1 CVI_QRCode_Service_Create

【描述】

创建二维码扫描服务

【语法】

```
int32_t CVI_QRCode_Service_Create(CVI_QRCODE_SERVICE_HANDLE_T *hdl, CVI_QRCODE_SERVICE_PARAM_S *attr);
```

【参数】

参数名称	描述	输入/输出
hdl	二维码服务句柄	输入
attr	参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: cvi_qrcode_ser.h
- 库文件: libcvi_qrcode_service.a/libcvi_qrcode_service.so

【注意】

无

【举例】

无

【相关主题】

无

9.7.1.2 CVI_QRCode_Service_Destroy

【描述】

销毁二维码扫描服务

【语法】

```
int32_t CVI_QRCode_Service_Destroy(CVI_QRCODE_SERVICE_HANDLE_T hdl);
```

【参数】

参数名称	描述	输入/输出
hdl	二维码服务句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件: `cvi_qrcode_ser.h`
- 库文件: `libcvi_qrcode_service.a/libcvi_qrcode_service.so`

【注意】

无

【举例】

无

【相关主题】

无

9.7.2 数据类型

相关数据类型定义如下:

- `CVI_QRCODE_SERVICE_HANDLE_T`: 二维码服务句柄
- `CVI_QRCODE_SERVICE_PARAM_S`: 二维码服务参数

9.7.2.1 `CVI_QRCODE_SERVICE_HANDLE_T`

【说明】

二维码服务句柄

【定义】

```
typedef void *CVI_QRCODE_SERVICE_HANDLE_T;
```

【成员】

无

【注意事项】

无。

【相关数据类型及接口】

无。

9.7.2.2 CVI_QRCODE_SERVICE_PARAM_S

【说明】

二维码服务参数

【定义】

```
typedef struct cviQRCODE_PARAM_S
{
    uint32_t w;
    uint32_t h;
    CVI_MAPI_VPROC_HANDLE_T vproc;
    uint32_t vproc_chnid;
} CVI_QRCODE_SERVICE_PARAM_S;
```

【成员】

成员名称	描述
w	图像宽度
h	图像高度
vproc	vpss 句柄
vproc_chnid	vpss chn id

【注意事项】

无。

【相关数据类型及接口】

无。