

音频服务 SDK 使用指南

李家伟 | 7月6日修改

本文面向基于 Booster Robotics SDK 进行二次开发的用户，说明当前 SDK 公开的音频服务能力、接口边界和典型使用方式。
依据当前源码重新整理：

- booster_robotics_sdk/include/booster/robot/audio/audio_manager.h
- booster_robotics_sdk/include/booster/robot/audio/audio_player.h
- booster_robotics_sdk/include/booster/robot/audio/audio_recorder.h
- booster_robotics_sdk/include/booster/robot/audio/audio_localizer.h
- booster_robotics_sdk/include/booster/robot/audio/audio_capture_stream.h
- booster_robotics_sdk/include/booster/robot/audio/audio_types.h
- booster_robotics_sdk/src/robot/audio/*.cpp
- booster_robotics_sdk/src/python/binding.cpp
- booster_robotics_sdk/python_example/sdk_pybind_audio_example.py
- booster_system/booster_audio/service/include/ros/audio_rpc_method.h
- booster_system/booster_audio/service/include/ros/audio_service_constants.h
- booster_system/booster_audio/service/src/ros/handlers/*.cpp

1. 当前能力概览

当前 SDK 提供以下音频业务能力和 1 个统一入口管理器：

能力	C++ 入口	Python 入口	说明
统一管理	AudioManager	AudioManager	初始化 DDS/RPC、注册 client_id、创建子对象、系统控制 [1.7]、设备查询 [1.7]、蓝牙管理 [1.7]、全局 topic 回调
播放	AudioPlayer	AudioPlayer	文件播放和 PCM 流式播放
录音	AudioRecorder	AudioRecorder	录制机器人端输入音频到文件
声源定位	AudioLocalizer	AudioLocalizer	查询 DOA 角度
实时采集流	AudioCaptureStream	AudioCaptureStream	订阅 raw PCM / NAEC PCM 实时帧
系统音频控制 [1.7]	AudioManager	AudioManager	系统音量、系统静音
设备查询 [1.7]	AudioManager	AudioManager	查询输入/输出音频设备
蓝牙音频 [1.7]	AudioManager	AudioManager	扫描、列设备、连接、断开、忘记设备

当前 SDK 与音频服务通过 DDS/RPC 通信。SDK 侧 RPC channel 与服务端 AudioManager 保持一一对应，服务端当前注册 35 个 RPC 方法，范围从 register_client 到 forget_bluetooth_device。

版本标记说明：[1.7] 表示该能力或接口从 1.7 版本开始新增/公开。标记优先放在能力名称、章节标题、接口说明或推荐场景后；同一能力下的参数字段不重复标注。

1.1 Release Notes: SDK 1.7 主要变化

本节汇总音频服务 SDK 在 1.7 版本面向业务开发者的主要变化。完整接口说明见后文各章节。

- AudioManager 直接承担更多系统级音频能力，包括系统音量、系统静音、设备查询和蓝牙音频管理。

- 新增系统音频控制接口：`SetSystemVolume()`、`GetSystemVolume()`、`SetSystemMute()`、`GetSystemMute()`。
- 新增设备查询接口：`GetDevices()`，可查询输入或输出音频设备，并用于后续选择播放、录音或采集流的目标设备。
- 新增 session 级首选设备设置：`AudioPlayer.SetPreferredDevice()`、`AudioRecorder.SetPreferredDevice()`、`AudioCaptureStream.SetPreferredDevice()`；该设置只作用于当前 session，不等于系统默认设备。
- 新增蓝牙音频管理接口：扫描、获取设备列表、连接、断开和忘记蓝牙音频设备。蓝牙耳机默认使用 A2DP 模式，如需使用耳机麦克风，需要显式选择 HFP 模式。
- Python 绑定使用 PascalCase 方法名，例如 `GetDevices()`、`SetPreferredDevice()`、`ConnectBluetoothDevice()`；旧 camelCase 方法名不再作为当前公开接口使用。
- 旧的 `AudioSystemController` 入口不再公开，系统音量/静音能力统一从 `AudioManager` 调用。

2. Deprecated 接口

以下接口或文档描述来自旧版文档，但当前 SDK 公共接口已经不支持或不应作为公开 API 使用：

	A	B	C
1	旧接口/旧描述	当前状态	替代方式
2	<code>include/booster/robot/audio/audio_system_controller.h</code> <code>[[Deprecated]]</code>	当前 SDK 头目录中不存在该文件	使用 <code>AudioManager::SetSystemVolume/GetSystemVolume/SetSystemMute/GetSystemMute</code> [1.7]
3	<code>AudioSystemController</code> <code>[[Deprecated]]</code>	当前 SDK 不再公开该类	使用 <code>AudioManager</code> 直接做系统音量/静音控制 [1.7]
4	<code>AudioManager::CreateSystemController()</code> <code>[[Deprecated]]</code>	当前 <code>AudioManager</code> 没有该工厂方法	使用 <code>AudioManager</code> 自身的系统控制接口
5	<code>AudioManager::CallService(...)</code> 作为 Python/C++ 业务公开接口 <code>[[Deprecated]]</code>	当前 <code>CallService</code> 是 <code>AudioManager</code> 私有实现细节，Python 绑定也没有暴露	使用 <code>AudioManager</code> 、 <code>AudioPlayer</code> 、 <code>AudioRecorder</code> 、 <code>AudioCaptureStream</code> 的高层方法
6	<code>AudioManager::ServiceMethod</code> 作为业务层公开枚举 <code>[[Deprecated]]</code>	当前 <code>ServiceMethod</code> 是私有枚举	不直接依赖底层 <code>method</code> ；如需排查可参考本文 RPC 映射
7	Python 侧 camelCase 方法名，例如 <code>setPreferredDevice</code> <code>[[Deprecated]]</code>	当前 Python 绑定只保留 PascalCase	使用 <code>SetPreferredDevice</code> 、 <code>GetDevices</code> 、 <code>ConnectBluetoothDevice</code> 等

3. 使用前提

1. 机器人端音频服务已经启动。
2. SDK 与机器人端 DDS/RPC 网络可达。
3. C++ 用户需要主动检查 `int32_t` 返回值。
4. Python 绑定对多数高层方法做了异常封装，底层返回非 0 时抛出 `RuntimeError("API call failed, code = N")`。
5. Python 复杂网络环境建议显式初始化 DDS：

代码块

```
1 br.ChannelFactory.Instance().Init(0, net_if)
```

`AudioManager::Init()` 会尝试初始化音频 DDS runtime，并向音频服务注册 `client_id`。如果 `FASTRTPS_DEFAULT_PROFILES_FILE` 指向旧路径，SDK 会优先使用 `/opt/booster/BoosterRos2/fastdds_profile_udp_only.xml`。

4. 返回值与错误码

SDK RPC 通用状态码定义在 `include/booster/robot/rpc/error.hpp`：

	A	B	C
1	值	名称	含义
2	-1	kRpcStatusCodeInvalid	本地前置条件或参数非法
3	0	kRpcStatusCodeSuccess	成功
4	100	kRpcStatusCodeTimeout	请求超时
5	400	kRpcStatusCodeBadRequest	请求参数非法
6	409	kRpcStatusCodeConflict	当前状态冲突
7	429	kRpcStatusCodeRequestTooFrequent	请求过于频繁
8	500	kRpcStatusCodeInternalServerError	内部错误或响应解析失败
9	501	kRpcStatusCodeServerRefused	服务端拒绝
10	502	kRpcStatusCodeStateTransitionFailed	状态迁移失败

音频服务自身错误码来自 `booster_audio/service/include/audio_errors.h`：

值	名称	含义
2001	SERVICE_DEGRADED	服务降级或关键组件不可用
4001	BLUETOOTH_UNAVAILABLE	蓝牙子系统不可用
4002	BLUETOOTH_BUSY	蓝牙忙
4003	BLUETOOTH_UNKNOWN_DEVICE	未知蓝牙设备
4004	BLUETOOTH_PAIR_FAILED	配对失败
4005	BLUETOOTH_CONNECT_FAILED	连接失败
4006	BLUETOOTH_TIMEOUT	蓝牙操作超时
4007	BLUETOOTH_NOT_AUDIO_DEVICE	不是音频设备
4008	BLUETOOTH_PROFILE_UNSUPPORTED	profile 不支持
4009	BLUETOOTH_AUTH_REJECTED	认证拒绝
5001	DEVICE_BUSY	设备忙
7001	NOT_SUPPORTED	当前能力不支持
7002	LOCALIZATION_UNAVAILABLE	声源定位不可用
7003	ALSA_DEVICE_NOT_FOUND	ALSA 设备未找到
7004	ALGO_INIT_FAILED	算法初始化失败
7005	ALGO_PROCESS_FAILED	算法处理失败
7006	SYSTEM_VOLUME_CTRL_FAILED	系统音量控制失败
7007	SYSTEM_MUTE_CTRL_FAILED	系统静音控制失败
7008	MEDIA_DECODE_FAILED	媒体解码失败
7009	MEDIA_ENCODE_FAILED	媒体编码失败
7010	MEDIA_FORMAT_UNSUPPORTED	媒体格式不支持

SDK 侧还公开常量：

代码块

```
1 booster::robot::audio::kAudioErrNotSupported == 7001
```

Python 侧同名导出：

代码块

```
1 br.kAudioErrNotSupported
```

5. 推荐生命周期

典型调用顺序：

1. 初始化 `ChannelFactory`，Python 推荐显式指定网卡。
2. 创建 `AudioManager`。
3. 调用 `AudioManager.Init()` / `AudioManager::Init()`。
4. 通过 `AudioManager` 创建 `AudioPlayer` / `AudioRecorder` / `AudioLocalizer` / `AudioCaptureStream`。
5. 注册状态、进度、错误或帧回调。
6. 调用具体对象 `Init()` 建立服务端会话。
7. 调用 `Start()` / `Pause()` / `Stop()` / `Reset()` 等控制方法。
8. 会话结束后调用 `Destroy()` 或 `Release()`。
9. 全部音频对象不用后调用 `AudioManager.Shutdown()` / `AudioManager::Shutdown()`。

注意：

- 一个 `AudioPlayer` / `AudioRecorder` / `AudioCaptureStream` 对象对应一个服务端 session。
- `Destroy()` 会释放服务端会话。
- `Release()` 目前只在 `AudioPlayer` 和 `AudioRecorder` 上存在，语义等同 `Destroy()`。
- `AudioCaptureStream` 当前没有 `Release()`。
- `GetCachedState()` 只读本地缓存，不发起 RPC，不保证一定是服务端最新状态。

6. 核心类型

6.1 枚举

6.2 结构体

类型	关键字段	默认值/说明
PcmFormat	sample_rate_hz, channels, bits_per_sample	默认 16000/1/16
PlayerInitOptions	source_type, source_uri, sample_rate_hz, channels, bits_per_sample, priority	默认 PCM file、16000/1/16、kMedium；文件播放时 source_uri 必须使用绝对路径
RecorderInitOptions	output_path, sample_rate_hz, channels, bits_per_sample	默认 16000/1/16，output_path 必填且必须使用绝对路径
AudioCaptureStreamOptions	enable_raw_pcm, enable_naec_pcm, requested_raw_format	默认 raw 开、NAEC 关、16000/1/16
PlayerInfo	state, played_bytes, total_bytes, volume	播放状态与进度
RecorderInfo	state, captured_bytes	录音状态与进度
AudioCaptureFrame	frame_seq, timestamp_ms, raw_*, naec_*	实时帧数据
AudioCaptureStreamInfo	state, raw_enabled, naec_enabled, actual_*_format, published_frames, dropped_frames	采集流状态
AudioDeviceInfo	device_id, display_name, direction, transport, backend_affinity, is_available, is_system_default, supports_input, supports_output, provider_name, native_id, metadata_json	SDK 当前只公开 service 设备信息子集
AudioError	ret_code, ret_msg, error_category, error_detail	子对象错误回调入参
BluetoothDeviceInfo	address, name, rssi, state, major_class, paired, trusted, connected, is_audio_sink, is_audio_source, is_hfp_capable, connected_profiles, linked_pulse_sink_id, linked_pulse_source_id, last_seen_ms	蓝牙设备状态
BluetoothScanOptions	timeout_ms, audio_only	默认 30000, true
BluetoothConnectOptions	address, auto_pair, make_default, preferred_profile, timeout_ms	address 必填；默认 true/true/kNone/15000
BluetoothConnectResult	device, pulse_sink_id, pulse_source_id, active_profile, pulse_endpoint_ready, default_sink_applied, default_source_applied, default_route_error_code, default_route_error_msg	蓝牙连接结果

7. AudioManager

`AudioManager` 是当前音频 SDK 的统一入口。

7.1 生命周期与对象工厂

C++ 方法	Python 方法	说明
int32_t Init()	Init()	初始化音频 DDS runtime，注册 client_id，预热部分 RPC client
void Shutdown()	Shutdown()	关闭 topic subscriber 和 RPC client，清空本地状态
bool IsInitialized() const	IsInitialized()	查询本地初始化状态
std::string GetClientId() const	GetClientId()	返回服务端注册得到的 client_id
std::string GenerateRequestId()	GenerateRequestId()	生成 audio_req_<epoch_ms>_<seq>
std::shared_ptr<AudioPlayer> CreatePlayer()	CreatePlayer()	创建播放器对象
std::shared_ptr<AudioRecorder> CreateRecorder()	CreateRecorder()	创建录音对象
std::shared_ptr<AudioLocalizer> CreateLocalizer()	CreateLocalizer()	创建定位对象
std::shared_ptr<AudioCaptureStream> CreateCaptureStream()	CreateCaptureStream()	创建采集流对象

7.2 系统音频控制 [1.7]

C++ 方法	Python 方法	说明
int32_t SetSystemVolume(float volume)	SetSystemVolume(volume)	设置系统输出音量，范围 [0.0, 1.0] [1.7]
int32_t GetSystemVolume(float *volume)	GetSystemVolume()	查询系统输出音量 [1.7]
int32_t SetSystemMute(bool mute)	SetSystemMute(mute)	设置系统静音 [1.7]
int32_t GetSystemMute(bool *mute)	GetSystemMute()	查询系统静音 [1.7]

这些接口当前直接属于 `AudioManager`，不需要也不能再通过 `AudioSystemController` 调用。

7.3 设备查询 [1.7]

C++ 方法	Python 方法	说明
int32_t GetDevices(AudioDeviceQueryType query_type, std::vector<AudioDeviceInfo> *devices)	GetDevices(query_type)	查询输入或输出设备 [1.7]

`query_type` 只能是：

- `AudioDeviceQueryType::GET_DEVICES_INPUTS`
- `AudioDeviceQueryType::GET_DEVICES_OUTPUTS`

Python 示例脚本中 `devices` 命令默认查询输出设备：

```
代码块
1 devices [inputs|outputs]
```

7.4 蓝牙音频管理 [1.7]

C++ 方法	Python 方法	说明
int32_t StartBluetoothScan(const BluetoothScanOptions &options)	StartBluetoothScan(options)	开始扫描 [1.7]
int32_t StopBluetoothScan()	StopBluetoothScan()	停止扫描 [1.7]
int32_t GetBluetoothDevices(bool include_unpaired, std::vector<BluetoothDeviceInfo> *devices)	GetBluetoothDevices(include_unpaired=True)	查询蓝牙设备 [1.7]
int32_t ConnectBluetoothDevice(const BluetoothConnectOptions &options, BluetoothConnectResult *result)	ConnectBluetoothDevice(options)	连接蓝牙音频设备 [1.7]
int32_t DisconnectBluetoothDevice(const std::string &address)	DisconnectBluetoothDevice(address)	断开设备 [1.7]
int32_t ForgetBluetoothDevice(const std::string &address)	ForgetBluetoothDevice(address)	忘记设备 [1.7]

参数约束：

- BluetoothScanOptions.timeout_ms 不能小于 0。
- BluetoothConnectOptions.address 必填。
- BluetoothConnectOptions.timeout_ms 必须大于 0。
- ConnectBluetoothDevice 的 RPC timeout 取 $\max(5000, \text{options.timeout_ms})$ 。
- make_default=true 时, service 会尝试设置默认 sink/source, 结果通过 default_* 字段返回。

7.5 全局 topic 回调

C++ 方法	Python 方法	说明
void SetProgressCallback(ProgressCallback callback)	SetProgressCallback(callback)	订阅 /booster/audio/progress
void SetEventCallback(EventCallback callback)	SetEventCallback(callback)	订阅 /booster/audio/events
void SetErrorCallback(ErrorCallback callback)	SetErrorCallback(callback)	订阅 /booster/audio/errors
void SetBluetoothEventCallback(BluetoothEventCallback callback)	SetBluetoothEventCallback(callback)	订阅 /booster/audio/bluetooth_events [1.7]

全局 topic 回调收到的是服务级 topic, 可能包含同一进程中多个对象的消息。示例脚本用 `topic.header.client_id == manager.GetClientId()` 过滤当前 manager。

AudioTopicHeader 字段：

- client_id
- request_id
- session_type
- session_id
- timestamp_ms

8. AudioPlayer

8.1 适用场景

- 播放机器人端可访问的 PCM/WAV/MP3 文件。
- 将外部生成的 PCM 数据通过 SDK 分片推送给机器人端播放。

8.2 接口列表

C++ 方法	Python 方法	说明
int32_t Init(const PlayerInitOptions &options)	Init(options)	初始化播放会话
int32_t Start()	Start()	开始播放
int32_t Pause()	Pause()	暂停
int32_t Stop()	Stop()	停止
int32_t Reset()	Reset()	重置播放器
int32_t Destroy()	Destroy()	销毁服务端播放会话
int32_t Release()	Release()	当前等同 Destroy()
int32_t SetVolume(float volume)	SetVolume(volume)	设置播放器会话音量，范围 [0.0, 1.0]
int32_t SetPreferredDevice(const std::string &device_id)	SetPreferredDevice(device_id)	设置当前播放 session 的首选输出设备 [1.7]
int32_t GetInfo(PlayerInfo *out) const	GetInfo()	查询播放状态
int32_t PushPcmStream(const uint8_t *data, size_t size)	PushPcmStream(data)	推送 PCM 流数据
int64_t GetSessionId() const	GetSessionId()	查询 session id
PlayerState GetCachedState() const	GetCachedState()	查询本地缓存状态
void SetStateCallback(PlayerState Callback callback)	SetStateCallback(callback)	状态回调
void SetProgressCallback(PlayerProgressCallback callback)	SetProgressCallback(callback)	进度回调
void SetCompletionCallback(PlayerCompletionCallback callback)	SetCompletionCallback(callback)	完成回调
void SetErrorCallback(PlayerErrorCallback callback)	SetErrorCallback(callback)	错误回调

8.3 初始化模式

文件播放：

- `source_type` 可选 `kPcmFile`、`kWavFile`、`kMp3File`。
- `source_uri` 必填，必须使用机器人端音频服务可访问的绝对路径。
- PCM 文件必须正确设置 `sample_rate_hz/channels/bits_per_sample`。
- WAV/MP3 通常也建议填写真实格式，便于排查问题。

流式播放：

- `source_type` 必须是 `kPcmStream`。
- `source_uri` 可为空；SDK 会使用内部占位值 `pcm_stream`。
- `Init()` 后调用 `Start()`，再通过 `PushPcmStream()` 推送 PCM 字节。
- `PushPcmStream()` 只接受非空数据，且当前对象必须以 `kPcmStream` 初始化。

8.4 回调行为

- `SetProgressCallback` 由 `/booster/audio/progress` 更新 `PlayerInfo`。
- `SetCompletionCallback` 只在状态首次变为 `kCompleted` 时触发。
- 错误 topic 会把缓存状态置为 `kError`，并触发 `SetErrorCallback`。
- 子对象内部会按 `client_id/session_type/session_id` 过滤，只处理当前 session 的消息。

9. AudioRecorder

9.1 适用场景

- 录制机器人端麦克风输入到文件。

9.2 接口列表

C++ 方法	Python 方法	说明
int32_t Init(const RecorderInitOptions &options)	Init(options)	初始化录音会话
int32_t Start()	Start()	开始录音
int32_t Pause()	Pause()	暂停录音
int32_t Stop()	Stop()	停止录音
int32_t Destroy()	Destroy()	销毁服务端录音会话
int32_t Release()	Release()	当前等同 Destroy()
int32_t GetInfo(RecorderInfo *out) const	GetInfo()	查询录音状态
int32_t SetPreferredDevice(const std::string &device_id)	SetPreferredDevice(device_id)	设置当前录音 session 的首选输入设备 [1.7]
int64_t GetSessionId() const	GetSessionId()	查询 session id
RecorderState GetCachedState() const	GetCachedState()	查询本地缓存状态
void SetStateCallback(RecorderStateCallback callback)	SetStateCallback(callback)	状态回调
void SetProgressCallback(RecorderProgressCallback callback)	SetProgressCallback(callback)	进度回调
void SetErrorCallback(RecorderErrorCallback callback)	SetErrorCallback(callback)	错误回调

9.3 使用要点

- `output_path` 必填，必须使用机器人端音频服务可写的绝对路径。
- `sample_rate_hz/channels/bits_per_sample` 都必须大于 0。
- `GetInfo().captured_bytes` 可用于估算录音长度。
- 对运行中的录音 session 切换输入设备，service 当前会拒绝运行态输入迁移，返回 `NOT_SUPPORTED`。
- 录音结束后建议显式 `Stop()`，再 `Destroy()` 或 `Release()`。

10. AudioLocalizer

10.1 适用场景

- 查询当前声源方位角，DOA，单位为度。

10.2 接口列表

C++ 方法	Python 方法	说明
int32_t GetDoaAngle(int *angle_deg) const	GetDoaAngle()	查询 DOA 角度

10.3 使用要点

- `AudioLocalizer` 不维护独立 session，不需要 `Init()`。

- DOA 能力依赖机器人端后端能力，通常只在搭载自研环麦(BoosterAEC) 输入链路的机器上可用。
- 不支持时可能返回 `LOCALIZATION_UNAVAILABLE` 或其他后端错误码。

11. AudioSystemController [[Deprecated]]

旧文档中的 `AudioSystemController` 当前不再是 SDK 公共 API。

旧用途：

- 调整系统主音量。
- 打开或关闭系统静音。

当前替代：

代码块

```
1 AudioManager manager;  
2 manager.Init();  
3 manager.SetSystemVolume(0.5F);  
4 manager.SetSystemMute(false);
```

代码块

```
1 manager = br.AudioManager()  
2 manager.Init()  
3 manager.SetSystemVolume(0.5)  
4 manager.SetSystemMute(False)
```

12. AudioCaptureStream

12.1 适用场景

- 订阅实时音频帧。
- 获取原始 raw PCM 数据。
- 获取 NAEC 处理后的 PCM 数据。
- 为 ASR、语音唤醒、自定义算法提供实时输入。

12.2 接口列表

C++ 方法	Python 方法	说明
int32_t Init(const AudioCaptureStreamOptions &options)	Init(options)	初始化采集流 session
int32_t Start()	Start()	开始推送采集帧
int32_t Pause()	Pause()	暂停采集
int32_t Stop()	Stop()	停止采集
int32_t Destroy()	Destroy()	销毁采集流 session
int32_t GetInfo(AudioCaptureStreamInfo *out) const	GetInfo()	查询采集状态
int32_t SetPreferredDevice(const std::string &device_id)	SetPreferredDevice(device_id)	设置当前采集流 session 的首选输入设备 [1.7]
int64_t GetSessionId() const	GetSessionId()	查询 session id
AudioCaptureStreamState GetCachedState() const	GetCachedState()	查询本地缓存状态
void SetFrameCallback(AudioCaptureFrameCallback callback)	SetFrameCallback(callback)	实时帧回调
void SetStateCallback(AudioCaptureStreamStateCallback callback)	SetStateCallback(callback)	状态回调
void SetErrorCallback(AudioCaptureStreamErrorCallback callback)	SetErrorCallback(callback)	错误回调

`Release()` [[Deprecated]] : 当前 `AudioCaptureStream` 不提供 `Release()` , 请使用 `Destroy()` 。

12.3 初始化参数

`AudioCaptureStreamOptions` :

- `enable_raw_pcm` : 是否启用 raw PCM , 默认 `true` 。
- `enable_naec_pcm` : 是否启用 NAEC PCM , 默认 `false` 。
- `requested_raw_format` : raw PCM 请求格式, 默认 `16000/1/16` 。

约束 :

- `enable_raw_pcm` 和 `enable_naec_pcm` 至少一个为 `true` 。
- 启用 raw PCM 时, `sample_rate_hz/channels/bits_per_sample` 都必须大于 0。
- 当前 SDK 对 raw PCM 请求的 `bits_per_sample` 只接受 `16` 。
- NAEC 依赖机器人端自研环麦/BoosterAEC 能力。

12.4 帧数据

`AudioCaptureFrame` 关键字段 :

- `frame_seq` : 帧序号。
- `timestamp_ms` : 服务端时间戳。
- `raw_valid` : raw PCM 是否有效。
- `raw_format` : raw PCM 实际格式。
- `raw_frame_samples_per_channel` : raw 每通道样本数。
- `raw_pcm` : raw PCM int16 样本数组。
- `naec_valid` : NAEC PCM 是否有效。
- `naec_format` : NAEC PCM 实际格式。
- `naec_frame_samples_per_channel` : NAEC 每通道样本数。
- `naec_pcm` : NAEC PCM int16 样本数组。

12.5 使用要点

- `Init()` 成功后 SDK 会根据 service 返回的 `data_topic_name` 创建独立 DDS subscriber。
- `Start()` 后状态可能先由控制 RPC 更新为 `kStreaming`，也可能在收到第一帧时变为 `kStreaming`。
- 对运行中的 capture session 切换输入设备，service 当前会拒绝运行态输入迁移。
- 如果采集流初始化时启用了 NAEC，选择普通 Pulse 输入设备可能返回 `NOT_SUPPORTED`。

13. RPC 映射

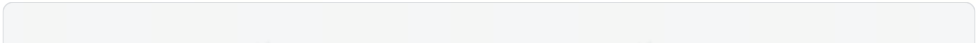
业务开发不应直接调用 `CallService`，但排查时可以用下表对齐 SDK 与 service：

能力	RPC channel
注册	rt/booster/audio/register_client
播放初始化	rt/booster/audio/init_player
播放开始	rt/booster/audio/start_player
播放暂停	rt/booster/audio/pause_player
播放停止	rt/booster/audio/stop_player
播放重置	rt/booster/audio/reset_player
播放销毁	rt/booster/audio/destroy_player
播放音量	rt/booster/audio/set_volume
播放信息	rt/booster/audio/get_player_info
PCM 流推送	rt/booster/audio/send_pcm_data
录音初始化	rt/booster/audio/init_recorder
录音开始	rt/booster/audio/start_recorder
录音暂停	rt/booster/audio/pause_recorder
录音停止	rt/booster/audio/stop_recorder
录音销毁	rt/booster/audio/destroy_recorder
录音信息	rt/booster/audio/get_recorder_info
DOA	rt/booster/audio/get_doa_angle
系统音量设置 [1.7]	rt/booster/audio/set_system_volume
系统音量查询 [1.7]	rt/booster/audio/get_system_volume
系统静音设置 [1.7]	rt/booster/audio/set_system_mute
系统静音查询 [1.7]	rt/booster/audio/get_system_mute
采集流初始化	rt/booster/audio/init_capture_stream
采集流开始	rt/booster/audio/start_capture_stream
采集流暂停	rt/booster/audio/pause_capture_stream
采集流停止	rt/booster/audio/stop_capture_stream
采集流销毁	rt/booster/audio/destroy_capture_stream
采集流信息	rt/booster/audio/get_capture_stream_info
设备查询 [1.7]	rt/booster/audio/get_devices
session 首选设备 [1.7]	rt/booster/audio/set_preferred_device
蓝牙扫描开始 [1.7]	rt/booster/audio/start_bluetooth_scan
蓝牙扫描停止 [1.7]	rt/booster/audio/stop_bluetooth_scan
蓝牙设备列表 [1.7]	rt/booster/audio/get_bluetooth_devices
蓝牙连接 [1.7]	rt/booster/audio/connect_bluetooth_device
蓝牙断开 [1.7]	rt/booster/audio/disconnect_bluetooth_device
蓝牙忘记设备 [1.7]	rt/booster/audio/forget_bluetooth_device

14. C++ 示例

本节示例参考 `~/boosteraec_demo/` 中的 `audio_play_demo.cpp`、`audio_record_demo.cpp`、`audio_interface_coverage_demo.cpp` 和 `audio_bluetooth_demo.cpp` 提炼。为便于阅读，部分示例省略了完整命令行解析；实际工程中应检查每个 `int32_t` 返回值，并在退出前显式 `Destroy()` 会话。

14.1 WAV 文件播放



```

1  #include <chrono>
2  #include <iostream>
3  #include <thread>
4
5  #include <booster/robot/audio/audio_manager.h>
6  #include <booster/robot/audio/audio_player.h>
7  #include <booster/robot/audio/audio_types.h>
8
9  int main() {
10     using namespace booster::robot::audio;
11
12     AudioManager manager;
13     int32_t ret = manager.Init();
14     if (ret != 0) {
15         std::cerr << "AudioManager::Init failed: " << ret << std::endl;
16         return ret;
17     }
18
19     auto player = manager.CreatePlayer();
20     player->SetStateCallback([](PlayerState state) {
21         std::cout << "player state=" << static_cast<int>(state) << std::endl;
22     });
23     player->SetErrorCallback([](const AudioError &err) {
24         std::cerr << "player error code=" << err.ret_code
25             << " msg=" << err.ret_msg << std::endl;
26     });
27
28     PlayerInitOptions options;
29     options.source_type = AudioSourceType::kWavFile;
30     options.source_uri = "/opt/booster/demo.wav";
31     options.priority = PlayerPriority::kMedium;
32
33     ret = player->Init(options);
34     if (ret != 0) {
35         std::cerr << "player init failed: " << ret << std::endl;
36         manager.Shutdown();
37         return ret;
38     }
39
40     ret = player->Start();
41     if (ret != 0) {
42         std::cerr << "player start failed: " << ret << std::endl;
43     }
44
45     std::this_thread::sleep_for(std::chrono::seconds(3));
46     player->Destroy();
47     manager.Shutdown();
48     return ret;
49 }

```

14.2 查询输出设备并设置播放器首选设备 [1.7]

代码块

```

1  using namespace booster::robot::audio;
2
3  AudioManager manager;
4  if (manager.Init() != 0) {
5      return;
6  }
7
8  std::vector<AudioDeviceInfo> outputs;
9  if (manager.GetDevices(AudioDeviceQueryType::GET_DEVICES_OUTPUTS, &outputs) == 0) {
10     for (const auto &device : outputs) {
11         std::cout << device.device_id << " " << device.display_name << std::endl;

```

```

12     }
13 }
14
15 auto player = manager.CreatePlayer();
16 // player->Init(...) 成功后才能设置当前 session 的 preferred device。
17 player->SetPreferredDevice("pulse_output_xxx");

```

14.3 录音到 WAV 文件

`output_path` 必须是机器人端音频服务可写的绝对路径。

代码块

```

1  #include <chrono>
2  #include <iostream>
3  #include <thread>
4
5  #include <booster/robot/audio/audio_manager.h>
6  #include <booster/robot/audio/audio_recorder.h>
7  #include <booster/robot/audio/audio_types.h>
8
9  int main() {
10     using namespace booster::robot::audio;
11
12     AudioManager manager;
13     if (manager.Init() != 0) {
14         return 1;
15     }
16
17     auto recorder = manager.CreateRecorder();
18     recorder->SetProgressCallback([](const RecorderInfo &info) {
19         std::cout << "captured_bytes=" << info.captured_bytes << std::endl;
20     });
21     recorder->SetErrorCallback([](const AudioError &err) {
22         std::cerr << "recorder error code=" << err.ret_code
23             << " msg=" << err.ret_msg << std::endl;
24     });
25
26     RecorderInitOptions options;
27     options.output_path = "/tmp/booster_record.wav";
28     options.sample_rate_hz = 16000;
29     options.channels = 1;
30     options.bits_per_sample = 16;
31
32     if (recorder->Init(options) != 0) {
33         manager.Shutdown();
34         return 1;
35     }
36     if (recorder->Start() != 0) {
37         recorder->Destroy();
38         manager.Shutdown();
39         return 1;
40     }
41
42     std::this_thread::sleep_for(std::chrono::seconds(5));
43     recorder->Stop();
44
45     RecorderInfo info;
46     if (recorder->GetInfo(&info) == 0) {
47         std::cout << "final captured_bytes=" << info.captured_bytes << std::endl;
48     }
49
50     recorder->Destroy();
51     manager.Shutdown();
52     return 0;

```

```
53 }
```

14.4 PCM 流式播放

本地文件分片读取后通过 `PushPcmStream()` 推给机器人端播放。这里的输入路径是运行 SDK 程序所在机器上的文件路径。

代码块

```
1  #include <chrono>
2  #include <cstdint>
3  #include <cstdint>
4  #include <fstream>
5  #include <thread>
6  #include <vector>
7
8  #include <booster/robot/audio/audio_manager.h>
9  #include <booster/robot/audio/audio_player.h>
10 #include <booster/robot/audio/audio_types.h>
11
12 int main() {
13     using namespace booster::robot::audio;
14
15     AudioManager manager;
16     if (manager.Init() != 0) {
17         return 1;
18     }
19
20     auto player = manager.CreatePlayer();
21
22     PlayerInitOptions options;
23     options.source_type = AudioSourceType::kPcmStream;
24     options.sample_rate_hz = 16000;
25     options.channels = 1;
26     options.bits_per_sample = 16;
27
28     if (player->Init(options) != 0 || player->Start() != 0) {
29         player->Destroy();
30         manager.Shutdown();
31         return 1;
32     }
33
34     std::ifstream input("/tmp/input.pcm", std::ios::binary);
35     std::vector<uint8_t> chunk(4096);
36     while (input) {
37         input.read(reinterpret_cast<char*>(chunk.data()),
38                   static_cast<std::streamsize>(chunk.size()));
39         const auto n = input.gcount();
40         if (n <= 0) {
41             break;
42         }
43         player->PushPcmStream(chunk.data(), static_cast<std::size_t>(n));
44         std::this_thread::sleep_for(std::chrono::milliseconds(20));
45     }
46
47     player->Stop();
48     player->Destroy();
49     manager.Shutdown();
50     return 0;
51 }
```

14.5 实时采集 raw PCM 帧

采集流可用于订阅 raw PCM / NAEC PCM。raw PCM 请求格式当前只支持 16-bit。

代码块

```

1  #include <chrono>
2  #include <iostream>
3  #include <thread>
4
5  #include <booster/robot/audio/audio_capture_stream.h>
6  #include <booster/robot/audio/audio_manager.h>
7  #include <booster/robot/audio/audio_types.h>
8
9  int main() {
10     using namespace booster::robot::audio;
11
12     AudioManager manager;
13     if (manager.Init() != 0) {
14         return 1;
15     }
16
17     auto capture = manager.CreateCaptureStream();
18     capture->SetFrameCallback([](const AudioCaptureFrame &frame) {
19         if (!frame.raw_valid) {
20             return;
21         }
22         std::cout << "frame_seq=" << frame.frame_seq
23             << " samples=" << frame.raw_pcm.size()
24             << " format=" << frame.raw_format.sample_rate_hz << "/"
25             << frame.raw_format.channels << "/"
26             << frame.raw_format.bits_per_sample << std::endl;
27     });
28
29     AudioCaptureStreamOptions options;
30     options.enable_raw_pcm = true;
31     options.enable_naec_pcm = false;
32     options.requested_raw_format = PcmFormat{16000, 1, 16};
33
34     if (capture->Init(options) != 0 || capture->Start() != 0) {
35         capture->Destroy();
36         manager.Shutdown();
37         return 1;
38     }
39
40     std::this_thread::sleep_for(std::chrono::seconds(3));
41     capture->Stop();
42     capture->Destroy();
43     manager.Shutdown();
44     return 0;
45 }

```

14.6 系统音量和设备查询 [1.7]

代码块

```

1  #include <iostream>
2  #include <vector>
3
4  #include <booster/robot/audio/audio_manager.h>
5  #include <booster/robot/audio/audio_types.h>
6
7  int main() {
8     using namespace booster::robot::audio;
9
10     AudioManager manager;
11     if (manager.Init() != 0) {
12         return 1;
13     }
14
15     float volume = 0.0F;

```

```
16     if (manager.GetSystemVolume(&volume) == 0) {
17         std::cout << "current volume=" << volume << std::endl;
18     }
19     manager.SetSystemVolume(0.5F);
20     manager.SetSystemMute(false);
21
22     std::vector<AudioDeviceInfo> inputs;
23     std::vector<AudioDeviceInfo> outputs;
24     manager.GetDevices(AudioDeviceQueryType::GET_DEVICES_INPUTS, &inputs);
25     manager.GetDevices(AudioDeviceQueryType::GET_DEVICES_OUTPUTS, &outputs);
26
27     for (const auto &device : outputs) {
28         std::cout << "output device_id=" << device.device_id
29                 << " name=" << device.display_name
30                 << " default=" << device.is_system_default << std::endl;
31     }
32
33     manager.Shutdown();
34     return 0;
35 }
```

14.7 蓝牙连接并指定播放器输出设备 [1.7]

蓝牙耳机默认建议使用 A2DP 播放链路；如果要使用耳机麦克风，应显式选择 HFP，并使用 `pulse_source_id` 配合录音或采集输入。

代码块

```
1  #include <chrono>
2  #include <iostream>
3  #include <thread>
4
5  #include <booster/robot/audio/audio_manager.h>
6  #include <booster/robot/audio/audio_player.h>
7  #include <booster/robot/audio/audio_types.h>
8
9  int main() {
10     using namespace booster::robot::audio;
11
12     AudioManager manager;
13     if (manager.Init() != 0) {
14         return 1;
15     }
16
17     BluetoothConnectOptions connect;
18     connect.address = "AA:BB:CC:DD:EE:FF";
19     connect.auto_pair = true;
20     connect.make_default = true;
21     connect.preferred_profile = BluetoothAudioProfile::kA2dpSink;
22     connect.timeout_ms = 15000;
23
24     BluetoothConnectResult result;
25     if (manager.ConnectBluetoothDevice(connect, &result) != 0 ||
26         result.pulse_sink_id.empty()) {
27         manager.Shutdown();
28         return 1;
29     }
30
31     auto player = manager.CreatePlayer();
32     PlayerInitOptions options;
33     options.source_type = AudioSourceType::kWavFile;
34     options.source_uri = "/opt/booster/demo.wav";
35     if (player->Init(options) == 0) {
36         player->SetPreferredDevice(result.pulse_sink_id);
37         player->Start();
38         std::this_thread::sleep_for(std::chrono::seconds(3));
39     }
```

```
40
41     player->Destroy();
42     manager.DisconnectBluetoothDevice(connect.address);
43     manager.Shutdown();
44     return 0;
45 }
```

15. Python 示例

15.1 初始化

代码块

```
1  #!/usr/bin/env python3
2
3  import booster_robotics_sdk_python as br
4
5  net_if = "eno1"
6  br.ChannelFactory.Instance().Init(0, net_if)
7
8  manager = br.AudioManager()
9  manager.Init()
10 print(manager.GetClientId())
```

15.2 播放 WAV 文件

代码块

```
1  import time
2  import booster_robotics_sdk_python as br
3
4  br.ChannelFactory.Instance().Init(0, "eno1")
5  manager = br.AudioManager()
6  manager.Init()
7
8  player = manager.CreatePlayer()
9  player.SetStateCallback(lambda state: print("player state", state))
10 player.SetErrorCallback(lambda err: print("player error", err.ret_code, err.ret_
11
12 options = br.PlayerInitOptions()
13 options.source_type = br.AudioSourceType.kWavFile
14 options.source_uri = "/opt/booster/demo.wav"
15 options.priority = br.PlayerPriority.kMedium
16
17 player.Init(options)
18 player.Start()
19 time.sleep(3)
20 player.Destroy()
21 manager.Shutdown()
```

15.3 PCM 流式播放

代码块

```
1  import time
2  import booster_robotics_sdk_python as br
3
4  br.ChannelFactory.Instance().Init(0, "eno1")
5  manager = br.AudioManager()
6  manager.Init()
7  player = manager.CreatePlayer()
8
9  options = br.PlayerInitOptions()
```

```
10 options.source_type = br.AudioSourceType.kPcmStream
11 options.sample_rate_hz = 16000
12 options.channels = 1
13 options.bits_per_sample = 16
14
15 player.Init(options)
16 player.Start()
17
18 with open("/tmp/input.pcm", "rb") as f:
19     while True:
20         chunk = f.read(4096)
21         if not chunk:
22             break
23         player.PushPcmStream(chunk)
24         time.sleep(0.02)
25
26 player.Stop()
27 player.Destroy()
28 manager.Shutdown()
```

15.4 蓝牙连接 [1.7]

代码块

```
1 import booster_robotics_sdk_python as br
2
3 br.ChannelFactory.Instance().Init(0, "eno1")
4 manager = br.AudioManager()
5 manager.Init()
6
7 options = br.BluetoothScanOptions()
8 options.timeout_ms = 30000
9 options.audio_only = True
10 manager.StartBluetoothScan(options)
11
12 devices = manager.GetBluetoothDevices(True)
13 for device in devices:
14     print(device.address, device.name, device.is_audio_sink, device.is_audio_source)
15
16 connect = br.BluetoothConnectOptions()
17 connect.address = "AA:BB:CC:DD:EE:FF"
18 connect.preferred_profile = br.BluetoothAudioProfile.kNone
19 connect.timeout_ms = 15000
20 connect.auto_pair = True
21 connect.make_default = True
22
23 result = manager.ConnectBluetoothDevice(connect)
24 print(result.active_profile, result.pulse_endpoint_ready)
```

15.5 交互式示例脚本

仓库提供交互式示例：

代码块

```
1 python3 python_example/sdk_pybind_audio_example.py <networkInterface>
```

常用命令：

代码块

```
1 manager_info
2 player_init wav /opt/booster/demo.wav
3 player_start
```

```
4  player_info
5  player_destroy
6
7  player_stream_init 16000 1 16
8  player_push /tmp/input.pcm 4096 20
9
10 rec_init /tmp/out.wav 16000 1 16
11 rec_start
12 rec_stop
13 rec_destroy
14
15 sys_volume
16 sys_volume 0.5
17 sys_mute on
18 sys_mute off
19
20 devices outputs
21 devices inputs
22 player_device <device_id>
23 rec_device <device_id>
24 cap_device <device_id>
25
26 doa
27
28 bt_scan 30000 on
29 bt_devices true
30 bt_connect <address> none 15000 true true
31 bt_disconnect <address>
32 bt_forget <address>
33
34 cap_init raw 16000 1 16
35 cap_start
36 cap_info
37 cap_stop
38 cap_destroy
```

16. 接口选择建议

场景	推荐接口
播放机器人端文件	AudioManager.CreatePlayer() + AudioPlayer.Init(kWavFile/kMp3File/kPcmFile)
播放本地生成的 PCM 数据	AudioPlayer.Init(kPcmStream) + PushPcmStream()
录音到机器人端文件	AudioManager.CreateRecorder() + AudioRecorder
获取实时 PCM 帧	AudioManager.CreateCaptureStream() + AudioCaptureStream.SetFrameCallback()
查询声源方向	AudioManager.CreateLocalizer() + AudioLocalizer.GetDoaAngle()
调整系统音量/静音 [1.7]	AudioManager.SetSystemVolume/GetSystemVolume/ SetSystemMute/GetSystemMute
查询输入/输出设备 [1.7]	AudioManager.GetDevices()
切换某个 session 的设备 [1.7]	AudioPlayer/AudioRecorder/ AudioCaptureStream.SetPreferredDevice()
蓝牙扫描/连接/断开 [1.7]	AudioManager 蓝牙方法

17. 常见注意事项

- Python 绑定当前使用 PascalCase 方法名，不支持旧 camelCase 名。
- 先 Init()，后 Start()；对象未初始化或 session id 无效时会返回 kRpcStatusCodeInvalid。
- SetPreferredDevice() [1.7] 不是系统默认设备设置，而是当前 session 的首选设备设置。

- `player_device <device_id>` 必须在 `player_init` 或 `player_stream_init` 成功之后调用。
- `rec_device` 和 `cap_device` 应使用 `devices inputs` 返回的设备 id。
- `player_device` 应使用 `devices outputs` 返回的设备 id。
- 运行中的 recorder/capture input migration 当前不支持，需先停止/销毁后重新初始化到目标设备。
- raw capture 的 `bits_per_sample` 当前只支持 16。
- `AudioCaptureStream` 没有 `Release()`；用 `Destroy()`。
- `GetCachedState()` 只读本地缓存；需要服务端状态时使用 `GetInfo()`。
- 机器人连接蓝牙音频设备的能力暂不支持极客版。
- 机器人对部分荣耀耳机连接可能会失败，遇到配对或 profile 连接超时时应优先换设备交叉验证。
- 机器人连接蓝牙耳机默认使用 A2DP 模式，只能用于播放音频；如需使用耳机麦克风，需要在 `BluetoothConnectOptions.preferred_profile` 中指定 HFP 模式，例如 C++ `BluetoothAudioProfile::kHfpHeadset` 或 Python `br.BluetoothAudioProfile.kHfpHeadset`。
- 机器人关机后再次开机，如果耳机自动回连，仍会按 A2DP 模式回连，不会自动切换为 HFP；如果需要使用耳机麦克风，请在应用中先 `DisconnectBluetoothDevice()`，再以 HFP 模式重新 `ConnectBluetoothDevice()`。
- 对长时间任务建议同时注册状态回调和错误回调。
- 程序退出前建议显式 `Destroy()` 各 session，再 `AudioManager.Shutdown()`。