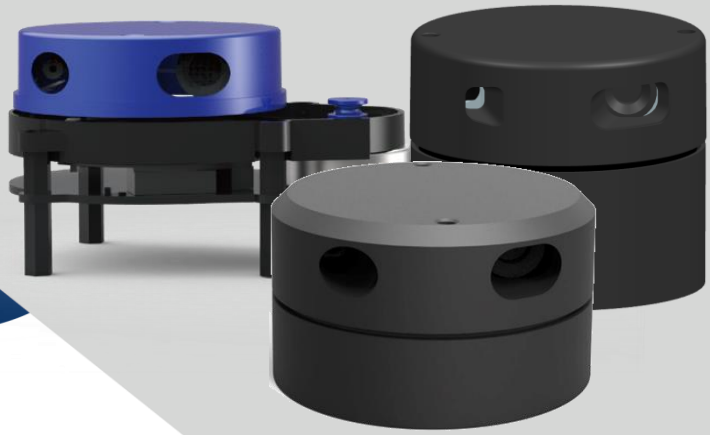
YDLIDAR SDK

使用手册



文档编码：01.13.000020

目录

YDLIDAR SDK 常用接口函数

YDLIDAR 在 Linux 下的驱动类 YDlidarDriver 的常用接口如下：

表 1 YDLIDAR SDK LINUX API

项目	修订内容
创建实例	static void initDriver()
获取实例	static YDlidarDriver* singleton()
销毁实例	static void done()
打开串口	result_t connect(const char * port_path, uint32_t baudrate);
关闭串口	void disconnect();
获取状态信息	result_t getHealth(device_health & health, uint32_t timeout = DEFAULT_TIMEOUT);
获取设备信息	result_t getDeviceInfo(device_info & info, uint32_t timeout = DEFAULT_TIMEOUT);
开始扫描	result_t startScan(bool force = false, uint32_t timeout = DEFAULT_TIMEOUT) ;
停止扫描	result_t stop();
抓取一圈雷达数据	result_t grabScanData(node_info * nodebuffer, size_t & count, uint32_t timeout = DEFAULT_TIMEOUT);
获取 SDK 版本	const std::string getSDKVersion();
设备重置	result_t reset(uint32_t timeout = DEFAULT_TIMEOUT);
获取扫描频率	result_t getFrequency(uint32_t model, size_t count, float & frequency);
获取采样率	result_t getSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT);
设置采样率	result_t setSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT);

注：result_t 为 int 的宏定义。

YDLIDAR SDK 描述

创建实例

```
static void initDriver()
```

initDriver ()静态方法创建驱动实例，没有返回值。

获取驱动实例

```
static YDlidarDriver* singleton()
```

singleton ()获取驱动实例，返回值便是驱动实例指针。

销毁实例

```
  static void done()
```

done ()销毁驱动实例，没有返回值。

打开串口

```
  result_t connect(const char * port_path, uint32_t baudrate);
```

connect ()传参为串口名称与波特率（X4 默认为 128000，F4 默认为 153600，G4 默认为 230400），返回值不为-1 时，说明打开串口成功。

关闭串口

```
  void disconnect();
```

disconnect()关闭串口，没有返回值。

获取设备状态信息

```
  result_t getHealth(device_health & health, uint32_t timeout =
DEFAULT_TIMEOUT);
```

device_health 为设备状态结构体，getHealth()传参为状态结构体实例便可，返回值有 0、-1 与-2，当返回值为 0 时说明获取数据正确，为-1 时说明获取数据失败，为-2 时说明获取数据超时。

获取设备信息

```
  result_t getDeviceInfo(device_info & info, uint32_t timeout =
DEFAULT_TIMEOUT);
```

device_info 为设备信息结构体，getDeviceInfo ()传参为设备信息结构体实例便可，返回值有 0、-1 与-2，当返回值为 0 时说明获取数据正确，为-1 时说明获取数据失败，为-2 时说明获取数据超时。

开始扫描

```
  result_t startScan(bool force = false, uint32_t timeout =
DEFAULT_TIMEOUT) ;
```

startScan ()不用传参，返回值有 0、-1 与-2，当返回值为 0 时说明雷达开始扫描成功，为-1 时说明发送扫描命令失败，为-2 时说明发送扫描命令超时。

停止扫描

```
  result_t stop();
```

stop ()不用传参，返回值有 0、-1 与-2，当返回值为 0 时说明雷达停止扫描成功，为-1 时说明发送扫描命令失败，为-2 时说明发送扫描命令超时。

抓取一圈雷达数据

```
  result_t grabScanData(node_info * nodebuffer, size_t & count, uint32_t timeout = DEFAULT_TIMEOUT);
```

node_info 为雷达数据结构体，grabScanData ()传参为雷达数据结构体实例与一圈雷达数据个数，返回值有 0、-1 与-2，当返回值为 0 时说明获取数据成功，为-1 时说明获取数据失败，为-2 时说明获取数据超时。

获取 SDK 版本

```
  const std::string getSDKVersion();
```

getSDKVersion()不用传参，返回值为 SDK 版本号。

设备重置

```
  result_t reset(uint32_t timeout = DEFAULT_TIMEOUT);
```

reset()不用传参，返回值为 0 时，设备重置成功。

获取扫描频率

```
  result_t getFrequency(uint32_t model, size_t count, float & frequency);
```

getFrequency()的参数为 model 雷达型号与雷达扫描一圈的有效点数 count，获取值 frequency 为对应型号雷达的扫描频率。返回值为 0 时，获取数据成功。

获取采样率

```
  result_t getSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT);
```

getSamplingRate()的参数为采样率结构体 sampling_rate，返回值为 0 时，获取数据成功。

设置采样率

```
  result_t setSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT);
```

setSamplingRate()的参数为采样率结构体 sampling_rate，返回值为 0 时，设置成功。

修订

日期	版本	修订内容
2017-12-08	1.0	初撰