四维视觉 SDK 技术手册

第一章 产品概述

在机器视觉系统中，光源的主要目的是将待检测物体与背景之间产生足够强的对比度，形成有利于图像处理的成像效果；因此，选择合适的光源将帮助系统更好地识别目标，在一定程度上降低系统的图像处理算法复杂度，提高系统整体性能。

1.1 光源颜色的重要性和颜色混合

光源颜色会影响光照目标表面的反射特性，合理的光照颜色可以使目标特征与周围区域产生足够的灰度值差异。例如，在视觉应用中的“同色打白，异色打黑”，就是利用了互补色原理，能更好地突出物体的特征与细节，提高图像的对比度和清晰度。图 1-1 是三原色与颜色混合原理的示意图，图 1-2 展示了在不同颜色下的成像效果。

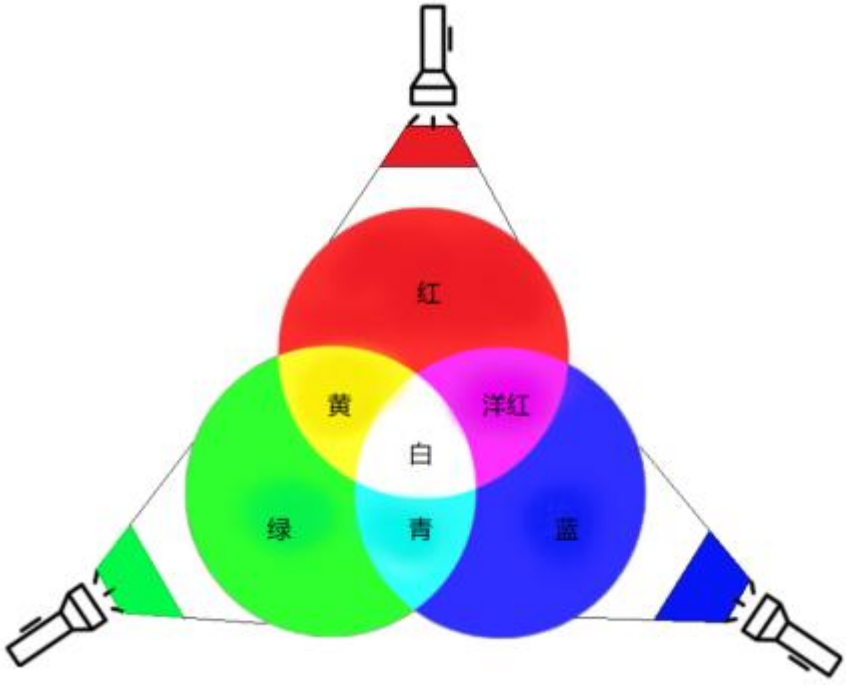


图 1-1 三原色与颜色混合

四维视觉的光源产品可以使用多种颜色混合，实现全色域颜色的生成，方便用户在实际的生产、科研中更好地利用光源颜色解决各类型问题，例如需要明暗对比强烈或复杂背景环境的机器视觉任务。

图 1-2 在自然光、以及蓝色光源和红色光源光照下的效果

1.2 光源方向的重要性

在机器视觉中，光照方向也同样影响着图像质量和图像特征提取效果。如图 1-3 所示，不同的光照方向会影响仪器设备对检测目标反射光的接收；明视野是直接利用目标的反射光，有利于对整体轮廓的提取，以及对高对比度的需求；暗视野则是利用目标的散射光，有利于对表面质量和纹理的提取；从图 1-4 可以明显地看出不同光照方向对成像的影响。

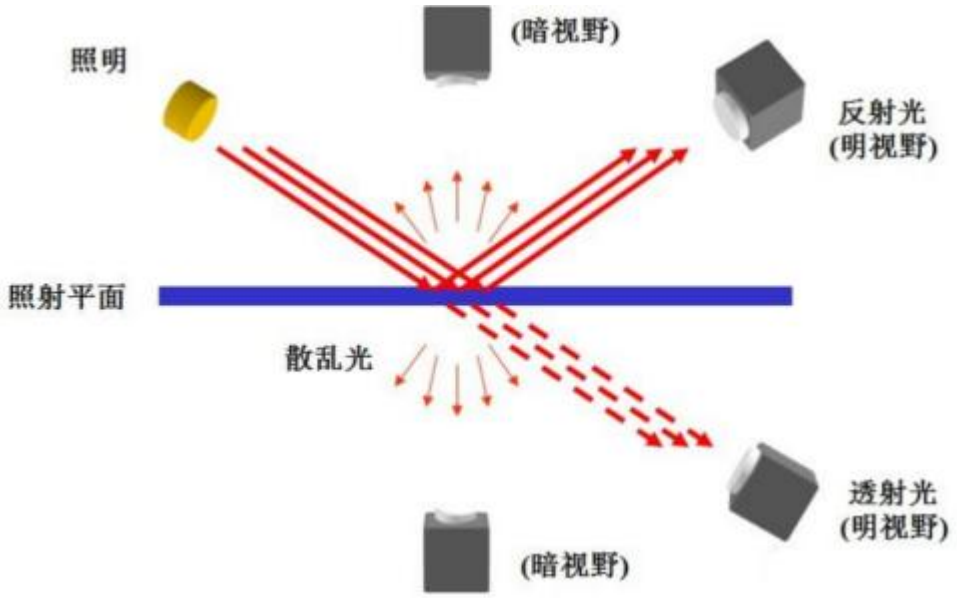


图 1-3 光照方向产生的明视野与暗视野

正因为光照方向对视觉检测的重要性，四维视觉提供了环形、条形、球顶等各种样式的光源，以尽可能满足用户对不同光照方向的需求。

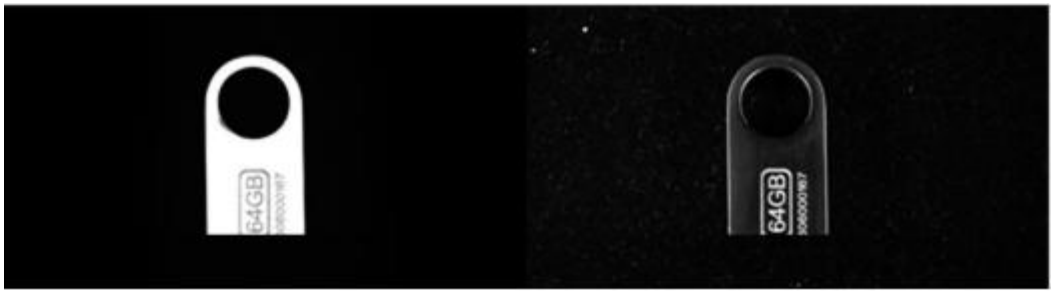


图 1-4 物体在明视野和暗视野的成像效果

1.3 四维视觉的光源产品

四维视觉目前研发了各类型的光源产品，包括环形光源、同轴光源、条形光源、球顶光源等，如图 1-5 所示。

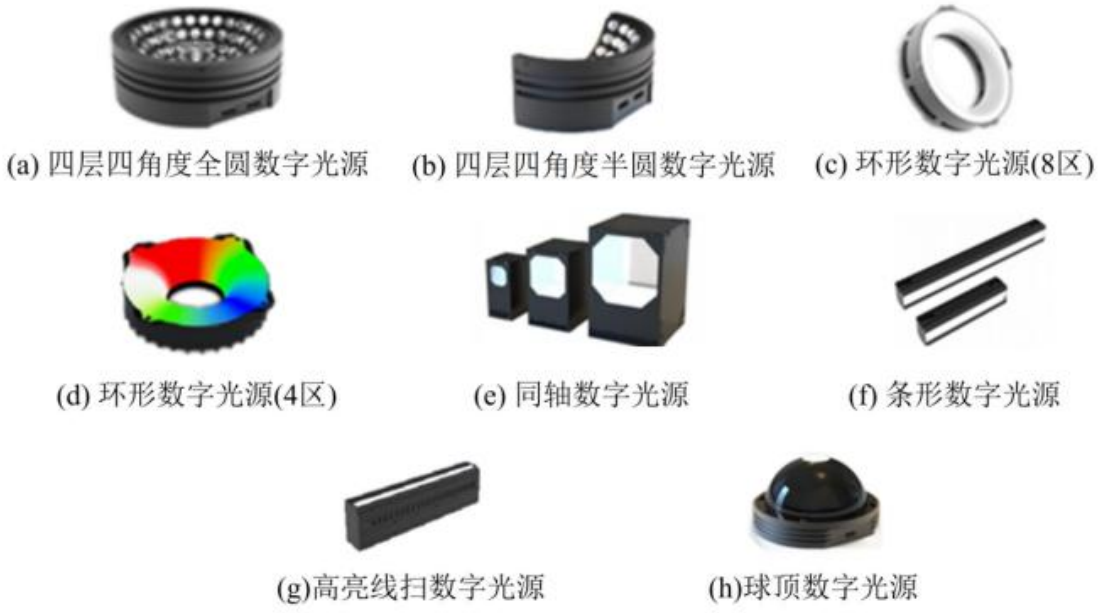


图 1-5 四维视觉的光源产品

1.3.1 总体参数

各类型光源的总体参数参见表 1-1，其中分区数目代表该类型光源能独立控制的区域数目，分区方式见 1.3.3 以及第二章的介绍。

产品类型	分区数目	颜色数目	分区方式	工作电压
四层四角度全圆数字光源	4	4	0/1/2	24V
四层四角度半圆数字光源	4	4	0/1/2	24V
环形数字光源(8区)	8	8	0/1/2	24V
环形数字光源(4区)	4	4	0/1/2	24V
同轴数字光源	1	4	0	24V
条形数字光源	1	4	0	24V
高亮线扫数字光源	1	2	0	48V
球顶数字光源	1	4	0	24V
	4	4	0/1/2	

表 1-1 四维视觉产品的总体参数

1.3.2 光源 UID 与光源地址

光源 UID 是指光源设备的唯一标识符，光源地址是指用于控制光源的标识序号，如图 1-6 所示。通过 UID 可以唯一确认光源设备，变更该设备的地址；通过地址则可以完成对光源的绝大部分操作。

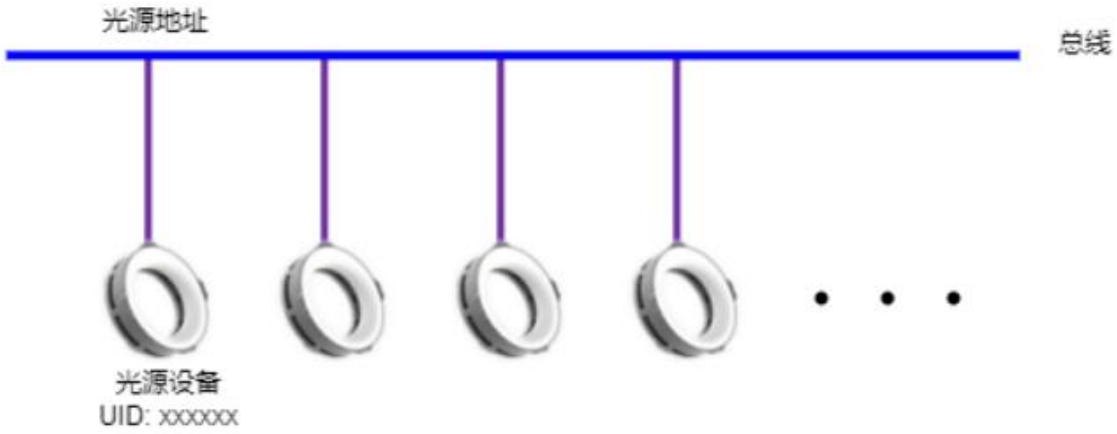


图 1-6 光源设备与地址

四维视觉光源的地址范围为 1~32，默认的初始地址为 1。在一条总线上支持最多 32 台光源设备。用户可以借助地址对光源进行分组操作和管理，例如，将同类型的多个光源挂载在同一个地址，通过该地址就可以同时控制这些光源了。在修改地址时应避免不同类型的光源挂载在同一地址，否则极有可能会对后续的调光、保存场景等操作失效。

1.3.3 分区与分区方式

分区是指可以独立进行控制的光源区域，分区方式是指如何对分区进行控制。以环形数字光源(4 区)为例，见表 1-2(示例中的亮灯顺序和颜色不代表实际应用，亮度、颜色等可以根据实际产品型号自行设置)。该类型光源共有 4 个分区均分整个圆环。在分区方式为 0 时，是将 4 个分区视为一个整体。在分区方式为 1 时，每个区都可以设置单独的亮度、颜色，亮度接近 0 时可视为灭灯，每次都可以选择亮 1~4 任意分区。在分区方式为 2 时，只能控制 4 个分区中的一个进行亮度和颜色的设置，其它 3 个区则灭灯。理论上，分区方式 1 可以实现对分区方式 0 和分区方式 2 的功能实现，但在具体的一些实践中，使用分区方式 0 或 2 会更方便地进行调光等操作。具体产品型号的分区介绍请参见第二章以及产品手册。

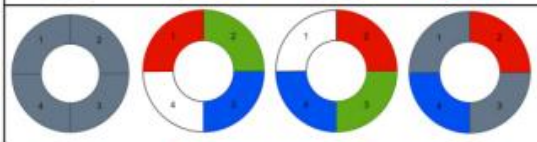
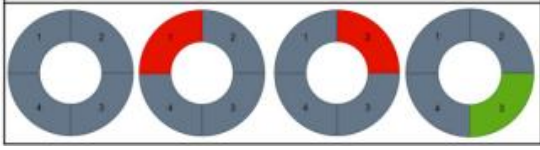
	分区方式0，将4个分区视为一个整体。左图为灭灯->对光源设置各种颜色(可设置颜色限制以具体产品型号为准)。
	分区方式1，每个分区可以进行单独设置。左图为灭灯->同时对每个分区设置亮度、颜色。
	分区方式2，只能对某一分区进行设置。左图为灭灯->某一分区设置亮度、颜色。

表 1-2 环形数字光源(4 区)的分区与分区方式示例

1.3.4 调光

调光是指设置光源的分区方式、亮度和颜色。分区方式如前所述。亮度是光源的总体亮度值，参数范围是 0~1024。颜色的支持数目依产品型号有所差异，比如环形数字光源(4 区)支持红、蓝、绿、白 4 种基础颜色以及对这 4 种颜色的混色，参见表 1-1 和具体的产品手册；每种颜色同样可以设置自己的亮度值，参数范围是 0~1024。颜色的最终亮度由以下公式获得：

$$\frac{B}{1024} \times C$$

其中 B 为总体亮度值，C 为颜色亮度值。例如亮度(B)取值为 512，某颜色的亮

度值(C)为 256，则该颜色的最终亮度为(512/1024)*256=128。

1.3.5 场景与工作模式

一般情况下，需要多次调整光源的分区方式、亮度和颜色，直到调光效果满足特定的任务需求，很多时候还会希望将不同的调光效果以某种形式重现。这里就涉及到了场景、模式和触发的概念，场景即为想要保存的调光效果(分区、亮度、颜色等)，模式即为如何重现这些场景，而触发则是通知光源该执行某一个场景了(详见后续说明)。四维视觉的光源设备支持最多 8 个自定义场景，且可以修改各个场景和场景总数；有 4 种工作模式供用户选择使用；通过高/低电平来触发对工作模式下场景的执行。

场景由分区方式、亮度、颜色和(或)亮灯/灭灯持续时间组成，描述了光源在触发时所呈现出来的光照效果。可以依据实际产品型号对场景进行设置，在多场景工作模式下还可将不同场景搭配使用(关于亮灯/灭灯持续时间、多场景工作模式详见后续介绍)。如图 1-7 所示，对环形数字光源(4 区)设置了 3 个场景，分别为分区方式 1，各分区颜色红绿蓝白；分区方式 0，整体为蓝色；分区方式 2，分区序号 2 为绿色。

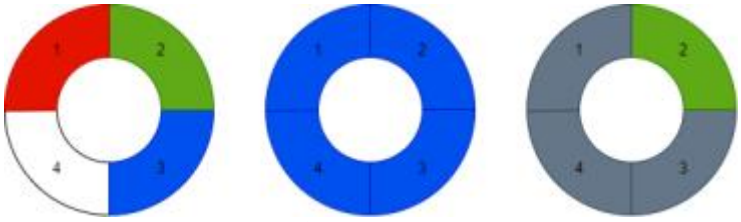


图 1-7 对环形数字光源(4 区)设置的 3 个场景

4 种工作模式分别为跟随模式(Pu-C)、单场景模式(Pu-P)、多场景连续模式(TTL_S)和多场景点动模式(TTL_C)。Pu-C 是指光源在检测到有效电平时亮灯，反之则灭灯。Pu-P 是指光源在检测到有效电平时保持亮灯至预设的持续时间(保存该场景时需要设置亮灯持续时间)，然后灭灯，等待下一次的有效电平。TTL_S 是指光源在检测到有效电平时，连续执行一遍所有以保存的场景，然后灭灯并等待下一次有效电平；TTL_S 工作模式下的各场景需要设置亮灯持续时间和灭灯持续时间。TTL_C 是指光源在检测到有效电平时执行一个已保存的场景，然后灭灯，当下一次的有效电平触发，执行下一个场景；TTL_C 工作模式下的各场景同样需要设置亮灯持续时间和灭

灯持续时间。当某个有效电平在某场景亮灯/灭灯持续时间结束之前到达，将忽略本次电平触发。从图 1-8 中可以更直观的了解 4 种工作模式。

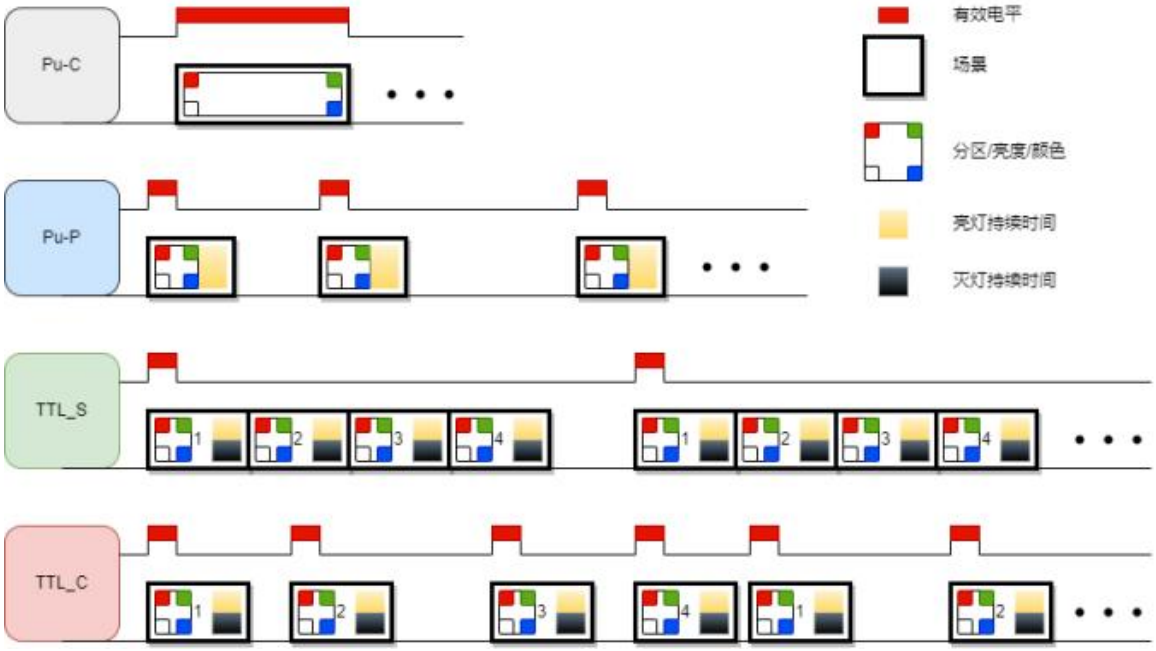


图 1-8 四种工作模式示意

在 Pu-C 和 Pu-P 工作模式下，因为只有一个场景，所以不必关心场景序号和场景总数。在 TTL_S 和 TTL_C 工作模式下，每次都是从序号 1 的场景开始，顺序执行直到场景总数，然后再次从序号 1 开始。

触发即为高电平触发或低电平触发。

1.3.6 保存场景与进入工作模式

保存场景即为将场景(见 1.3.5)保存为上述的 4 种工作模式之一。Pu-C 只包含一个场景，需保存该场景的触发电平方式。Pu-P 同样只包含一个场景，除了保存触发电平，还需要保存亮灯持续时间，单位为微秒，最大值为 65535。TTL_S 和 TTL_C 需要保存触发电平，亮灯持续时间和灭灯持续时间，因为涉及到了多个场景，还需要保存场景序号和场景总数。在保存场景为各工作模式时需要关心的参数见表 1-3。例如将 4 个场景保存为 TTL_S，这里需要设置各场景的分区、亮度和颜色，触发电平，各场景的亮灯持续时间和灭灯持续时间，场景序号(此例为 1、2、3、4)，场景总数(此例为 4)。保存场景通过光源地址来进行操作，要注意合理安排场景序号与场景总数，一般情况下不应出现“空洞”，比如设置了场景总数为 4，但场景序号仅有 1、

2、3。

工作模式	调光效果(分区、亮度、颜色)	触发电平(高/低电平)	亮灯持续时间(微秒，最大65535)	灭灯持续时间(微秒，最大65535)	场景序号(1到场景总数)	场景总数(最多8个)
跟随模式(Pu-C)	●	●				
单场景模式(Pu-P)	●	●	●			
多场景连续模式(TTL_S)	●	●	●	●	●	●
多场景点动模式(TTL_C)	●	●	●	●	●	●

表 1-3 保存场景时需要关心的参数

Pu-C/Pu-P 的场景和 TTL_S/TTL_C 的场景是独立存储的，Pu-C 和 Pu-P 共用一个场景，TTL_S 和 TTL_C 共用一组场景，如图 1-9 所示。

进入工作模式是指在保存场景成功后进入相应的工作模式。因 Pu-C 和 Pu-P 公用一个场景，所以支持保存场景为 Pu-P 工作模式，而选择进入 Pu-C 工作模式的这种操作，反之亦可；同理，TTL_S 和 TTL_C 工作模式也可执行类似的操作。若 Pu-C/Pu-P 和 TTL_S/TTL_C 都成功保存过，则可以随意选择进入哪种工作模式。例如将 4 个场景保存为 TTL_S 工作模式后，进入 TTL_S 工作模式，待检测到有效电平时将按照场景序号顺序执行已保存的各个场景，也可以选择进入 TTL_C 工作模式，则按照 TTL_C 工作模式的规则执行各个场景。若在此之前已经保存过 Pu-C 或 Pu-P 工作模式，则可以选择进入任意的工作模式。若在 TTL_C 下再次进入 TTL_C，当检测到下次有效电平时，将从场景序号 1 重新开始，而不管上次电平触发时执行的场景序号是多少。即在 TTL_C 工作模式下重新进入 TTL_C 工作模式的这种操作具有重置的功能。

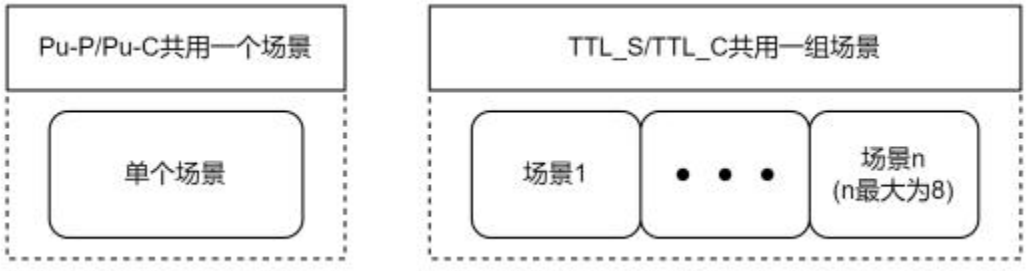


图 1-9 四种工作模式的场景存储关系

第二章 各型号产品的分区方式介绍

本章主要介绍各型号光源的分区与分区方式，部分光源有多个分区，部分光源只有一个分区。对于存在多个分区的光源，需要考虑不同分区方式对发光形式的影响。以下多个示例中所展示的发光颜色等不代表实际应用，用户可根据具体光源依照实际情况进行设置。

2.1 四层四角度全圆数字光源

该型号光源被同心环分为四个区域，共有三种分区方式 0，1，2，见表 2-1。

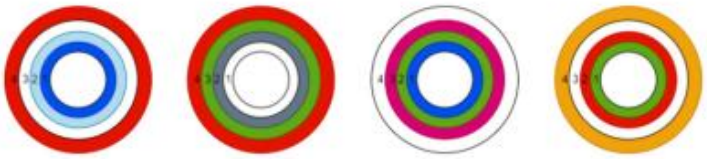
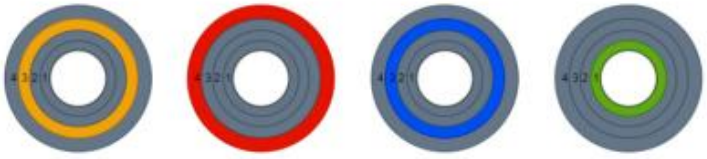
分区方式0		分区方式0将该类型光源的4个分区视为一个整体。
分区方式1		分区方式1可对该类型光源的每个分区进行单独设置，每个分区的亮度、颜色都互不影响。
分区方式2		分区方式2只能对该类型光源的某一分区进行设置，其它分区则保持灭灯。

表 2-1 四层四角度全圆数字光源的分区方式

2.2 四层四角度半圆数字光源

该型号光源共有三种分区方式 0，1，2，见表 2-2。除外形不同外，在分区方式上等同于四层四角度全圆数字光源(见 2.1)。

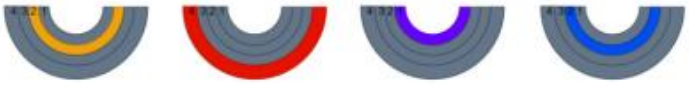
分区方式0		分区方式0将该类型光源的4个分区视为一个整体。
分区方式1		分区方式1可对该类型光源的每个分区进行单独设置，每个分区的亮度、颜色都互不影响。
分区方式2		分区方式2只能对该类型光源的某一分区进行设置，其它分区则保持灭灯。

表 2-2 四层四角度半圆数字光源的分区方式

2.3 环形数字光源(8 区)

该型号光源被均分为八个区域，共有三种分区方式 0，1，2，见表 2-3。

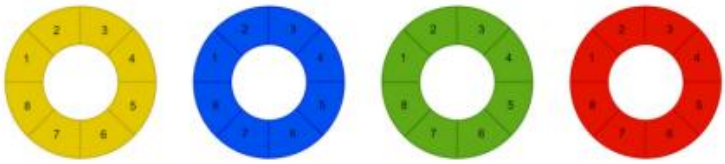
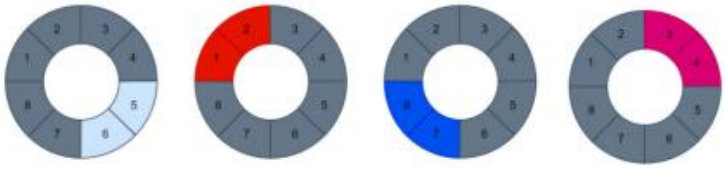
分区方式0		分区方式0将该类型光源的8个分区视为一个整体。
分区方式1		分区方式1将每相邻的两个分区视为一个分区，只能对该”合并”的区域进行设置，其它的则保持灭灯。
分区方式2		分区方式2只能对该类型光源的某一分区进行设置，其它分区则保持灭灯。

表 2-3 环形数字光源(8 区)的分区方式

2.4 环形数字光源(4 区)

该类型光源被均分为四个区域，共有三种分区方式 0，1，2，见表 2-4。除了均分整个圆环的形式不一样外，在分区方式上等同于四层四角度全圆数字光源(见 2.1)。

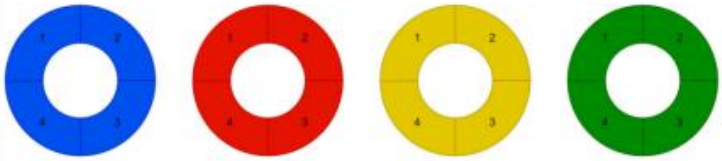
分区方式0		分区方式0将该类型光源的4个分区视为一个整体。
分区方式1		分区方式1可对该类型光源的每个分区进行单独设置，每个分区的亮度、颜色都互不影响。
分区方式2		分区方式2只能对该类型光源的某一分区进行设置，其它分区则保持灭灯。

表 2-4 环形数字光源(4 区)的分区方式

2.5 同轴数字光源

该类型光源是单分区的，发光区域视为一个整体，如图 2-1。

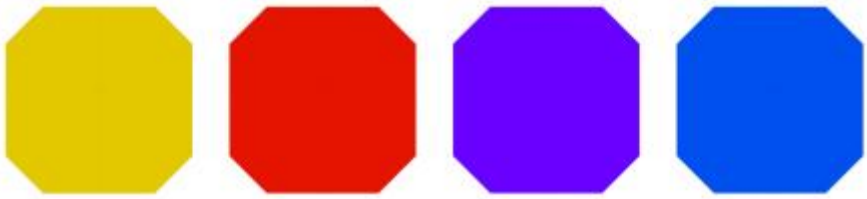


图 2-1 同轴数字光源

2.6 条形数字光源

该类型光源是单分区的，发光区域视为一个整体，如图 2-2。

图 2-2 条形数字光源

2.7 高亮线扫数字光源

该类型光源是单分区的，发光区域视为一个整体，如图 2-3。注意：该型号光源由两个 LED 实现暖色和冷色，但在 SDK(C++)中设置 LightParam 的 colors 成员变量(参见 3.3 小节和表 3-4)时只需填写一个数值代表色温。

图 2-3 高亮线扫数字光源

2.8 球顶数字光源

该类型光源分为单分区和四分区两种，四分区的球顶数字光源类似于上述的环形光源(4 区)，见表 2-5。

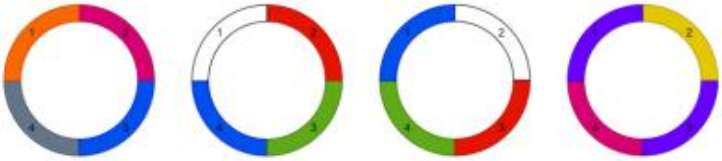
单分区		单分区的球顶数字光源。
分区方式0		分区方式0将该类型光源的4个分区视为一个整体。
分区方式1		分区方式1可对该类型光源的每个分区进行单独设置，每个分区的亮度、颜色都互不影响。
分区方式2		分区方式2只能对该类型光源的某一分区进行设置，其它分区则保持灭灯。

表 2-5 球顶数字光源

第三章 SDK 接口功能描述

SW_SDK 是四维视觉面向我司光源客户提供的开发工具包，以帮助开发者方便地实现对光源的配置与控制。该 SDK 采用 C++11 开发，编译器为 MSVC v143，支持 Windows 10/11 平台，提供 C 和 C++接口，运行时需要 Visual C++运行时库，比如 Microsoft Visual C++ 2015-2022 Redistributable(x64)。使用 C#，Python 等开发语言的项目请使用 C 接口。

3.1 C++接口

3.1.1 枚举类型

```
/*
 * ModelType是一个枚举类型，代表工作模式类型，其中，
 * @PULSE_C、@PULSE_P为单场景模式；
 * @TTL_S、@TTL_C为多场景模式。
 */
enum class ModelType : uint8_t
{
    PULSE_C = 0,
    PULSE_P = 1,
    TTL_S = 2,
    TTL_C = 3
};

/*
 * TTL是一个枚举类型，代表触发电平，其中，
 * @TTL_L为低电平触发，@TTL_H为高电平触发。
 */
enum class TTL : uint8_t
{
    TTL_L = 0,
    TTL_H = 1
};
```

3.1.2 结构体

```
/*
 * LightParam是一个结构体，包含了一些用于描述分区和调光等信息的成员变量；
 * @zoneMode为分区方式，有效范围请参考技术文档；
 * @currentZone代表当前操作的分区序号，范围为1到该产品类型的最大分区数；
 * @brightness表示当前分区的亮度，范围为0~1024；
 * @colors表示当前分区的颜色，每个通道的取值的范围为0~1024。
 */
struct LightParam
{
    uint8_t zoneMode;
    uint8_t currentZone;
    uint16_t brightness;
    std::vector<uint16_t> colors;
};

/*
 * DevInfo是一个结构体，包含了一些用于表示光源设备信息的成员变量；
 * @UID代表光源设备唯一标识符，该标识符大小为6-bytes；
 * @address为当前的光源设备地址，参数范围为1~32；
 * @productModel表示产品型号；
 * @zoneSize表示该产品型号所支持的最多分区数；
 * @colorSize表示该产品型号所支持的最多颜色数。
 */
struct DevInfo
{
    std::vector<uint8_t> UID;
    uint8_t address;
    std::string productModel;
    uint16_t zoneSize;
    uint16_t colorSize;
};
```

3.1.3 方法

3.1.3.1 初始化

```
/*
 * init()方法主要用于连接端口，获取光源设备信息等；
 * @comName为待连接的端口名称，如"COM3"；
 * 连接成功后将花费数秒至数十秒搜索和读取所有光源设备的信息，保存在@devInfos中，耗时与光源数目有关；
 * 请根据获得的@devInfos的实际情况进行下一步操作。
 */
bool init(const std::string &comName, std::vector<DevInfo> &devInfos);
```

3.1.3.2 修改地址

```
/*
 * changeAddr()方法用于修改光源设备的地址；
 * @UID为光源设备的唯一标识符，可以参考从init()获取的devInfos，其中包含了搜索到的光源设备UID；
 * @addr为光源设备的新地址，范围为1~32；
 * 修改地址成功后，请自行更新devInfos中光源的地址信息；
 * 请避免将不同类型光源置于同一地址，否则可能后续的setZoneAndColors()等对不同类型光源无效。
 */
bool changeAddr(const std::vector<uint8_t> &UID, uint8_t addr);
```

3.1.3.3 调光

```
/*
 * setZoneAndColors()方法用于显示当前的调光效果；
 * @lightParam为LightParam结构体实例，需要在调用该方法之前定义并设置各成员变量的数值；
 * @addr为该方法作用的光源设备地址，范围为1~32；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐使用默认值。
 * 请依据具体的光源型号合理设置LightParam的各成员变量。
 */
bool setZoneAndColors(const LightParam &lightParam, uint8_t addr, size_t delayMS = 10);
```

3.1.3.4 保存场景

```
/*
 * saveScene()方法用于保存场景；
 * @modelType代表工作模式类型，可输入参数为ModelType::PULSE_C，ModelType::PULSE_P，
ModelType::TTL_S或ModelType::TTL_C；
 * @ttl代表触发电平类型，可输入参数为TTL::TTL_L或TTL::TTL_H；
 * @addr为地址，范围1~32；
 * @sceneNo为该场景的序号，以1为起始，范围为1~@sceneSize，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C
和Pu-P工作模式下置1即可；
 * @sceneSize为场景的总数，最多支持8个场景，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C和Pu-P工作模式下
置1即可；
 * @onDur为亮灯持续时间，单位为微秒，范围为1~65535，在Pu-P，TTL_S和TTL_C工作模式下该参数有效，在Pu-C工
作模式下置1即可；
 * @offDur为灭灯持续时间，单位为微秒，范围为0~65535，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C和Pu-P
工作模式下置0即可；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐使用默认值。
 */
bool saveScene(ModelType modelType, TTL ttl, uint8_t addr, uint8_t sceneNo, uint8_t
sceneSize, uint16_t onDur, uint16_t offDur, size_t delayMS = 100);
```

3.1.3.5 进入工作模式

```
/*
 * enterWorkModel()方法用于进入工作场景；
 * @modelType代表工作模式类型，可输入参数为ModelType::PULSE_C，ModelType::PULSE_P，
ModelType::TTL_S或ModelType::TTL_C；
 * @addr为地址，范围1~32；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐使用默认值。
 * 该方法通常在成功调用saveScene()之后调用；
 * 当处于TTL_C工作模式下再次调用enterWorkModel(ModelType::TTL_C, addr)时，将重置为从场景1开始执行，
具体请见文档。
 */
bool enterWorkModel(ModelType modelType, uint8_t addr, size_t delayMS = 20);
```

3.2 C 接口

3.2.1 枚举类型

```
/*
 * ModelType是一个枚举类型，代表工作模式类型，其中，
 * @PULSE_C、@PULSE_P为单场景模式；
 * @TTL_S、@TTL_C为多场景模式。
 */
enum ModelType
{
    PULSE_C = 0,
    PULSE_P = 1,
    TTL_S = 2,
    TTL_C = 3
};

/*
 * TTL是一个枚举类型，代表触发电平，其中，
 * @TTL_L为低电平触发，@TTL_H为高电平触发。
 */
enum TTL
{
    TTL_L = 0,
    TTL_H = 1
};
```

3.2.2 结构体

```
/*
 * DevInfo是一个结构体，用于保存光源设备信息，其中：
 * @UID为光源设备的唯一标识符，长度为6个字节；
 * @address为光源设备的当前地址，范围为1~32；
 * @productModel为光源设备的型号；
 * @zoneSize为光源设备支持的分区数；
 * @colorSize为光源设备的颜色通道数。
 */
struct DevInfo
{
    int UID[6] = {0};
    int address;
    char productModel[128] = {0};
    int zoneSize;
    int colorSize;
};
```

3.2.3 方法

3.2.3.1 初始化

```
/*
 * init()方法主要用于连接端口，获取光源设备信息等；
 * @comNo为待连接的端口号，如"COM3"的3；
 * @devInfoNum为搜索到的设备数量；
 * @return返回一个DevInfo数组，数组大小为@devInfoNum，每个元素为一个DevInfo结构体实例；
 * 连接成功后，将花费数秒至数十秒搜索和读取光源设备的信息，耗时长短与光源设备数目有关。
 * 不再使用DevInfo数组时，请自行释放资源。
 */
DevInfo* init(int comNo, int* devInfoNum);
```

3.2.3.2 修改地址

```
/*
 * changeAddr()方法用于修改光源设备的地址；
 * @UID为光源设备的唯一标识符，可以参考从init()获取的DevInfo数组，其中包含了搜索到的光源设备UID；
 * @addr为光源设备的新地址，范围为1~32；
 * 修改地址成功后，请自行更新DevInfo中光源的地址信息；
 * 请避免将不同类型光源置于同一地址，否则可能后续的setZoneAndColors()等对不同类型光源无效。
 */
bool changeAddr(int UID[6], int addr);
```

3.2.3.3 调光

```
/*
 * setZoneAndColors()方法用于显示当前的调光效果；
 * @zoneMode为分区方式；
 * @currentZone为当前的分区号，范围为1~DevInfo::zoneSize；
 * @brightness为当前的亮度值，范围为0~1024；
 * @colors为当前的颜色值，通道数可由DevInfo::colorSize获取，每个元素为0~1024的整数值；
 * @addr为该方法作用的光源设备地址，范围为1~32；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐值10。
 * 请依据具体的光源型号合理设置各参数。
 */
bool setZoneAndColors(int zoneMode, int currentZone, int brightness, int colors[], int
addr, int delayMS);
```

3.2.3.4 保存场景

```
/*
 * saveScene()方法用于保存场景；
 * @modelType代表工作模式类型，可输入参数为ModelType::PULSE_C、ModelType::PULSE_P、
ModelType::TTL_S或ModelType::TTL_C；
 * @ttl代表触发电平类型，可输入参数为TTL::TTL_L或TTL::TTL_H；
 * @addr为地址，范围1~32；
 * @sceneNo为该场景的序号，以1为起始，范围为1~@sceneSize，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C
和Pu-P工作模式下置1即可；
 * @sceneSize为场景的总数，最多支持8个场景，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C和Pu-P工作模式下
置1即可；
 * @onDur为亮灯持续时间，单位为微秒，范围为1~65535，在Pu-P，TTL_S和TTL_C工作模式下该参数有效，在Pu-C工
作模式下置1即可；
 * @offDur为灭灯持续时间，单位为微秒，范围为0~65535，在TTL_S和TTL_C工作模式下该参数有效，在Pu-C和Pu-P
工作模式下置0即可；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐值100。
 */
bool saveScene(ModelType modelType, TTL ttl, int addr, int sceneNo, int sceneSize, int
onDur, int offDur, int delayMS);
```

3.2.3.5 进入工作模式

```
/*
 * enterWorkModel()方法用于进入工作场景；
 * @modelType代表工作模式类型，可输入参数为ModelType::PULSE_C、ModelType::PULSE_P、
ModelType::TTL_S或ModelType::TTL_C；
 * @addr为地址，范围1~32；
 * @delayMS为该方法发送报文后的间隔时间，单位为毫秒，可自行设置，推荐值20。
 * 该方法通常在成功调用saveScene()之后调用；
 * 当处于TTL_C工作模式下再次调用enterWorkModel(ModelType::TTL_C, addr)时，将重置为从场景1开始执行，
具体请见文档。
 */
bool enterWorkModel(ModelType modelType, int addr, int delayMS);
```

3.2.3.6 释放内部资源

```
/*
 * destroy()方法用于释放内部资源。
 */
void destroy();
```

第四章 SDK 调用方法

4.1 SW_SDK 的总体调用流程

总体调用流程见图 4-1，这里以 C++接口为例。

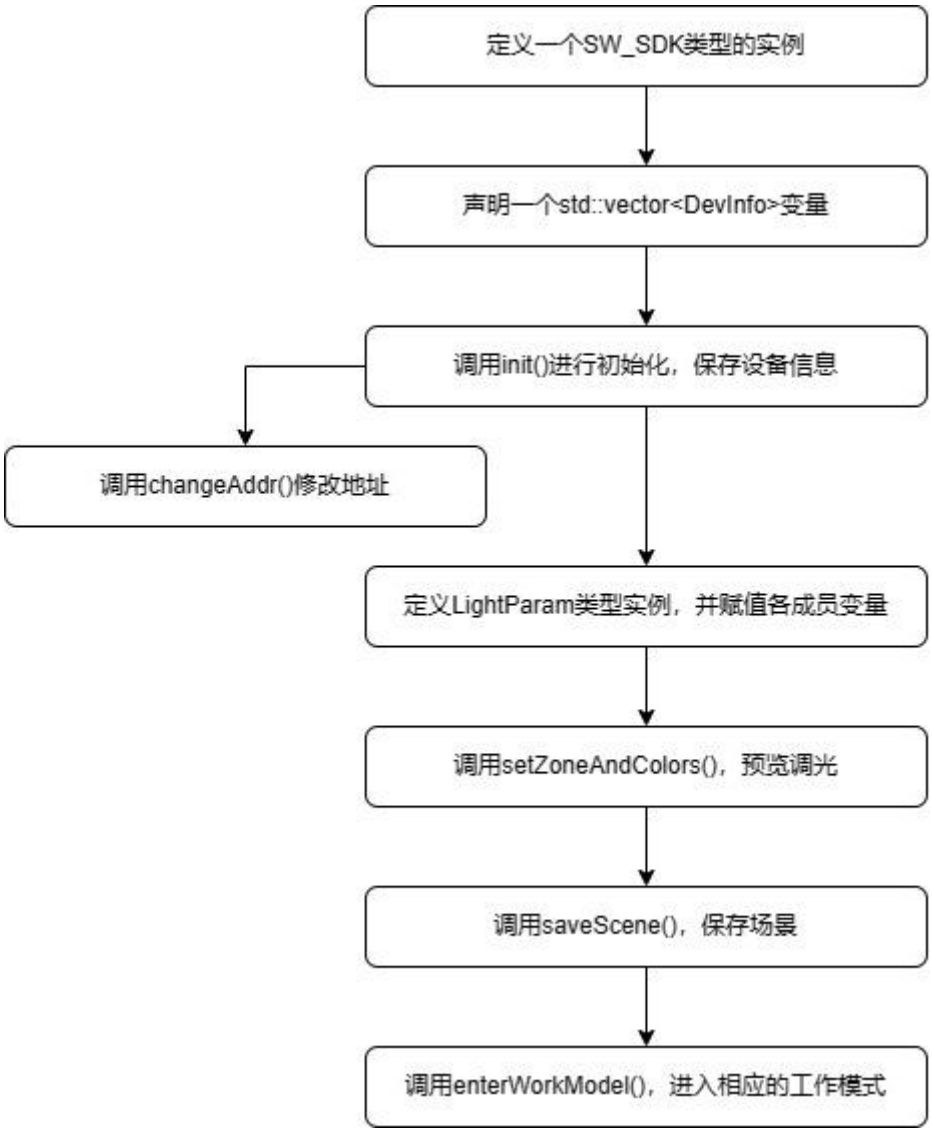


图 4-1 SW_SDK 的调用总体流程

4.2 SW_SDK 的调用示例

调用示例以 C++接口为例，关于 C#、Python 示例请参见 samples 文件夹。

1)定义一个 SW_SDK 实例，该类型遵循 RAII，在离开作用域时会自动释放资源。

```
SW_SDK demo;
```

2)声明一个 std::vector<DevInfo>变量。

```
std::vector<DevInfo> devInfos;
```

3)调用 init()，若返回成功，表明连接端口成功；请根据 devInfos.empty()或 devInfos.size()等判断获取的设备数目。

```
auto res = demo.init("COM3", devInfos);
```

4)可以选择调用 changeAddr()修改地址，UID 可通过 init()调用成功获取的 std::vector<DevInfo>中获得。若从未修改过地址，所有光源设备的初始默认地址为 1。

```
res = demo.changeAddr(devInfos.front().UID, addr1);
```

5)定义一个 LightParam 实例，并赋值各成员变量。

```
LightParam lpFor4z4c = {
    0,
    1,
    100,
    {100, 0, 0, 0, 0}};
```

或

```
lpFor4z4c.zoneMode = 1;
lpFor4z4c.currentZone = 3;
lpFor4z4c.brightness = 500;
lpFor4z4c.colors = {0, 200, 0, 200};
```

6)调用 setZoneAndColors()以展示调光效果。

```
demo.setZoneAndColors(lpFor4Z4C, addr1);
```

7)调用 saveScene()保存场景。

```
demo.saveScene(ModelType::TTL_S, TTL::TTL_L, addr1, 1, 8, 65535, 65535);
```

8)可选择重复 4)5)6)7)步骤，以达到将光源设备按地址分组、保存多个场景等的需求。

```
res = demo.changeAddr(UID1, addr1);

res = demo.changeAddr(UID2, addr2);

res = demo.changeAddr(UID3, addr3);
```

```
lpFor4z4c.zoneMode = 0;
lpFor4z4c.currentZone = 1;
lpFor4z4c.brightness = 100;
lpFor4z4c.colors = {100, 0, 0, 0};
demo.setZoneAndColors(lpFor4z4c, addr1);
demo.saveScene(ModelType::TTL_S, TTL::TTL_L, addr1, 2, 8, 65535, 65535);
```

```
lpFor4z4c.zoneMode = 1;
lpFor4z4c.currentZone = 4;
lpFor4z4c.brightness = 1200;
lpFor4z4c.colors = {100, 100, 100, 100};
demo.setZoneAndColors(lpFor4z4c, addr1);
demo.saveScene(ModelType::TTL_S, TTL::TTL_L, addr1, 3, 8, 65535, 65535);
```

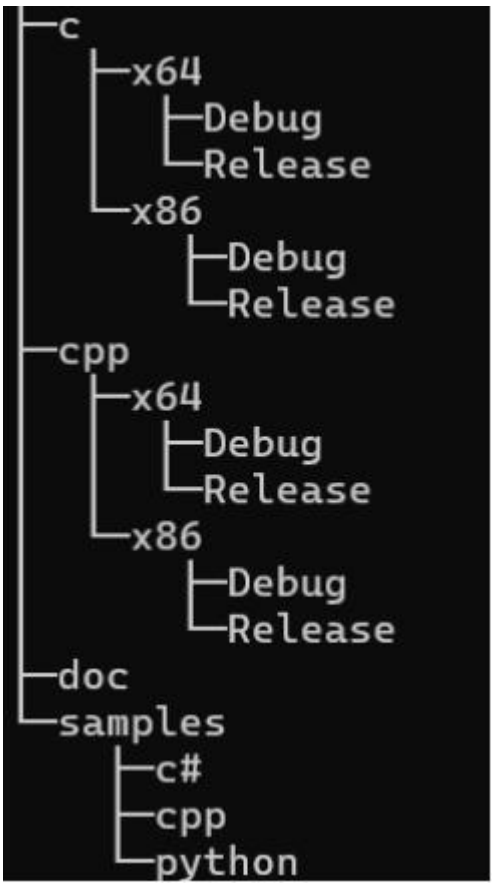
9)调用 enterWorkModel()进入工作模式。

```
demo.enterWorkModel(ModelType::TTL_S, addr1);
```

完整的示例代码请参阅 SW_SDK\examples 文件夹。

4.3 SW_SDK 的使用

SW_SDK 文件目录结构如下：



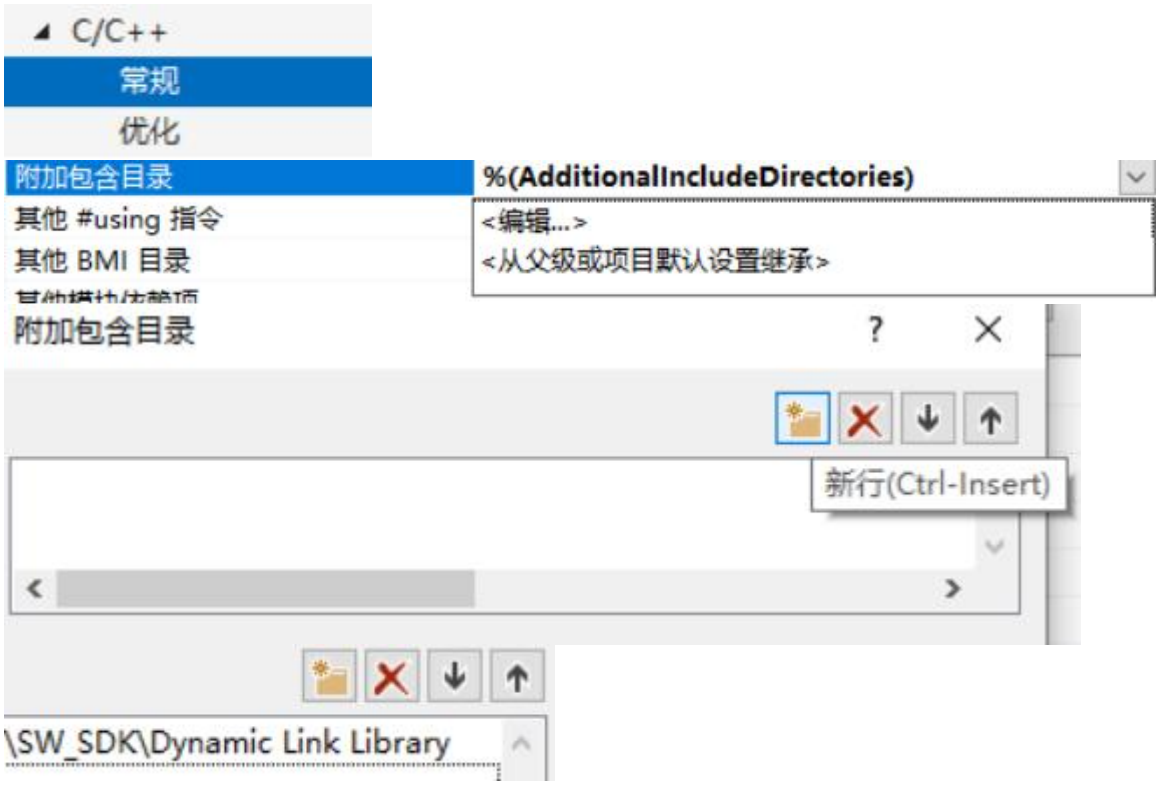
其中，c 和 cpp 文件夹中分别包含对应的头文件和动态库，samples 中包含了示例程序。

4.3.1 C++项目使用动态库

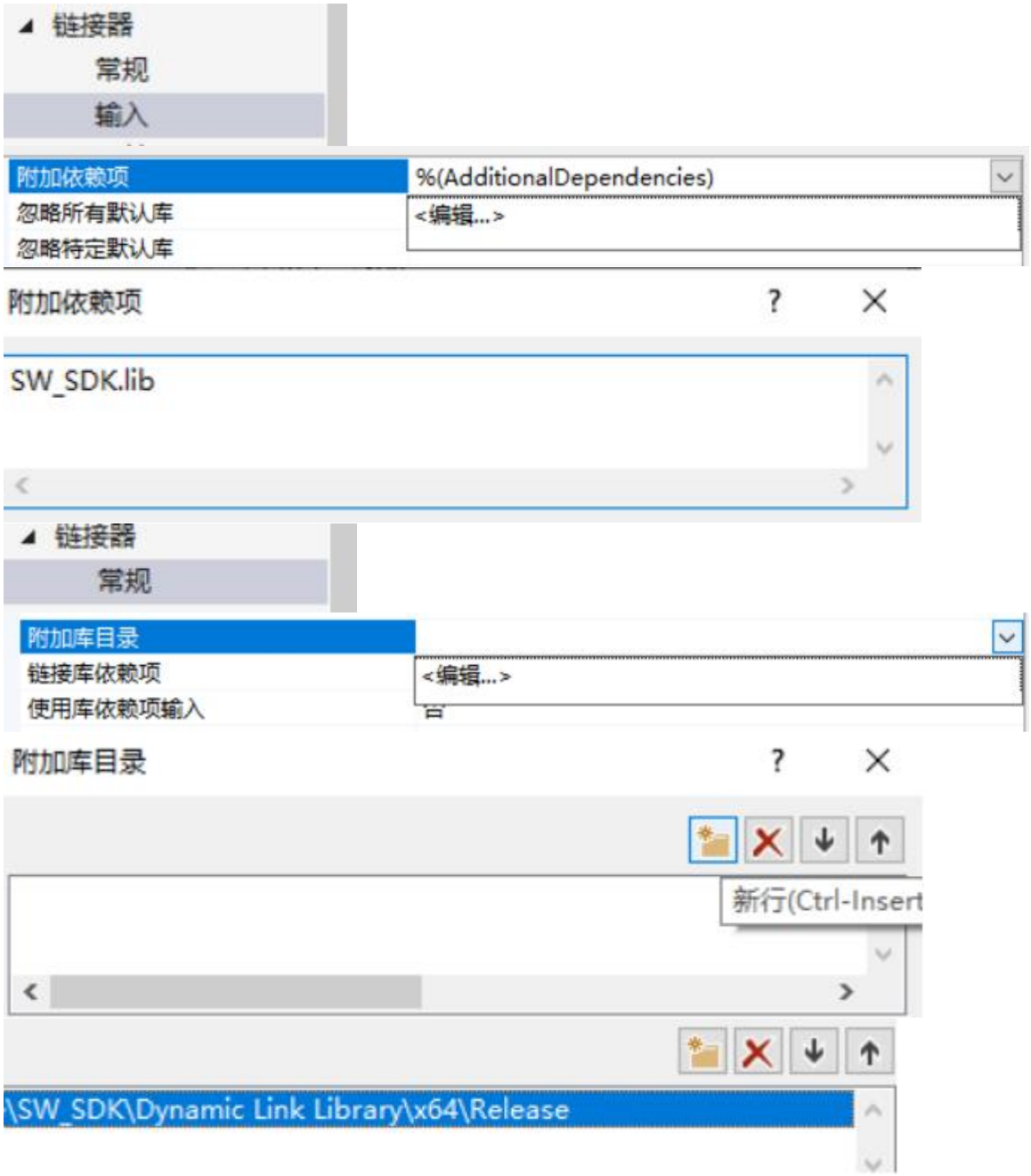
以下示例为 C++项目，VS2022 集成开发环境，<配置>为<Release>，<平台>为<x64>。假设 SW_SDK 的绝对路径为”Your_Work_Space\SW_SDK”。

示例代码中使用的是相对路径：#include “SW_SDK.h”，需要在项目属性中设置包含目录。点击<项目>选择<属性>，在属性页的配置选择<Release>，平台选择<x64>。

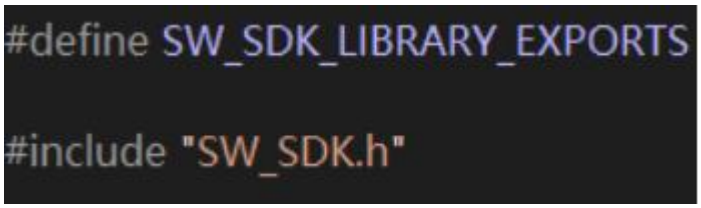
选择<C/C++>的<常规>属性页，在<附加包含目录>属性中，指定”SW_SDK.h”的路径。打开<附加包含目录>下拉列表，点击<编辑>，点击<新行>，然后选择行末尾的省略号(...)。在<选择目录>对话框中，选择包含”SW_SDK.h”的目录，在本例中为”Your_Work_Space\SW_SDK\Dynamic Link Library”，点击<确认>。



选择<链接器>的<输入>属性页，打开<附加依赖项>下拉列表，点击<编辑>，添加”SW_SDK.lib”，点击<确认>。选择<链接器>的<常规>属性页，打开<附加库目录>下拉列表，点击<编辑>，点击<新行>，然后选择行末尾的省略号(...)。在<选择目录>对话框中，选择包含”SW_SDK.lib”的目录，本例中为”Your_Work_Space\SW_SDK\Dynamic Link Library\x64\Release”，点击<确认>。点击项目属性页的<确定>或<应用>以保存对项目所做的更改。



在示例代码头文件的起始位置定义宏，SW_SDK_LIBRARY_EXPORTS



或者在项目属性页中选择<C/C++>的<预处理器>，打开<预处理定义>下拉列表，点击<编辑>，添加”SW_SDK_LIBRARY_EXPORTS”，点击<确定>。点击项目属性页的<确定>或<应用>以保存对项目所做的更改。

最后，将对应项目配置的 SW_SDK.dll 复制到包含可执行文件的目录中。本例中的对应的 SW_SDK.dll 为”Your_Work_Space\SW_SDK\cpp\x64\Release\SW_SDK.dll”。

4.3.2 C#、Python 项目动态库的使用

请参阅 samples\c#和 samples\python 的示例源码，编译之后将 SW_SDK.dll 复制到包含可执行文件的目录中，对于程序打包请使用相应的 Release 版本 dll。