OpenVINO 开发配置与必备基础知识

原创 贾志刚 英特尔物联网 *1 周前*

通过第一篇文章我们已经了解什么是 OpenVINO，它的诸多功能与全应用场景支持人工智能落地的能力。本篇我们将重点介绍 OpenVINO 开发流程与开发必备的基础知识与相关 API 函数对象。

环境配置

在具体介绍 OpenVINO 开发流程与开发必备基础知识之前，我们首先需要配置好 OpenVINO 的开发环境，这里我们以 Win10 系统下 OpenVINO C++/Python SDK 开发与应用集成为例来完成整个教程的配置与代码演示。基于 VS2017+OpenVINO2021.02 版本的环境配置可以总结为如下几个步骤：

1. 打开 VS2017，新建一个控制台应用，图示如下：

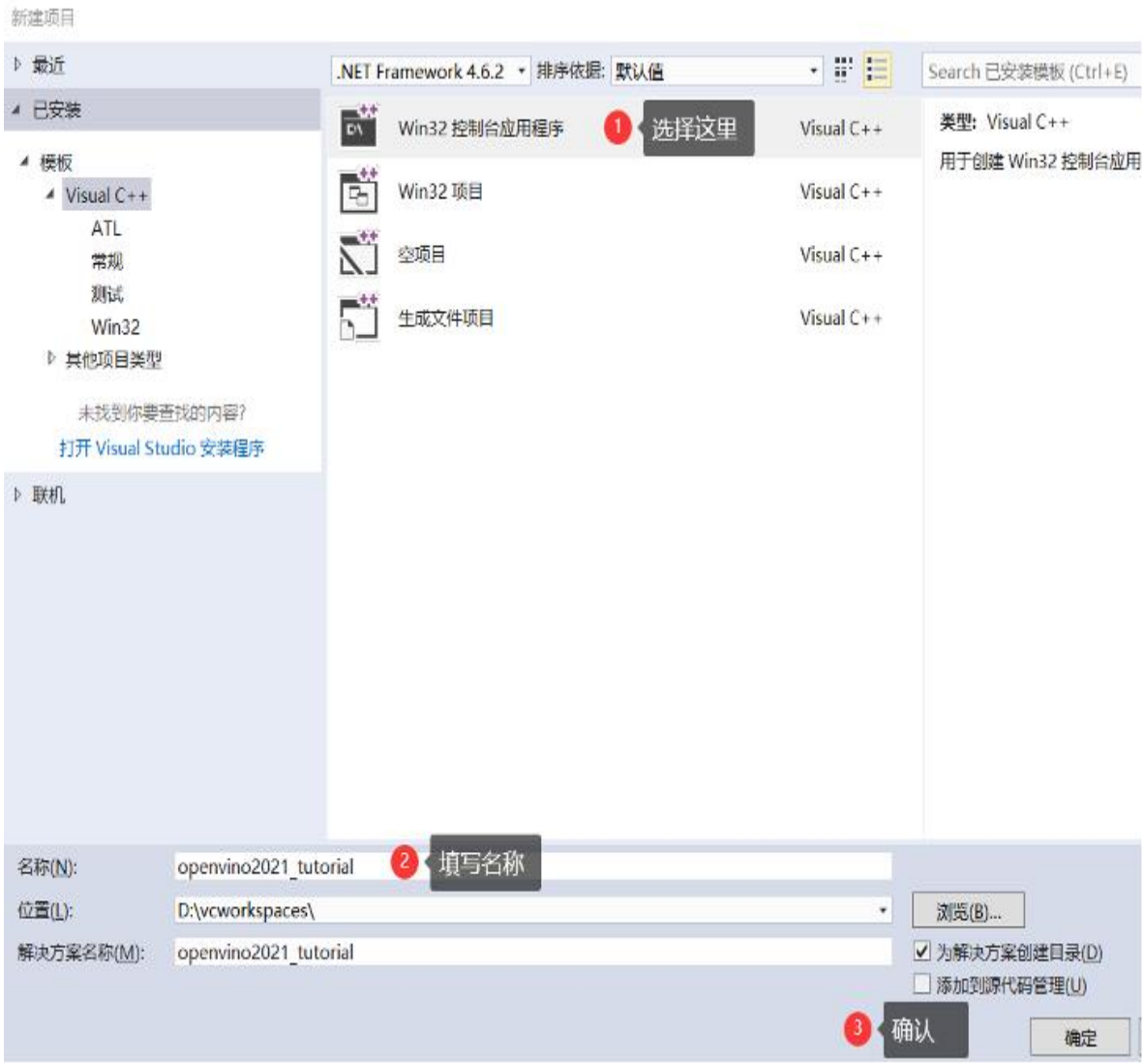

图 1

2. 打开属性管理器

打开属性管理器，选择 x64/release 然后配置包含路径，库路径、通过链接器添加 lib 文件，这部分的配置图示如下：

包含目录配置

```
C:\Program Files %28x86%29\Intel\openvino_2021.2.185\opencv\include\opencv2
C:\Program Files %28x86%29\Intel\openvino_2021.2.185\opencv\include
C:\Program Files %28x86%29\Intel\openvino_2021.2.185\deployment_tools\inference_engine\include
```

库路径配置

```
C:\Program Files %28x86%29\Intel\openvino_2021.2.185\opencv\lib
C:\Program Files %28x86%29\Intel\openvino_2021.2.185\deployment_tools\inference_engine\lib\intel64\Release
```

链接器：

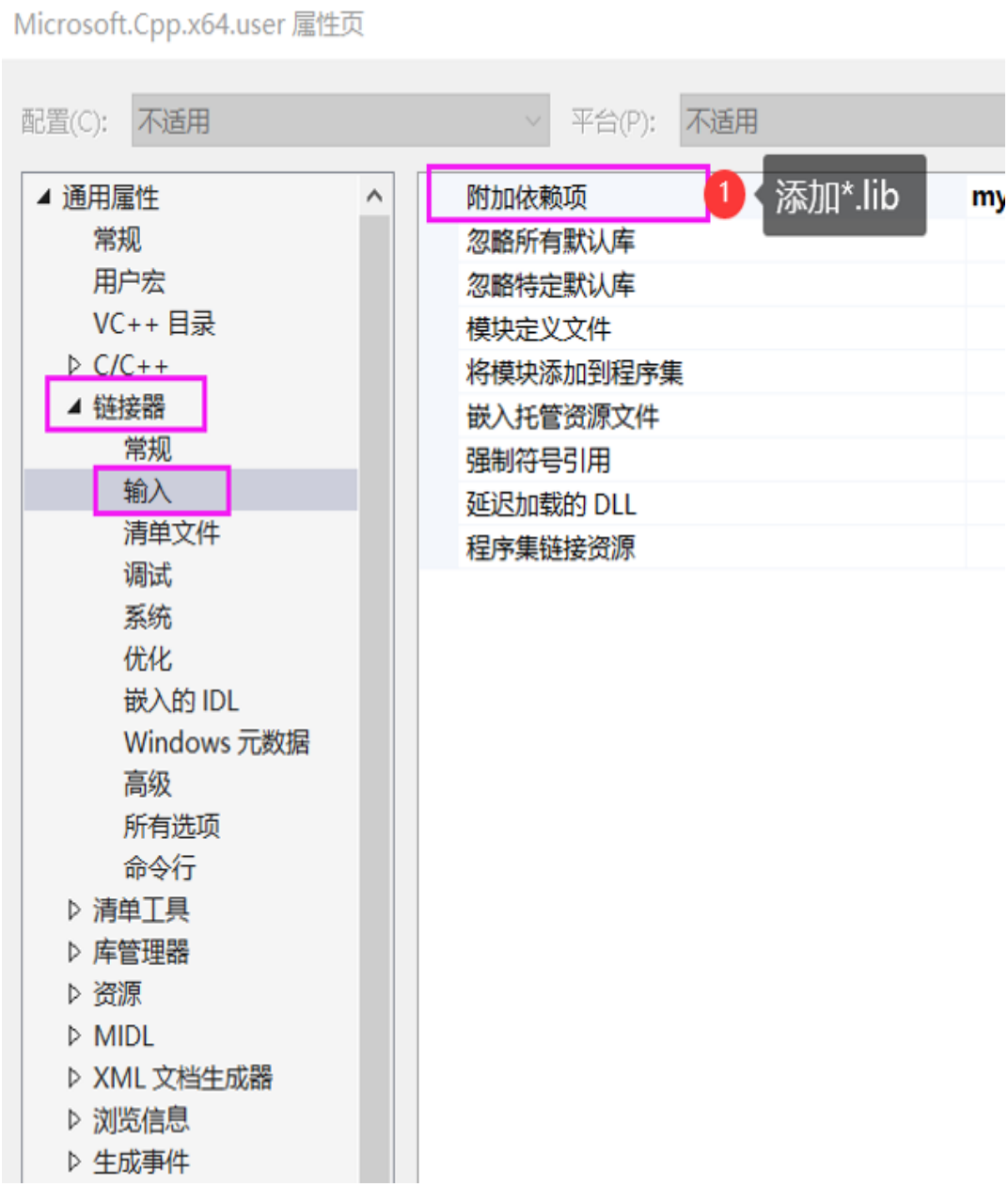

图 2

opencv_calib3d451.lib

opencv_core451.lib

opencv_dnn451.lib

opencv_features2d451.lib

opencv_flann451.lib

opencv_gapi451.lib

opencv_highgui451.lib

opencv_imgcodecs451.lib

opencv_imgproc451.lib

opencv_ml451.lib

opencv_objdetect451.lib

opencv_photo451.lib

opencv_stitching451.lib

opencv_video451.lib

opencv_videoio451.lib

inference_engine.lib

inference_engine_c_api.lib

inference_engine_ir_reader.lib

inference_engine_legacy.lib

inference_engine_lp_transformations.lib

inference_engine_onnx_reader.lib

inference_engine_preproc.lib

inference_engine_transformations.lib

最后配置环境变量，添加以下环境变量到系统的 path 中去，图示如下：

对于开发环境配置环节，如果还有不清楚的，可以参考 OpenVINO 中文社区的技术自愿者分享的视频，地址如下：

https://www.bilibili.com/video/BV1Hz4y1U7g6

设备查询与开发基础知识

完成上述配置以后，重启 VS2017，创建一个新的 cpp 文件，添加下面的代码到 cpp 文件中

```
#include <inference_engine.hpp>   1 导入头文件支持
#include <opencv2/opencv.hpp>

using namespace InferenceEngine;  2 推理引擎命名空间支持

int main(int argc, char** argv) {
    InferenceEngine::Core ie;  3 初始化IE
    std::vector<std::string> devices = ie.GetAvailableDevices();  4 查询支持设备
    for (std::string name : devices) {
        std::cout << "device name: " << name << std::endl;  5 输出到控制台
    }
    std::string cpuName = ie.GetMetric("CPU", METRIC_KEY(FULL_DEVICE_NAME)).as<std::string>();
    std::cout << "cpu full name: " << cpuName << std::endl;
    6 图像加载与显示支持
    cv::Mat src = cv::imread("D:/images/lena.jpg");
    cv::imshow("输入图像", src);
    7 等待键盘响应，退出与销毁窗口
    cv::waitKey(0);
    cv::destroyAllWindows();
    return 0;
}
```

运行结果如下：

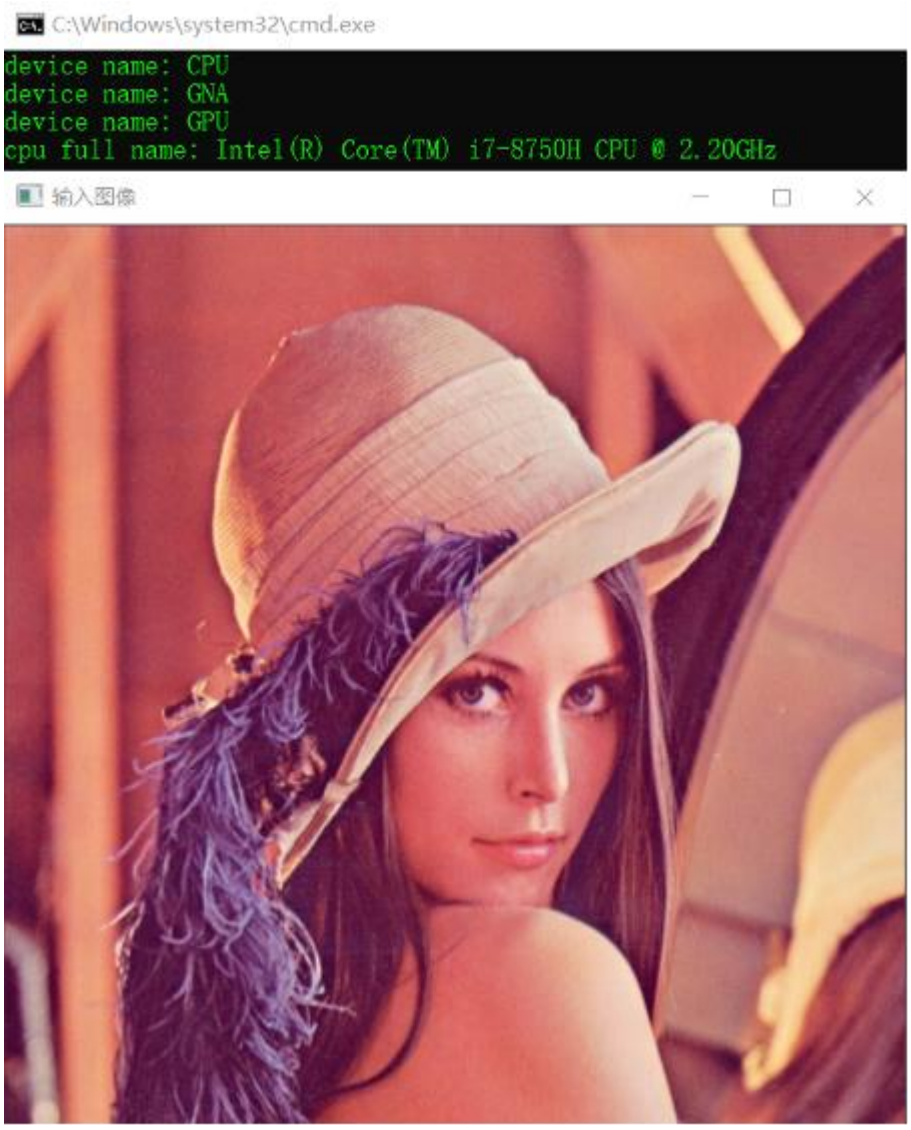

图 3

上述控制台输出来自 InferenceEngine::Core 的设备查询函数 GetAvailableDevices，它可以查询当前系统支持 IE 推理的硬件支持，该函数如下：

std::vectorInferenceEngine::Core::GetAvailableDevices() const

参数: 无

返回的支持设备的列表 vector

下面的就是加载图像与显示图像，使用的两个函数来自 OpenVINO 中的 OpenCV 组件支持，两个相关函数如下：

读取图像

```
Mat cv::imread(
const String &       filename,
int flags = IMREAD_COLOR
)
```

参数 filename 表示文件路径(包含文件名)

第二个参数为默认参数

加载成功返回的图像像素的矩阵数据结构 Mat，默认读取加载为彩色图像，三个通道顺序为 BGR。

显示图像

```
void cv::imshow(
     const String & winname,
     InputArray   mat
)
```

参数 winname 表示窗口名称，本例中为“输入窗口”

参数 mat 表示图像矩阵 Mat(显示图像的内存表示)

最终执行结果图上图 3 所示。对上述代码，我们可以通过进一笔的简化，要知道在 C++11 中，声明类型可以自动识别，通过 auto 来表示可以避免代码过长，同时支持 for 循环的时候通过 auto 自动识别每个 item 的类型,所以上述查询设备与打印部分的代码：

```
nferenceEngine::Core ie;
std::vectordevices = ie.GetAvailableDevices();
for (std::string name : devices) {
     std::cout << "device name: " << name << std::endl;
}
```

改写为如下的代码：

```
InferenceEngine::Core ie;
auto devices = ie.GetAvailableDevices();
for (auto name : devices) {
     std::cout << "device name: " << name << std::endl;
}
```

这样看上去代码就会比之前的整洁更加直观一点。在 OpenVINO SDK C++的开发中，有很多类别的声明都很长，我们可以通过使用 C++11 支持的自动类型识别关键字 auto 来减少不必要的代码书写，提高编码效率。此外类 InferenceEngine::Core 类是表示整个 IE 引擎的实例，支持从模型加载、输入与输出格式获取与设置、模型的推理与后处理等一系列的操作。关于如何使用 InferenceEngine::Core 实现模型推理的流程与相关 API 方法函数解释，我们将在一篇文章中详细介绍。