Leap Motion 开发白皮书

（李文敏 浙江大学心理与行为科学东西 杭州 310028）

1. 前言

现在的时代，几乎是 GUI 的最好时代，难以有更大的飞跃式突破，未来必然产生一种新的交互方式，我们称之为自然交互，而体感交互是自然交互的一种，体感交互基于人的手势动作等来间接操控计算机，达到更为自然的交互效果。目前市场上关于体感交互已经有很多设备，其中 Leap Motion 以其高精度的手势信息获取在体感交互占据一席之地。

2. 开发平台搭建

2.1 程序接口

Leap Motion 体感控制器基本支持当前流行的桌面操作系统（Windows、Mac 与 Linux），Leap Motion 程序作为一个服务（在 Windows 中）或一个守护进程（在 Mac 和 Linux 中），通过 USB 总线和 Leap Motion 体感控制器相连，基于 Leap Motion 的应用程序通过 Leap Motion Service 来获取运动追踪数据。

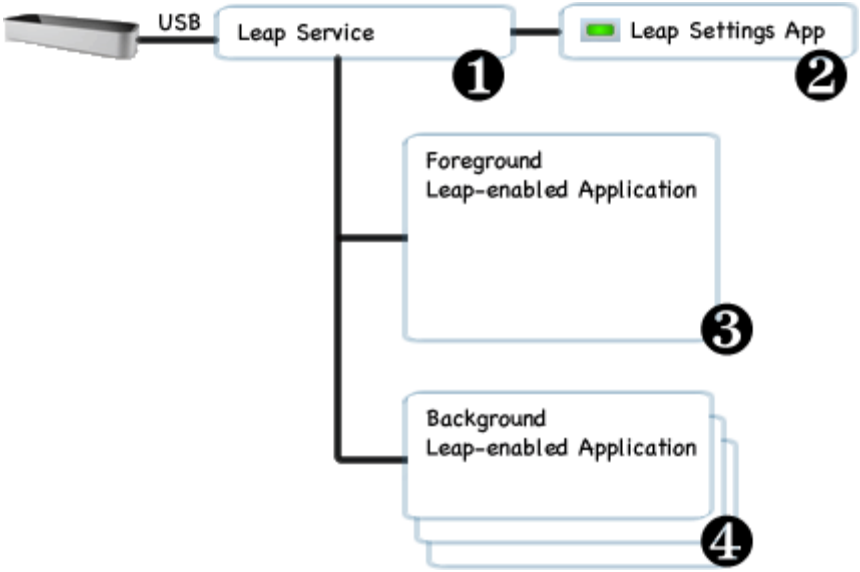
Leap Motion 的 SDK 提供两类 API 来获取 Leap Motion 的数据，这些 API 可以允许我们在多个语言环境下开发基于 Leap Motion 控制器的应用程序，包括在浏览器中运行 JavaScript 语言。

2.1.1 程序编程接口

Leap Motion SDK 提供两类 API 从 Leap Motion 服务中来获取追踪数据，一个是原生接口，一个是网页套接字 Web Socket。原生接口允许我们创建新的 Leap 应用程序的动态库。而网页套接字接口 Web Socket 和 JavaScript 客户端库则允许我们可以创建基于 Leap 的应用。

2.1.2 Leap Motion 的应用接口

应用程序接口由动态链接库提供，该库连接到 Leap Motion 服务，并且为我们的程序提供追踪数据，如果我们通过 C++和 Objective-C 开发应用，可以直接链接到动态库，如果我们通过 Java、C#、Python 开发应用，可以通过绑定链接。



图示：基于 Leap 的应用程序

1） Leap Motion 的服务通过 USB 总线，从 Leap Motion 控制器接收数据并发送到正在运行的 Leap 应用，默认情况下服务仅向前台程序发送追踪数据，但是应用程序也可以通过询问等方式，使得他们从后台也可以接收数据。

2） Leap Motion 的对话框设置和服务是独立的，它允许计算机用户配置 Leap Motion 的安装，在 Windows 下，这个配置程序是个控制面板小程序，在 Mac 操作系统下是一个菜单条。

3） 前台 Leap Motion 应用程序从服务接收追踪数据，Leap 应用程序可以通过原生 Leap Motion 库链接到 Leap Motion 服务，我们的应用程序既可以通过 C++和 Objective-C 直接连到原生库上，也可以通过语言包装库（Java，C#和 Python）链接。

4） 当 Leap 应用程序失去了操作系统焦点，Leap Motion 服务则停止向应用程序发送数据，经过请求许可，应用程序可以在后台接收数据，当运行在后台时，由前台程序配置。

2.2 语言支持

Leap Motion 支持几乎所有的主流语言，包括 C++、C#、Python、JavaScript、Java、Objective-C 等编程语言，还包括其他开发工具如 Unity、Unreal。

Leap Motion 库由 C++编写，同时用了开源工具 SWIG 来生成 C#、Java 和 Python 的语言绑定（SWIG 生成的绑定开源翻译绑定语言对于 C++ Leap Motion 库的调用，每个 SWIG 绑定都使用两个附加库）；对于 JavaScript 和网页程序开发，Leap Motion 则提供了网页套接字服务和客户端 JavaScript 库。这些库文件均包含在官方 SDK 中。

Leap Motion 本身并不是非常复杂，最开始的难点在于语言运用，大多数的心理同学大一学的是 C（面向过程编程），而这里其实是不支持的。因此语言学习是大家需要跨过的第一关，尤其需要注意类相关的编程思想。

2.3 环境配置

2.3.1 开发工具

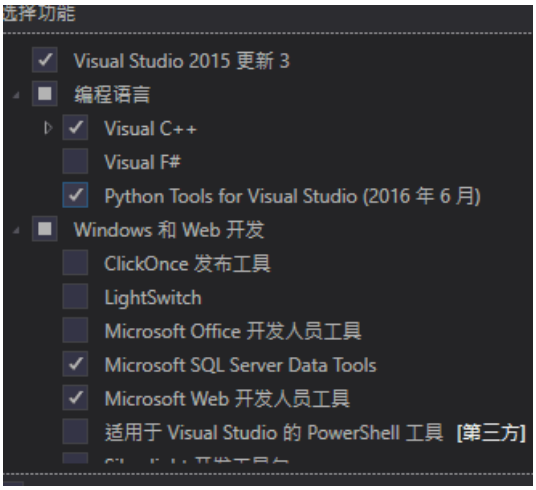
1）在硬件方面，我们需要：一个 Leap Motion 控制器、一根自带的 USB 数据线（附带还有一根数据线，并没有任何用处）、计算机。

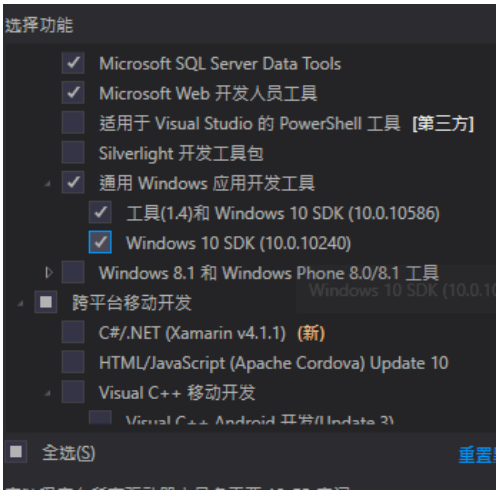
2）在软件方面，建议大家安装最新的 Visual Studio 2015 （Enterprise 版），虽然网站上显示要收费，实际上并不收费，激活并不难；相比较于其他编程软件，Visual Studio 支持大多数主流语言，包括 unity 和 unreal，可以满足不同开发者的需求，而相较于 Community 和 Professional 版，Enterprise 软件更加完善和强大。下面是安装指南：

Step1：（笔者为 Windows 10 专业版）。下载软件（官方下载链接 或 笔者分享的百度云盘链接 提取码为 jsmj）。

Step2：点击安装，设置必要项（建议安装在其他盘，软件大小较大，安装时间较长，4 个小时以上…，笔者的网速不给力硬是安装了一天…）

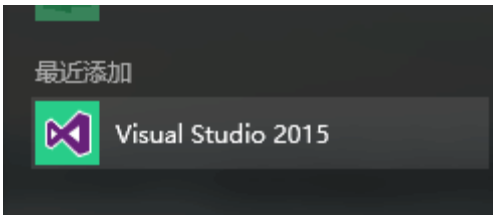
除了默认选项，编程语言里“Visual C++”、Windows 和 Web 开发里“通用 Windows 应用开发工具”，其他选项大家根据自己的兴趣安装（不是专业码农，没有必要全部安装，该软件可以后期修改安装，如果有必要再补安装）。



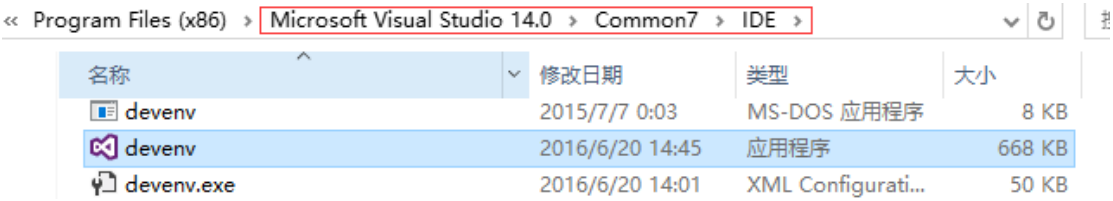


安装成功需要重启。

Step3：桌面并没有快捷方式，可以在开始菜单找到 Visual Studio 2015 快捷方式，发送到桌面即可。



有时候，开始菜单也找不到快捷方式，可以在安装位置的 IDE 文件夹下找到 devenv.exe 文件，发送快捷方式到桌面即可。



Step4：打开软件选择主题颜色和常规开发环境（我选择 Visual C++）即可启动。右上角登录，输入序列号激活，序列号大家可以网上搜，我这里也提供一个：HM6NR-QXX7C-DFW2Y-8B82K-WTYJV，亲身试验有效。

（如果你安装的是英文版，一开始可能不太习惯，可以切换语言，当然也可以一开始就选择中文版进行安装）

在工具—选项—环境—区域设置中选择语言设置，如果没有先获取其他语言安装包，下载安装，重启。

3）在开发方面，我们需要官方提供的相关库文件，即下载对应的 SDK。

Step1：下载 Leap Motion 官方 SDK（里面有两个版本，一个是虚拟现实，一个是桌面程序版本，下载桌面程序版本）保存到某个位置（保存到某个方便的位置）并安装。

Step2：回到 SDK 文件夹，SDK 点击安装后会给电脑安装 APP Home 应用商店，同时也安装 Leap Motion 服务。另一个文件夹里面才是重点，里面包含了我们开发所必须的相关文件和简易的 sample 代码。

2.3.2 编程环境搭建

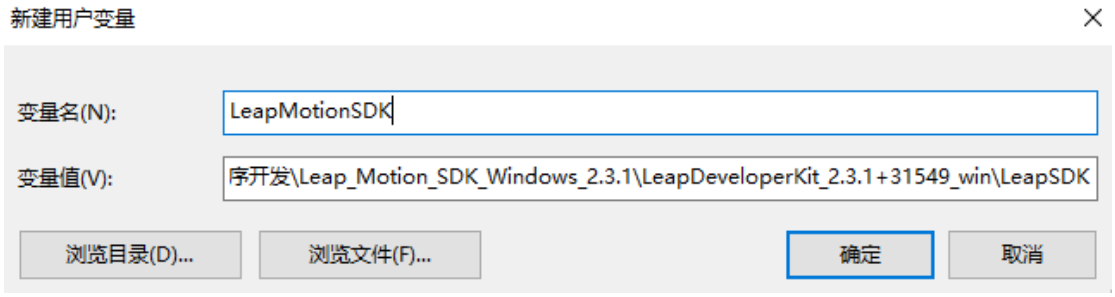
已经讲到很多编程语言都支持 Leap Motion 的程序开发，这里向大家介绍 C++开发环境下的环境配置。

在 C++中有许多工程项目，或 win32 项目或空白项目或 WFC，但是环境配置却基本相同，按如下步骤操作即可。

Step1:添加环境变量

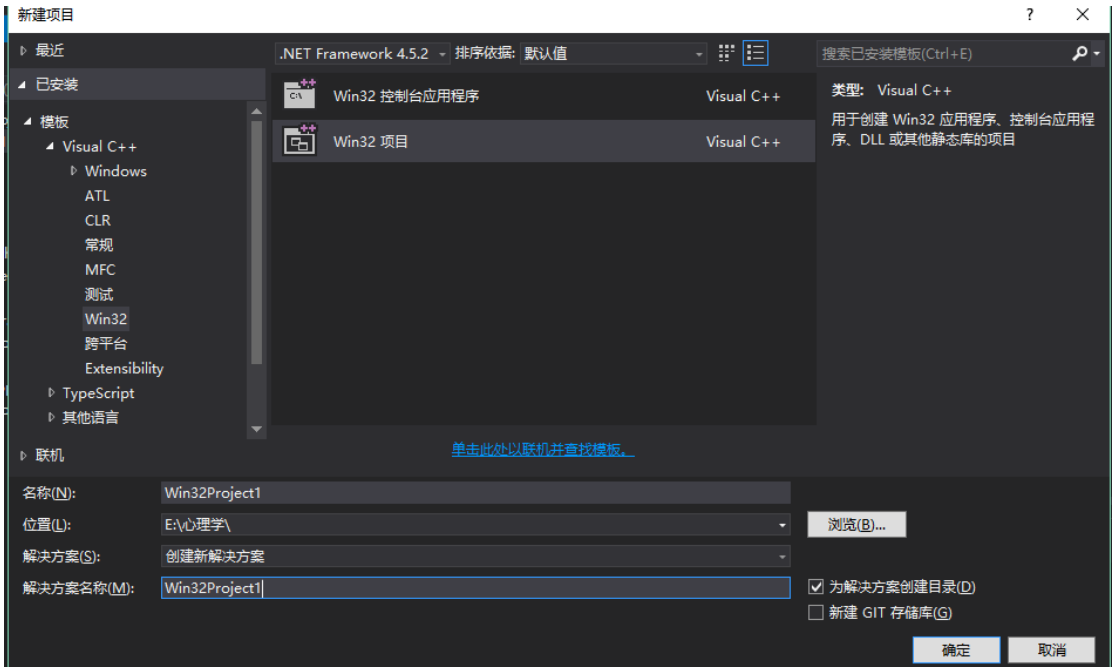
将下载好的 SDK 中 LeapSDK 文件夹所处计算机位置复制

右键计算机属性—高级系统设置—环境变量，添加一个新的变量名为 LeapMotionSDK、变量值为复制的位置的环境变量（也可以是其他名称，这里以 LeapMotionSDK 为例，需要记住该名称）



重启计算机使环境变量生效。

Step2:创建 win32 项目或 WFC 或空项目均可，以 win32 为例

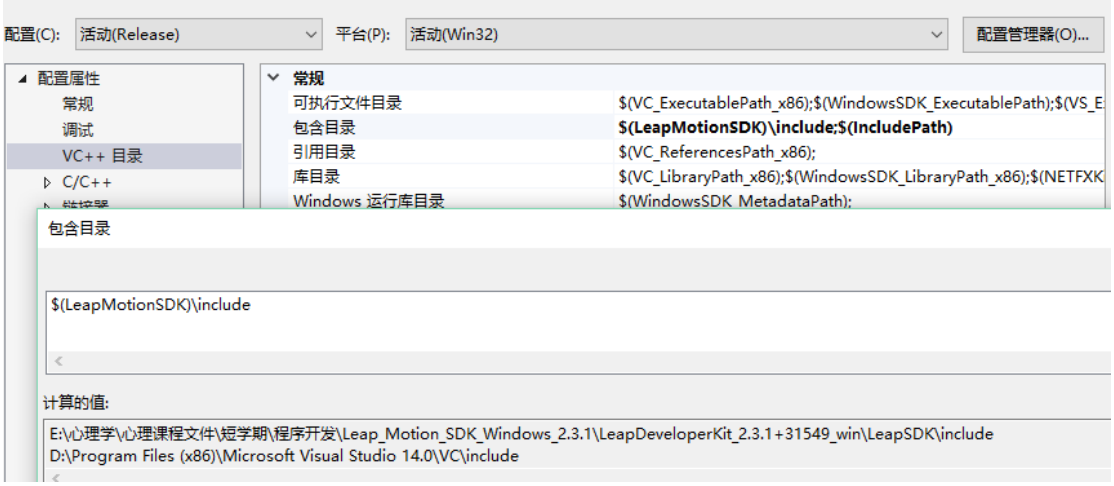


Step3:编程环境配置三部曲-项目属性设置

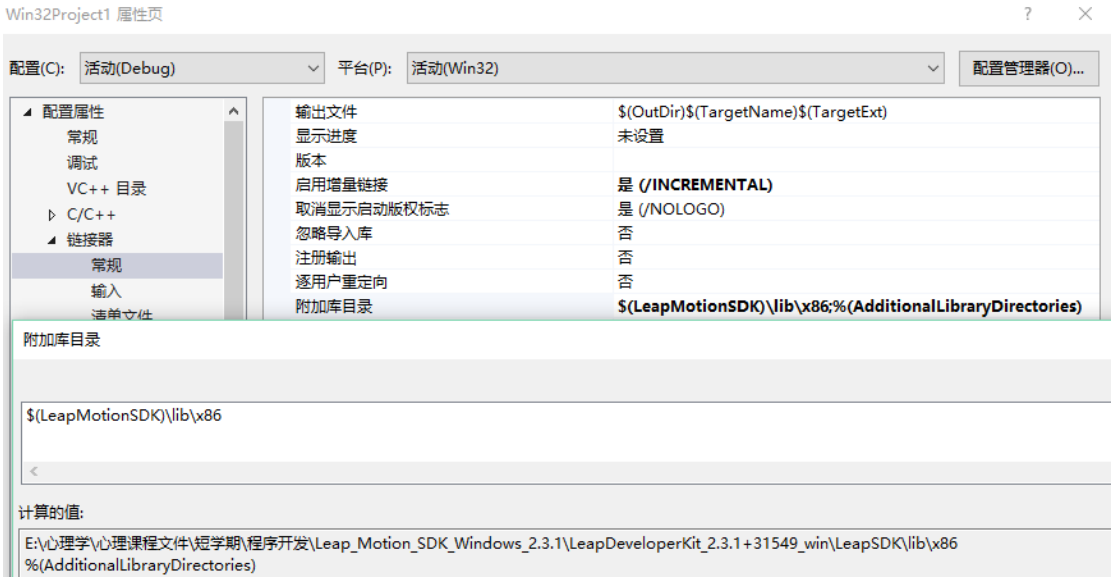
当创建环境变量并生效后，以后每次开发程序都只需要进行以下四步即可。

以下设置里出现的 LeapMotionSDK 与上文笔者设置环境变量名称一致，更换计算机后需要更改你自己计算机上的相关名称，否则笔者提供的代码无法正常编译）

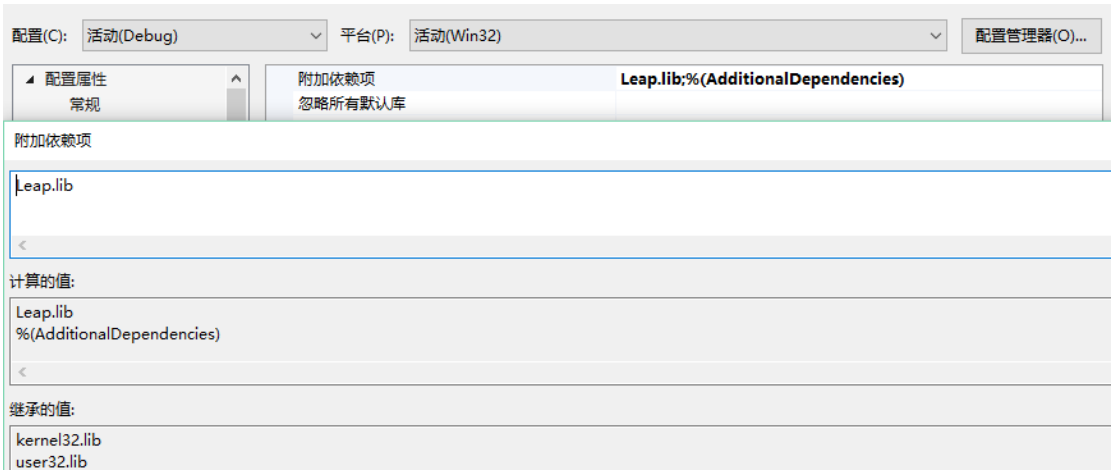
关键设置 1：VC++目录下设置包含目录为 $(LeapMotionSDK)\include



关键设置 2：链接器常规设置下—附加库目录$(LeapMotionSDK)\lib\x86



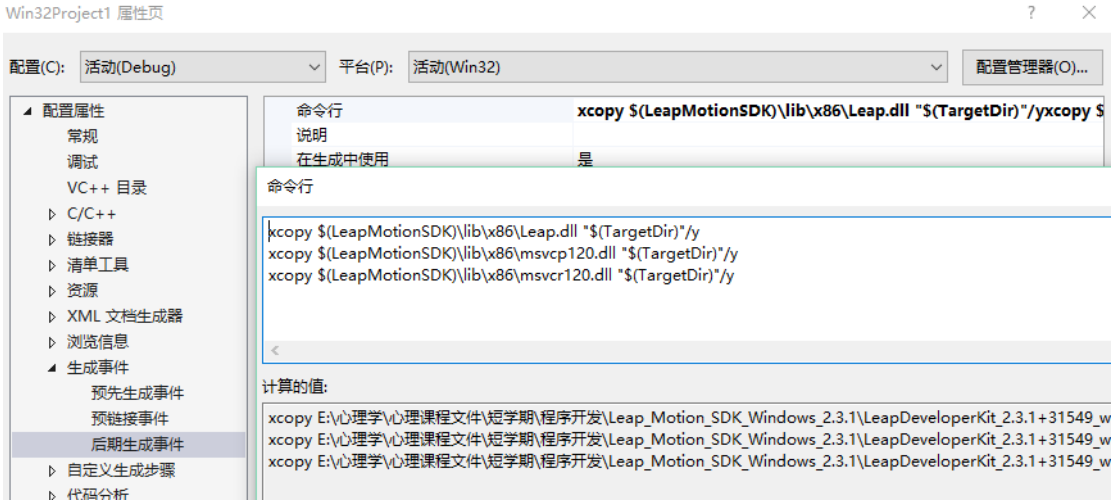
关键设置 3：链接器输入设置下—附加依赖项 Leap.lib（即 lib\x86 下的那个 Leap.lib 文件）



关键设置 4：该项设置关键又不关键，最后还需要将动态链接库复制到 Debug（或 Release）目录下，供可执行文件调用。

```
xcopy $(LeapMotionSDK)\lib\x86\Leap.dll "$(TargetDir)" /y
xcopy $(LeapMotionSDK)\lib\x86\msvcp120.dll "$(TargetDir)" /y
xcopy $(LeapMotionSDK)\lib\x86\msvcr120.dll "$(TargetDir)" /y
```

对于上述表述请完整复制粘贴，$括号里面的要与环境变量名称一致，TargetDir 外加双引号，结尾处/y 表示当已经存在时是否覆盖，y 表示是（覆盖）。



（如果更改为 Release，需要重新进行设置，Debug 和 Release 版某种程度属于不同项目）

3.Leap Motion 数据分析

3.1 Leap API

Leap API 是底层提供数据的，具体包括以下关键类（更多可查看 Leap.h 文件，LeapMath.h 内包含 Leap.h 涉及到数学相关定义）

Leap::Controller 控制器是连接应用程序和 Leap 设备之间的接口

Leap::Listener 监听器，用来处理 Leap 设备分发的消息

Leap::Frame 包含手和手指等的追踪数据

Leap::Hand 与 Leap::HandList 手的追踪数据、手列表数据

Leap::Finger 与 Leap::FingerList 手指追踪数据、手指列表数据

Leap::Vector 三维空间位置和方向矢量数据

Leap::Gesture 与 Leap::GestureList 手指的追踪数据、手势列表数据

3.2 两个对象

控制器对象 controller 是连接应用程序和 Leap 设备之间的主要接口，控制器对象会连接计算机的 Leap 设备，并获取 Leap 设备的追踪数据。

监听器 Listener 会绑定到创建的控制器对象上，并通过监听器类调用相关函数

3.3 获取追踪数据

Leap Motion 可以提供这么多数据，接下来需要学会如何去获取并加以利用，下面以 sample 为例进行解释。官方提供的三个解决方案，只有 2008 是配置好的，可以直接编译运行，连接 Leap 设备即可看到大量数据输出。

3.3.1 创建 controller 对象和监听器

每次调用前都必须声明 Leap.h 和 Leap 名称空间。

控制器对象 controller 是连接应用程序和 Leap 设备之间的主要接口，控制器对象 controller 会连接计算机的 Leap 设备，并通过 Leap::Frame 获取人手的追踪数据。

在主窗口创建的同时创建监听器、控制器对象，并且将监听器绑定到控制器上

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    // TODO: 在此放置代码。
    // Create a sample listener and controller
    SampleListener listener;
    Controller controller;
    // Have the sample listener receive events from the controller
    controller.addListener(listener);

    // 初始化全局字符串
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

实例化控制器对象并绑定监听器后，只要监听器检测到控制器状态发生改变、检测区域检测到新的帧数据，监听器对象就会自动调用 onFrame、onInit 等类函数。

```
    }
    // Remove the sample listener when done
    controller.removeListener(listener);
    return (int) msg.wParam;
}
```

在程序结尾处，或当窗口关闭时移除监听器，接触监听器与控制器对象的绑定。

3.3.2 获取设备状态

当 Leap 硬件设备与计算机的连接状态发生改变，控制器对象能够获取这种改变，通过 listener 监听。

void SampleListener::onInit(const Controller& controller)

初始化函数，创建监听器开始初始化时调用，不管设备是否连接

void SampleListener::onConnect(const Controller& controller)

实例化 controller 对象后，当 controller 对象连接到 Leap 设备，发送运动追踪数据时调用

void SampleListener::onDisconnect(const Controller& controller)

当 controller 对象与 Leap 设备断开连接的时候调用（拔出 Leap 设备或关闭 Leap 设备或图标处设置 Pause Tracking 时）

void SampleListener::onExit(const Controller& controller)

当监听器与 controller 对象分离的时候调用

void SampleListener::onFocusGained(const Controller& controller)

当手在检测区域获得检测焦点的时候调用，不清楚具体意思

void SampleListener::onFocusLost(const Controller& controller)

当手不在检测区域失去检测焦点的时候调用

void SampleListener::onDeviceChange(const Controller& controller)

当设备状态改变的时候调用，如设备连接和设备断开

void SampleListener::onServiceConnect(const Controller& controller)

当设备处于连接的时候调用

void SampleListener::onServiceDisconnect(const Controller& controller)

当设备断开连接的时候调用

3.3.3 获取追踪数据

实例化控制器对象并绑定监听器后，只要检测到新的帧数据，控制器对象自动调用 onFrame 等函数，用控制器类的 frame 方法获取追踪数据。

该函数可以检索人手列表信息并输出帧 ID、时间戳、手数量、手指数量、工具数量和手势等信息。

1）获取当前追踪数据和获取历史追踪数据

有些时候我们可能不仅仅希望获取当前帧数据，还想获取上一帧数据甚至上上帧，以进行比较之类的操作。

Frame frame = controller.frame(); //当前帧

Frame previousframe = controller.frame(1); //上一帧

frame(history)括号里面的 history 参数表示倒数第几帧 frame（history），一般最多追踪到倒数 60 帧

```
void SampleListener::onFrame(const Controller& controller) {
    // Get the most recent frame and report some basic information
    const Frame frame = controller.frame();
    std::cout << "Frame id: " << frame.id()
```

2）获取手的追踪数据

获取第一只手以及手的属性信息

```
// Get the first hand
const Hand hand = *hl;
std::string handType = hand.isLeft() ? "Left hand" : "Right hand";
std::cout << std::string(2, ' ') << handType << ", id: " << hand.id()
    << ", palm position: " << hand.palmPosition() << std::endl;
```

获取垂直手掌的矢量和手掌位置

```
// Get the hand's normal vector and direction
const Vector normal = hand.palmNormal();
const Vector direction = hand.direction();
```

计算手俯仰、旋转等角度数据

```
// Calculate the hand's pitch, roll, and yaw angles
std::cout << std::string(2, ' ') << "pitch: " << direction.pitch() * RAD_TO_DEG << " degrees,
    << "roll: " << normal.roll() * RAD_TO_DEG << " degrees, "
    << "yaw: " << direction.yaw() * RAD_TO_DEG << " degrees" << std::endl;
```

手臂骨骼数据

```
// Get the Arm bone
Arm arm = hand.arm();
std::cout << std::string(2, ' ') << "Arm direction: " << arm.direction()
    << " wrist position: " << arm.wristPosition()
    << " elbow position: " << arm.elbowPosition() << std::endl;
```

手指数据，包括手指位置、手指个数、手指骨骼数据

```
// Get fingers
const FingerList fingers = hand.fingers();
for (FingerList::const_iterator fl = fingers.begin(); fl != fingers.end(); ++fl)
{
    const Finger finger = *fl;
    std::cout << std::string(4, ' ') << fingerNames[finger.type()]
        << " finger, id: " << finger.id()
        << ", length: " << finger.length()
        << "mm, width: " << finger.width() << std::endl;

    // Get finger bones
    for (int b = 0; b < 4; ++b) {
        Bone::Type boneType = static_cast<Bone::Type>(b);
        Bone bone = finger.bone(boneType);
        std::cout << std::string(6, ' ') << boneNames[boneType]
            << " bone, start: " << bone.prevJoint()
            << ", end: " << bone.nextJoint()
            << ", direction: " << bone.direction() << std::endl;
    }
}
```

3）获取工具的追踪数据

```
// Get tools
const ToolList tools = frame.tools();
for (ToolList::const_iterator tl = tools.begin(); tl != tools.end(); ++tl) {
    const Tool tool = *tl;
    std::cout << std::string(2, ' ') << "Tool, id: " << tool.id()
        << ", position: " << tool.tipPosition()
        << ", direction: " << tool.direction() << std::endl;
}
```

4）获取手势的追踪数据

手势识别是需要开启的，例如 controller.enableGesture(Gesture::TYPE_CIRCLE)启用画圈手势；关闭该手势识别 controller.enableGesture(Gesture::TYPE_CIRCLE, false)

画圈手势：顺时针还是逆时针、进度属性（画圈的角度）

```
switch (gesture.type()) {
case Gesture::TYPE_CIRCLE:
{
    CircleGesture circle = gesture;
    std::string clockwiseness;
    if (circle.pointable().direction().angleTo(circle.normal()) <= PI / 2) {
        clockwiseness = "clockwise";
    }
    else {
        clockwiseness = "counterclockwise";
    }
    // Calculate angle swept since last frame
    float sweptAngle = 0;
    if (circle.state() != Gesture::STATE_START) {
        CircleGesture previousUpdate = CircleGesture(controller.frame(1).gesture(circle.id()));
        sweptAngle = (circle.progress() - previousUpdate.progress()) * 2 * PI;
    }
    std::cout << std::string(2, ' ')
        << "Circle id: " << gesture.id()
        << ", state: " << stateNames[gesture.state()]
        << ", progress: " << circle.progress()
        << ", radius: " << circle.radius()
        << ", angle " << sweptAngle * RAD_TO_DEG
        << ", " << clockwiseness << std::endl;
    break;
}
```

挥扫手势：挥扫的方向和速度

```
switch (gesture.type()) {
case Gesture::TYPE_CIRCLE:
{ ... }
case Gesture::TYPE_SWIPE:
{
    SwipeGesture swipe = gesture;
    std::cout << std::string(2, ' ')
        << "Swipe id: " << gesture.id()
        << ", state: " << stateNames[gesture.state()]
        << ", direction: " << swipe.direction()
        << ", speed: " << swipe.speed() << std::endl;
    break;
}
```

击键手势：击键位置和击键方向

```
switch (gesture.type()) {
case Gesture::TYPE_CIRCLE:
{ ... }
case Gesture::TYPE_SWIPE:
{ ... }
case Gesture::TYPE_KEY_TAP:
{
    KeyTapGesture tap = gesture;
    std::cout << std::string(2, ' ')
        << "Key Tap id: " << gesture.id()
        << ", state: " << stateNames[gesture.state()]
        << ", position: " << tap.position()
        << ", direction: " << tap.direction() << std::endl;
    break;
}
```

触屏手势：触屏位置和触屏方向

```
switch (gesture.type()) {
case Gesture::TYPE_CIRCLE:
{ ... }
case Gesture::TYPE_SWIPE:
{ ... }
case Gesture::TYPE_KEY_TAP:
{ ... }
case Gesture::TYPE_SCREEN_TAP:
{
    ScreenTapGesture screentap = gesture;
    std::cout << std::string(2, ' ')
        << "Screen Tap id: " << gesture.id()
        << ", state: " << stateNames[gesture.state()]
        << ", position: " << screentap.position()
        << ", direction: " << screentap.direction() << std::endl;
    break;
}
default:
    std::cout << std::string(2, ' ') << "Unknown gesture type." << std::endl;
    break;
}
```

4.程序开发

要实现用 Leap Motion 操作界面元素，最关键的一步是坐标映射实现，即手相对于 Leap Motion 的空间位置转化为屏幕闪的位置坐标。

```
//首先获取手相对于 leap 的位置求出检测区域
Vector palmPosition = hand.palmPosition();
float palmHeight = palmPosition.y;
float detectionWidth = (float)(palmHeight*tan(75.0 / 180.0 * PI) * 2);
float detectionHeight = 600;//有效监测高度
//获取手指尖位置坐标
Finger f1 = fingers[0];
Vector position = f1.tipPosition();
xp = position.x;
yp = position.y;
//屏幕大小
double screenWidth = 1375;
double screenHeight = 768;
//调节灵敏度,该数值越大越灵敏，移动较小距离也引起屏幕较大变化
sensitivity = 2;
detectionWidth = detectionWidth / sensitivity;
detectionHeight = detectionHeight / sensitivity;
//屏幕坐标映射
//leap 的中间为原点，故加 screenwidth、2
testnumberx = xp / detectionWidth*screenWidth + screenWidth / 2;
//600 为探测高度范围，参见笔者写的的使用说明书
testnumbery = screenHeight - yp / detectionHeight * screenHeight;
```

附录为：WFC 数据实时输出界面、LeapMotion 手势操作简易程序

5.联系方式

高在峰 zaifengg@gmail.com
李文敏 vinceli@zju.edu.cn