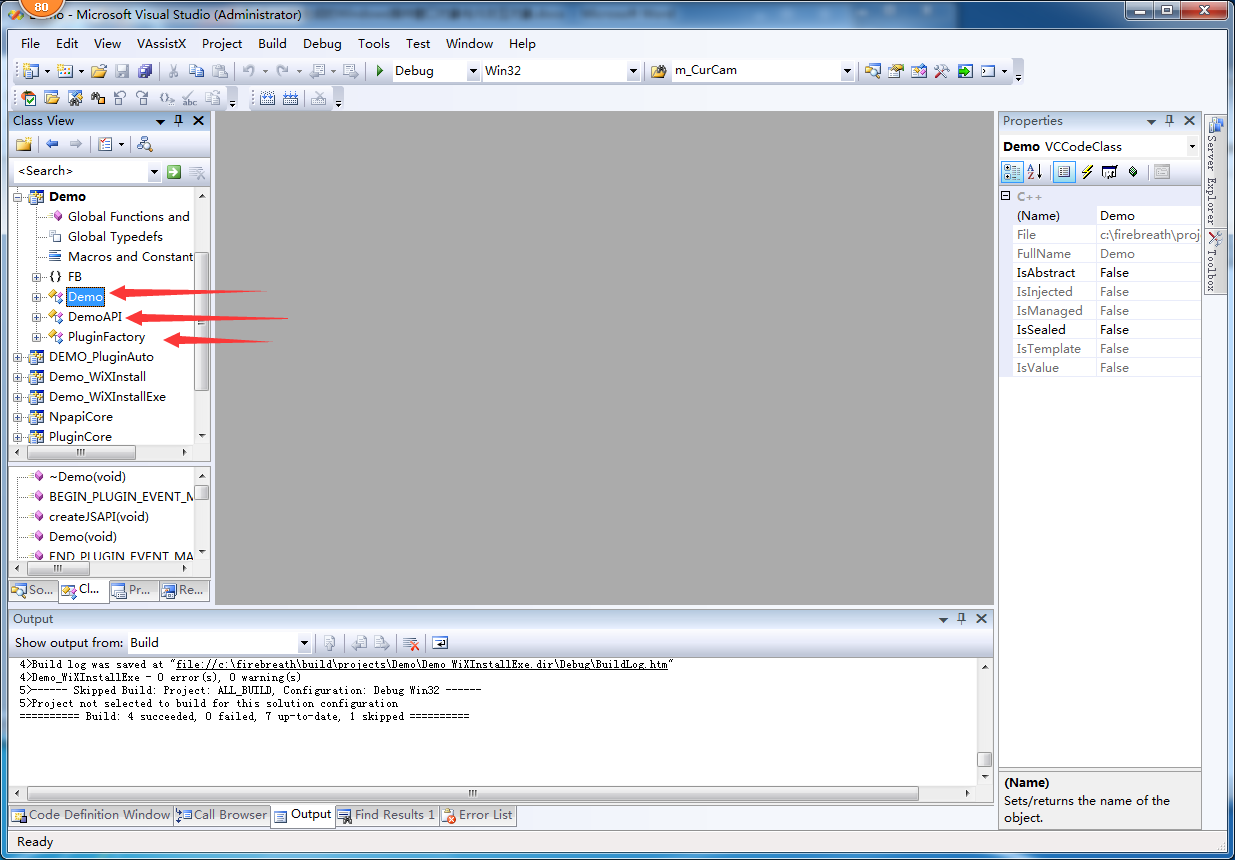
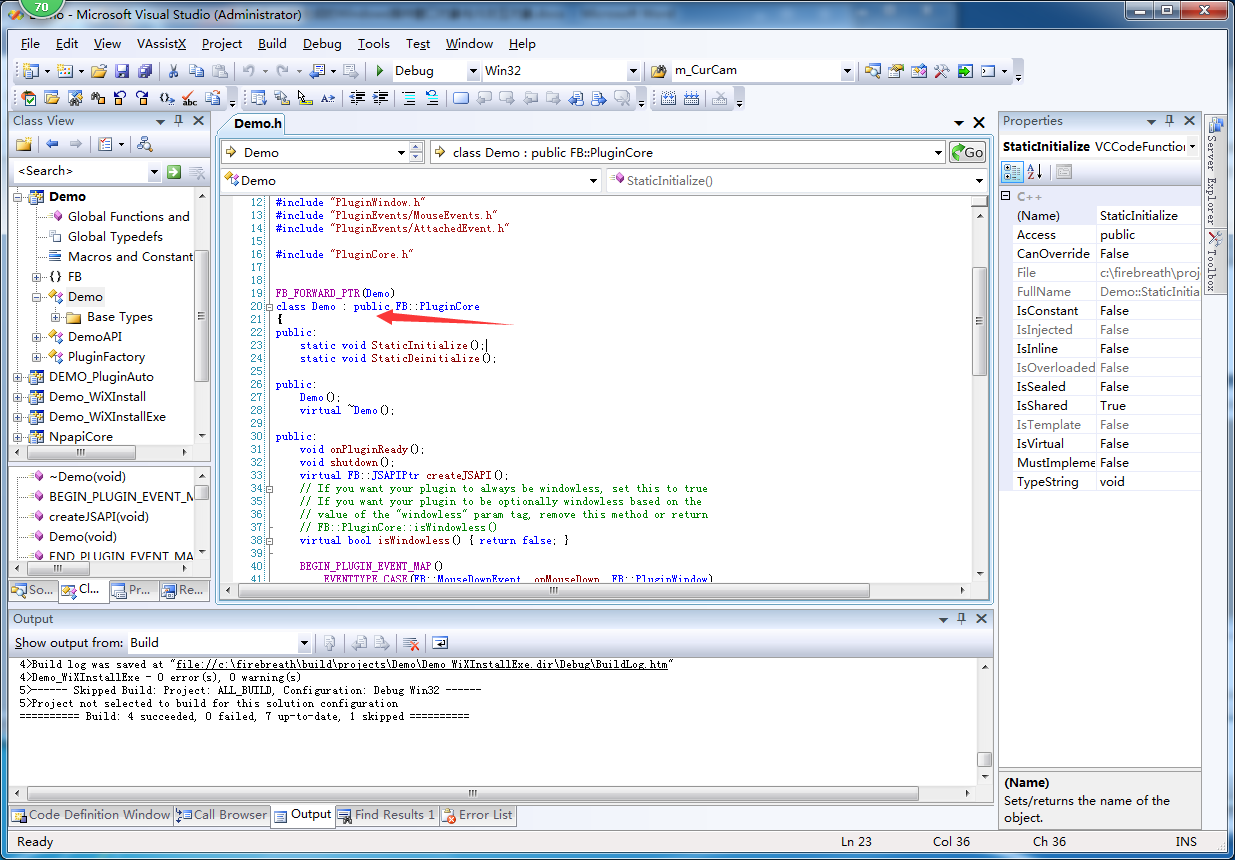
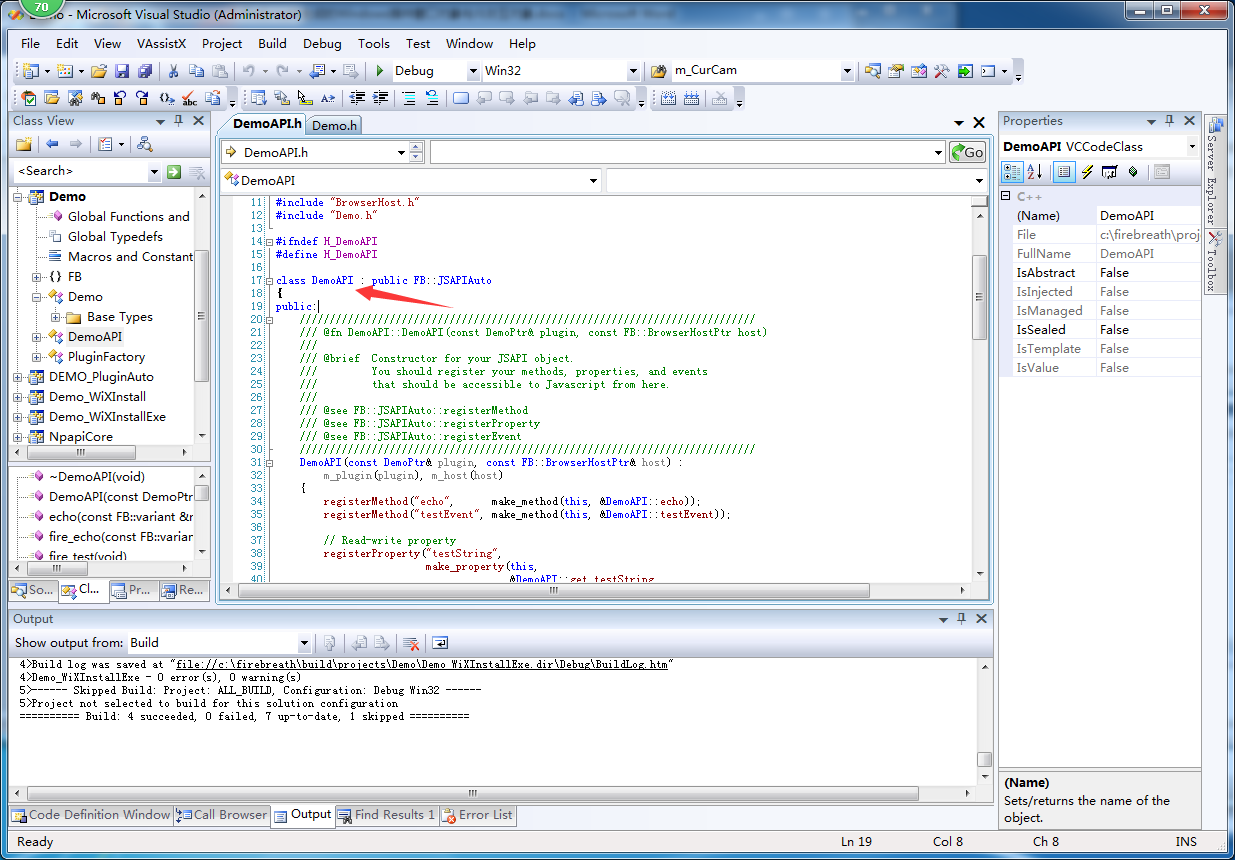
1. 查看Demo插件工程，如下图







对于Windows平台框架，Firebreath生成两个主要对象

1. FB::PluginCore派生类对象，主要与浏览器交互
2. FB::JSAPIAuto派生对象，主要与JS交互

注意：如何获取插件窗口句柄。

FB::PluginWindowWin的定义需要引用头文件：“#include "Win/PluginWindowWin.h"”

1. 在FB::PluginCore派生类对象中，可以通过下面方法得到插件窗口句柄：

FB::PluginWindowWin \*pwnd = getWindow()->get\_as<FB::PluginWindowWin>();

HWND hwnd = pwnd->getHWND();

1. 对于FB::JSAPIAuto派生对象，可以通过getPlugin()方法得到对应的FB::PluginCore派生对象，你可以用下面方法得到插件的窗口句柄：

FB::PluginWindowWin \*pwnd = getPlugin()->Get­Window()->g­et\_as<FB::Plu­­ginWind­ow­Win>();

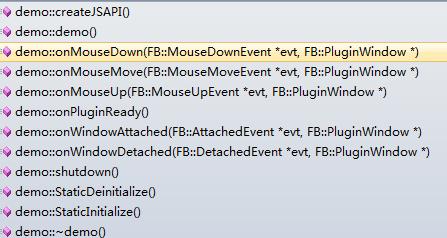
HWND hwnd = pwnd->getHWND();

**实现一个简单的功能(与浏览器交互)**

**转自**<http://www.blogjava.net/xiaomage234/archive/2012/08/28/386428.html>

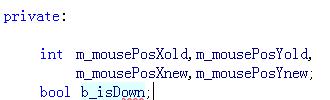
其实上面这个例子已经实现了很多比较实用的功能了，可是我比较菜，只能看懂一小部分。我想实现一个用鼠标在plugin窗口上画线的功能。经过摸索，终于可以实现我想实现的这个功能了。下面是我的步骤：

看程序的过程中我发现已经实现了鼠标按下，抬起，移动等事件的函数，但是函数体只有一个return语句，下图是所有已经实现的函数：

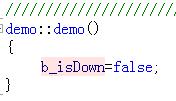


看到这些函数，我想画线的函数就可以利用这三个鼠标事件来来完成吧。于是我做了如下工作。

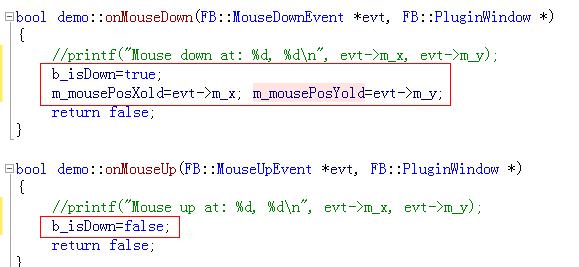
1，定义几个私有成员变量（demo.h文件中）：



2，在构造函数中初始化一下（demo.cpp文件中）：



3，先写onMouseDown和onMouseUp两个函数中的代码（作用：修改鼠标是否按下的状态）。如下图，框住的是添加的内容：



4，再来写鼠标移动时实现的功能，首先获取鼠标指针所在的点，与前面一样：m\_mousePosXnew=evt->m\_x; m\_mousePosYnew=evt->m\_y;就搞定了；然后就是画线的功能了（如果你对windows编程很熟悉就直接看后面的程序截图吧，因为我不熟悉，所以我将这个过程描述得详细一点，请理解），从old 的点画到new这个点。我虽然对windows程序开发不是很熟练，但是还是知道有个LineTo的函数，因此先就假设是LineTo(old,new)吧。于是就接下来写：

if (b\_isDown)

{

LineTo(old, new);

}

最后，将new这一点保存为old的点：

m\_mousePosXold= m\_mousePosXnew; m\_mousePosYold= m\_mousePosYnew;

所以这里就剩下搞清楚 LineTo这个函数了，要实现从一个点画线到另一个点在windows中其实是这样实现的：

HDC hDC;

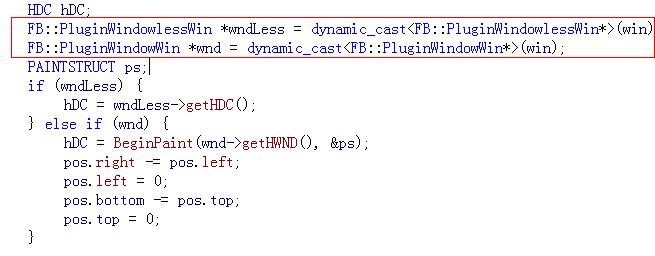
hDC=GetDC(hWnd);

MoveToEx(hDC,m\_mousePosXold,m\_mousePosYold,NULL);

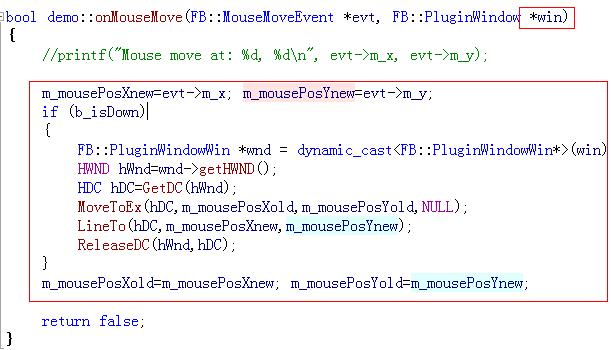
LineTo(hDC,m\_mousePosXnew,m\_mousePosYnew);

ReleaseDC(hWnd,hDC);

上面这样实现了，要使用HDC以及MoveToEx和LineTo需要包含头文件PluginWindowWin.h。还差个HWND，如何得到这个HWND，我也说不清楚，我是看到例子FBTestPlugin中的代码才知道可以这样获取的：FB::PluginWindowWin \*wnd = dynamic\_cast<FB::PluginWindowWin\*>(win); HWND hWnd=wnd->getHWND();下面是FBTestPlugin 的FBTestPlugin.cpp文件中的代码片段截图：

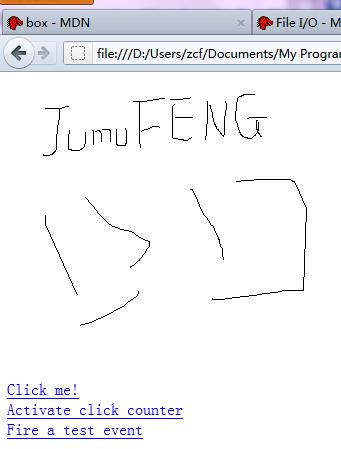


经过尝试，终于完成了我想要实现的这个简单功能，再次感慨一下开发plugin对windows编程的基础能力还是有相当需求的。下面是完整的onMouseMove函数代码截图，框住的是添加的内容（注意在文件最前面添加头文件的引用#include "PluginWindowWin.h"）：



做完这些修改之后，在demo这个项目上选择仅用于项目->仅重新生成demo。等生成完毕之后，去前面的位置找到npdemo.dll复制到Firefox相应profile下的plugins目录，重新启动Firefox，打开那个测试页面。

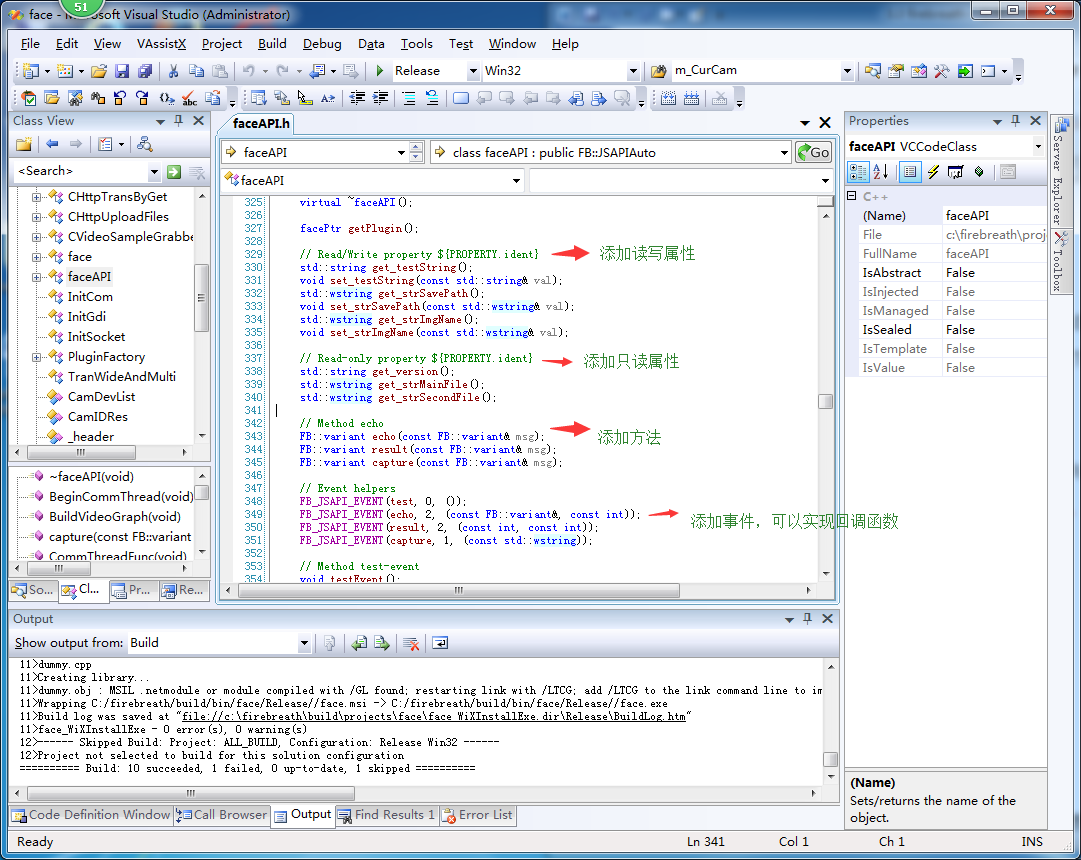
下面是运行效果，请忍受我的涂鸦，\(^o^)/~

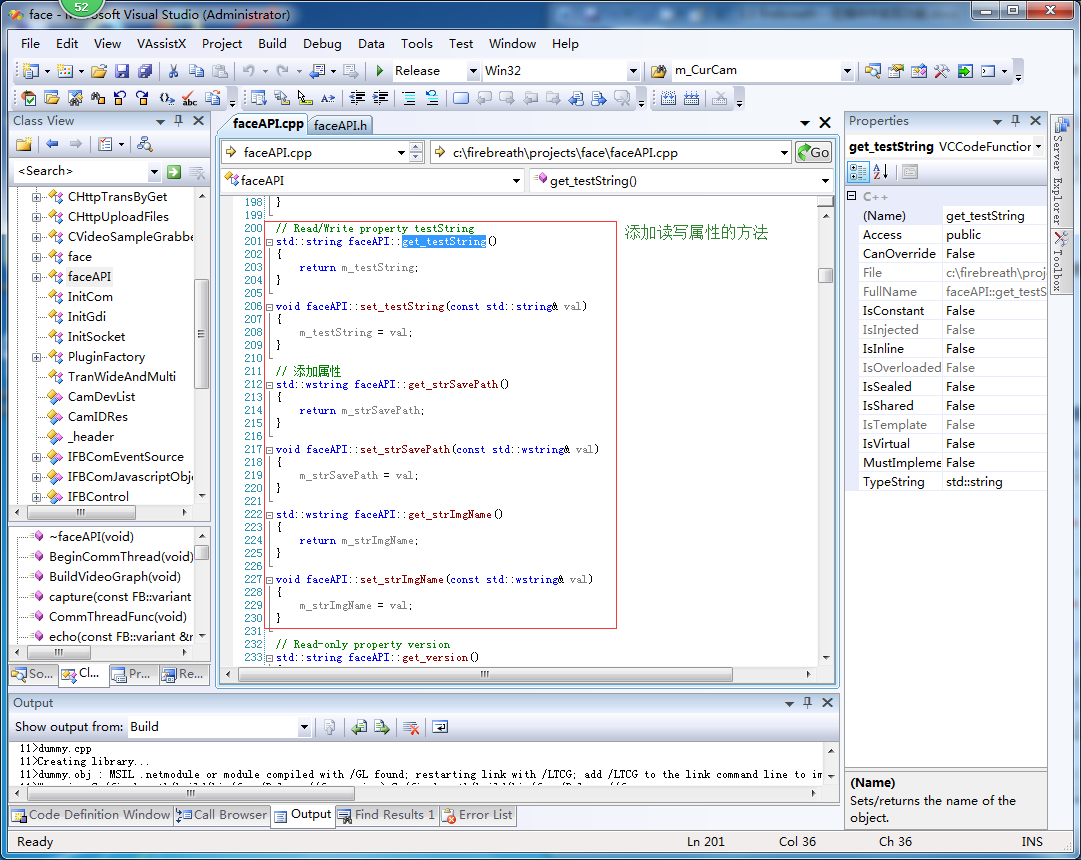


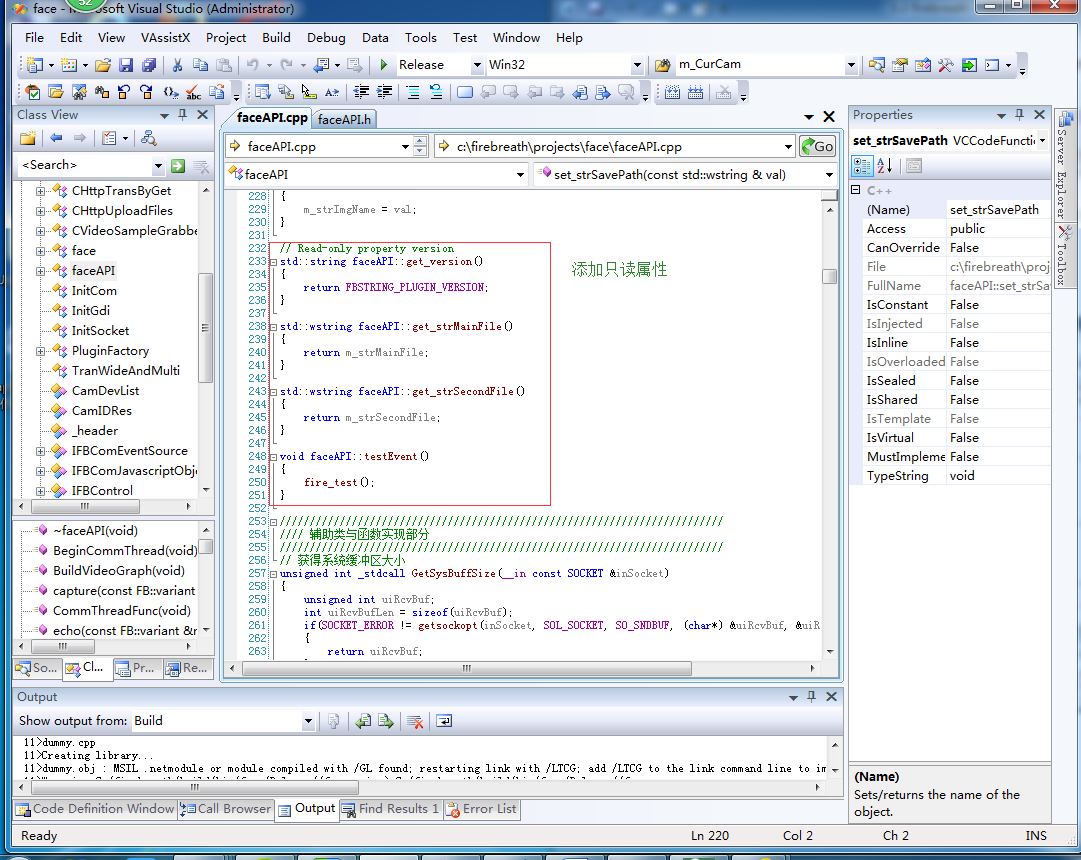
差不多了吧，这个简单的功能就这样了。或许再添加一个鼠标移出区域的事件，在其中将鼠标按下状态设为false会更好。

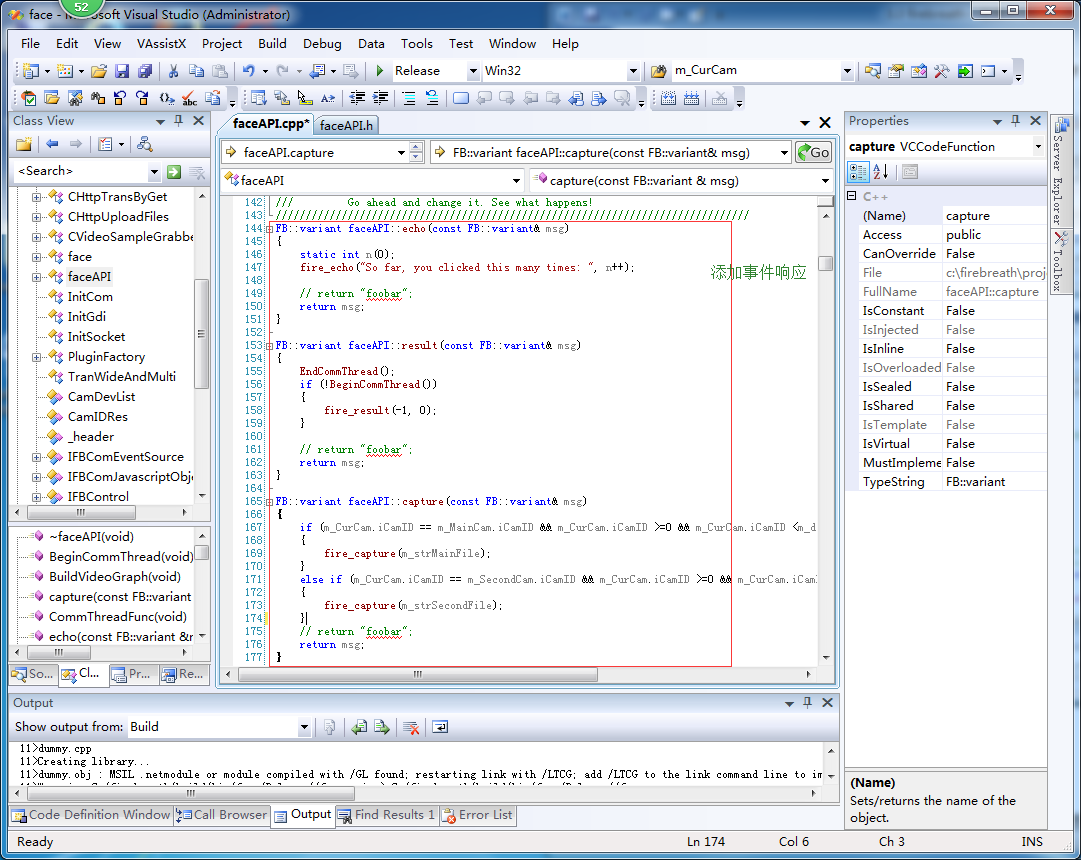
FB::JSAPIAuto派生对象，主要与JS交互。

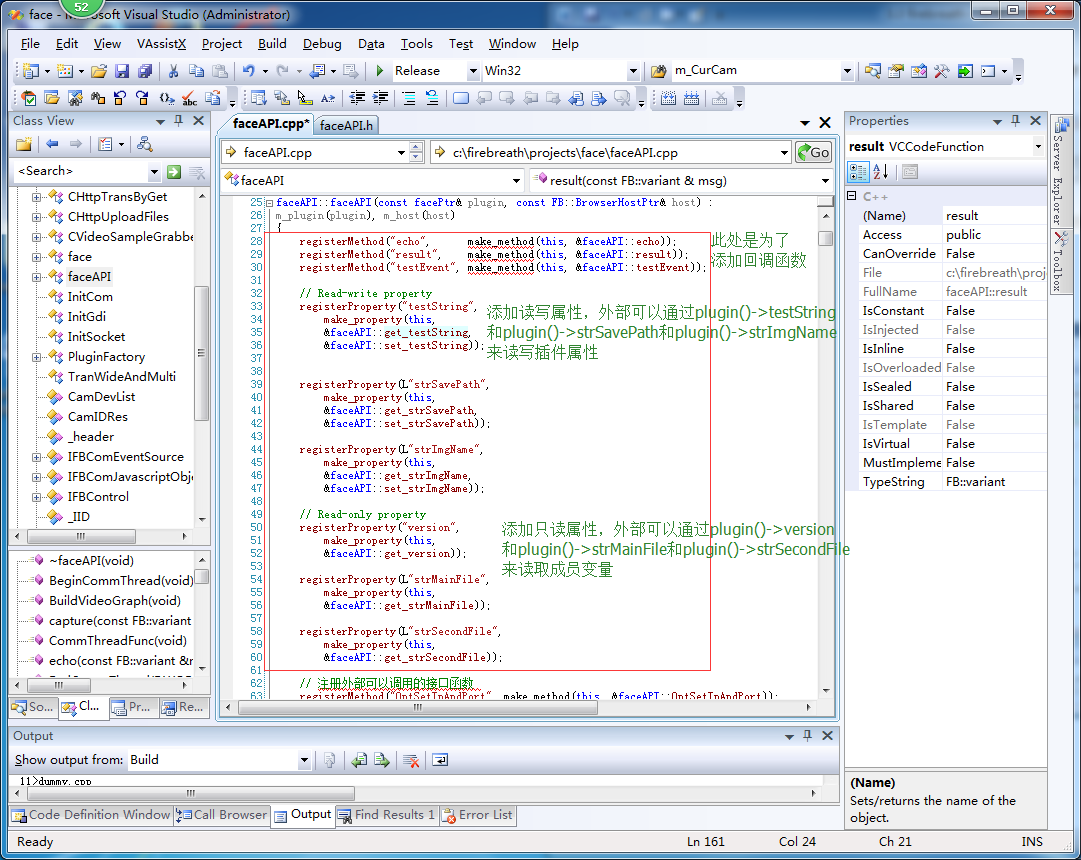
注意：在插件中添加属性和方法以及回调函数，特别要注意类型是否与JS兼容。下面多张图片展示了这个方法。

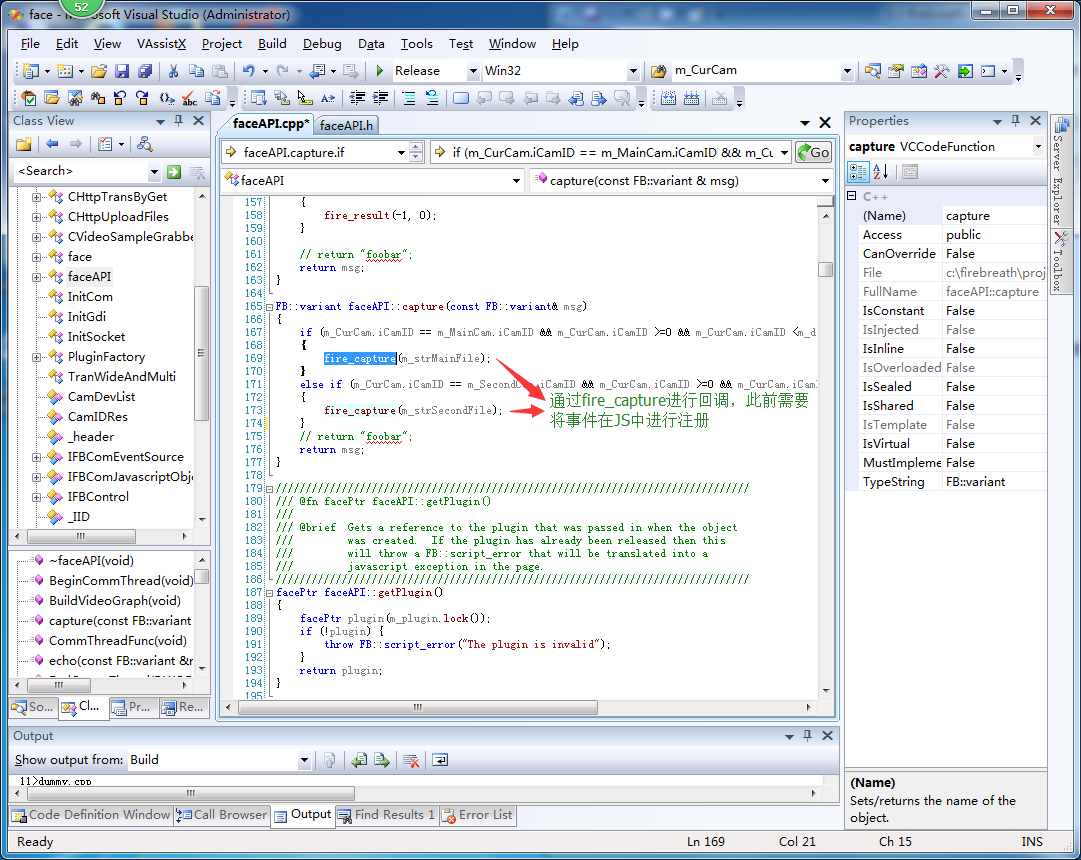


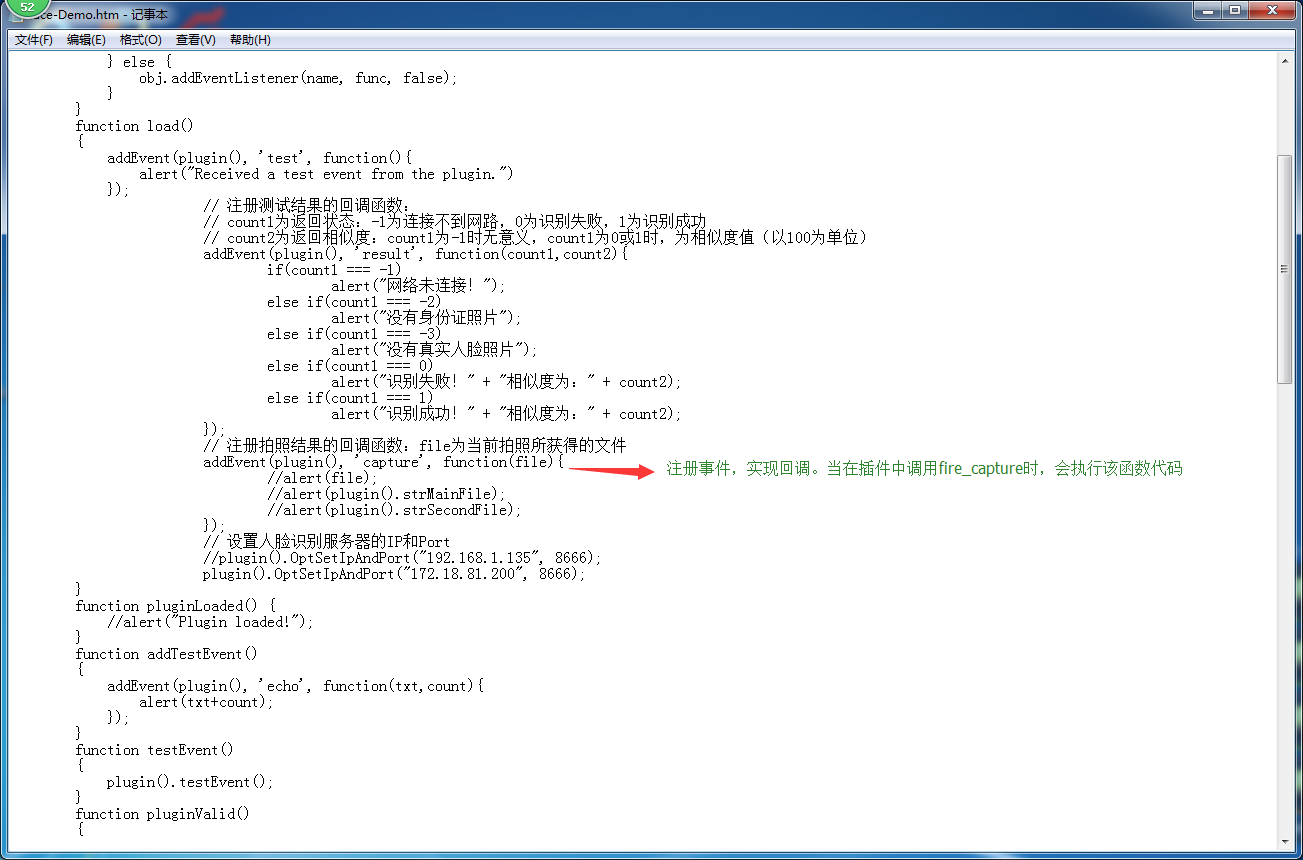


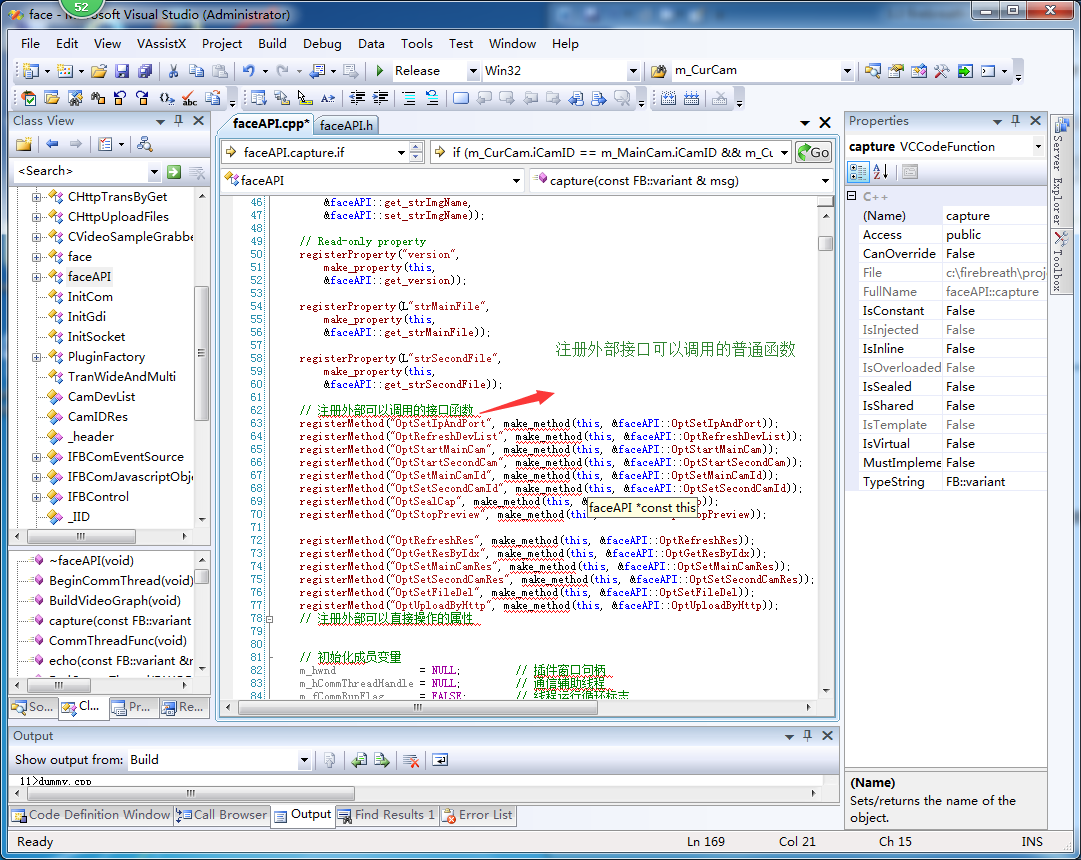












上面的图片展示了如果添加可读可写属性，只读属性，回调函数和普通函数，且所有方法都是可行的。下面，我详细的列举实现方法。

1. 添加可读可写属性

如要实现在JS中读写插件中某个字符串属性strSavePath。

plugin().strSavePath = "My Name"; // 写字符串strSavePath

alert(plugin().strSavePath); // 读字符串strSavePath

在派生类中添加成员函数和变量（FB::JSAPIAuto派生类）

std::wstring get\_strSavePath();

void set\_strSavePath(const std::wstring& val);

std::wstring m\_strSavePath; // 照片文件存储目录

在cpp文件中实现

// 添加属性

std::wstring faceAPI::get\_strSavePath()

{

return m\_strSavePath;

}

void faceAPI::set\_strSavePath(const std::wstring& val)

{

m\_strSavePath = val;

}

在构造函数中，注册该属性，其中strSavePath即可被外部调用

registerProperty(L"strSavePath",

make\_property(this,

&faceAPI::get\_strSavePath,

&faceAPI::set\_strSavePath));

1. 添加只读属性

plugin().strMainFile= "My Name"; // 错误，属性只读

alert(plugin().strMainFile); // 读字符串strSavePath

在派生类中添加成员函数和变量（FB::JSAPIAuto派生类）

std::wstring get\_strMainFile();

std::wstring m\_strMainFile; // 主摄像头拍摄的照片完整路径

在cpp文件实现

std::wstring faceAPI::get\_strMainFile()

{

return m\_strMainFile;

}

在构造函数注册只读属性

registerProperty(L"strMainFile",

make\_property(this,

&faceAPI::get\_strMainFile));

1. 添加JS中可调用函数

派生类中添加方法

int OptSetIpAndPort(string szIp, int szPort);// 设置人脸识别处理服务器

添加实现部分

int faceAPI::OptSetIpAndPort(string szIp, int szPort)

{

m\_strIp = szIp; // 服务器地址

m\_iPort = szPort; // 服务器端口号

return 0;

}

在构造函数中注册该方法

// 注册外部可以调用的接口函数

registerMethod("OptSetIpAndPort", make\_method(this, &faceAPI::OptSetIpAndPort));

如果不需要方法是用于内部调用，则不需要在构造函数中进行注册。

1. 添加回调函数

主要是通过注册事件的方法来实现回调函数功能。

JS中代码：注册事件

// 注册拍照结果的回调函数：file为当前拍照所获得的文件

addEvent(plugin(), 'capture', function(file){

//alert(file);

//alert(plugin().strMainFile);

//alert(plugin().strSecondFile);

});

在派生类中添加方法

FB::variant capture(const FB::variant& msg); // 添加事件

FB\_JSAPI\_EVENT(capture, 1, (const std::wstring));// 添加回调，第二个参数表示回调函数有1个参数，第三个参数表示参数类型

在Cpp文件中实现

FB::variant faceAPI::capture(const FB::variant& msg)

{

return msg;

}

如何实现回调，只要在函数中调用fire\_capture(L“”);来实现回调函数的功能。