Ps插件

概述

PS扩展插件开发分 3 种

- ExtendScript脚本
- 8li滤镜插件
- CEP扩展

区别

ExtendScript脚本

> 是adobe提供的自动化脚本，提供DOM来操作软件的各种功能，开发语言选择：

1. JavaScript
2. AppleScript
3. VBScript

8li插件

> adobe photoshop sdk开发的一个dll，可以直接操作photoshop里面的像素。滤镜插件一般使用这个来开发 。

Cep

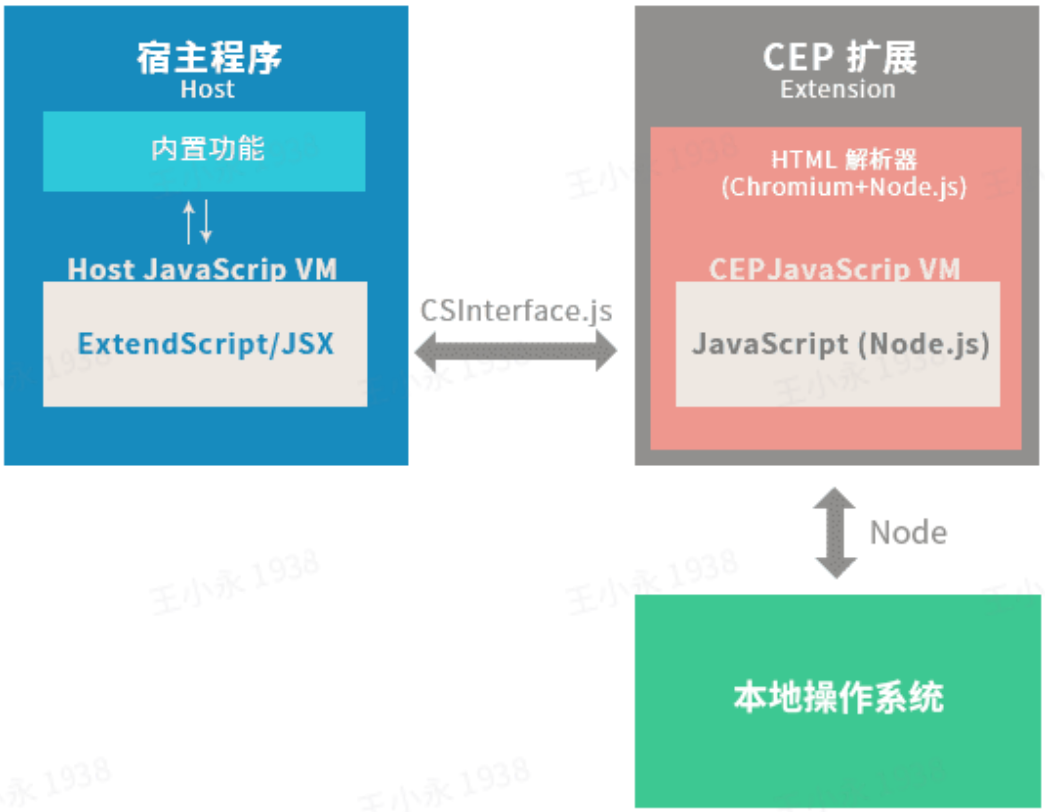
> cep，即通用扩展平台。现在常用的Photoshop扩展，都是cep扩展，官方推荐的就是这种方式。

CEP

- 很久以前使用flash开发，还有个拖控件的工具，现在不用这种方式了，一是因为以前的控件开发工具仅支持到Photoshop CC，现在的版本不支持这个开发工具了。二是因为adobe后来放弃了flash，转而使用了html5。从Photoshop CC 2014以后的版本，cep开发都使用html5+node.js。
- cep实际上对应的就是cef（内嵌的chromium），是支持跨平台的
- 现在使用html5+node.js开发的cep扩展是一个本地运行的web应用，面板实际是一个网页
- 而cep扩展的操作Photoshop的方式，是使用ExtendScript。

CEP架构

> CEP 上运行的实际上一个可以与宿主程序（比如 Photoshop）进行交互的Web APP，它的界面是由 HTML5 网页构成，通过 JavaScript 调用 ExtendScript 与宿主交互（如操作图层），通过 Node.js 与本地操作系统交互(如读写文件、调用本地程序)。



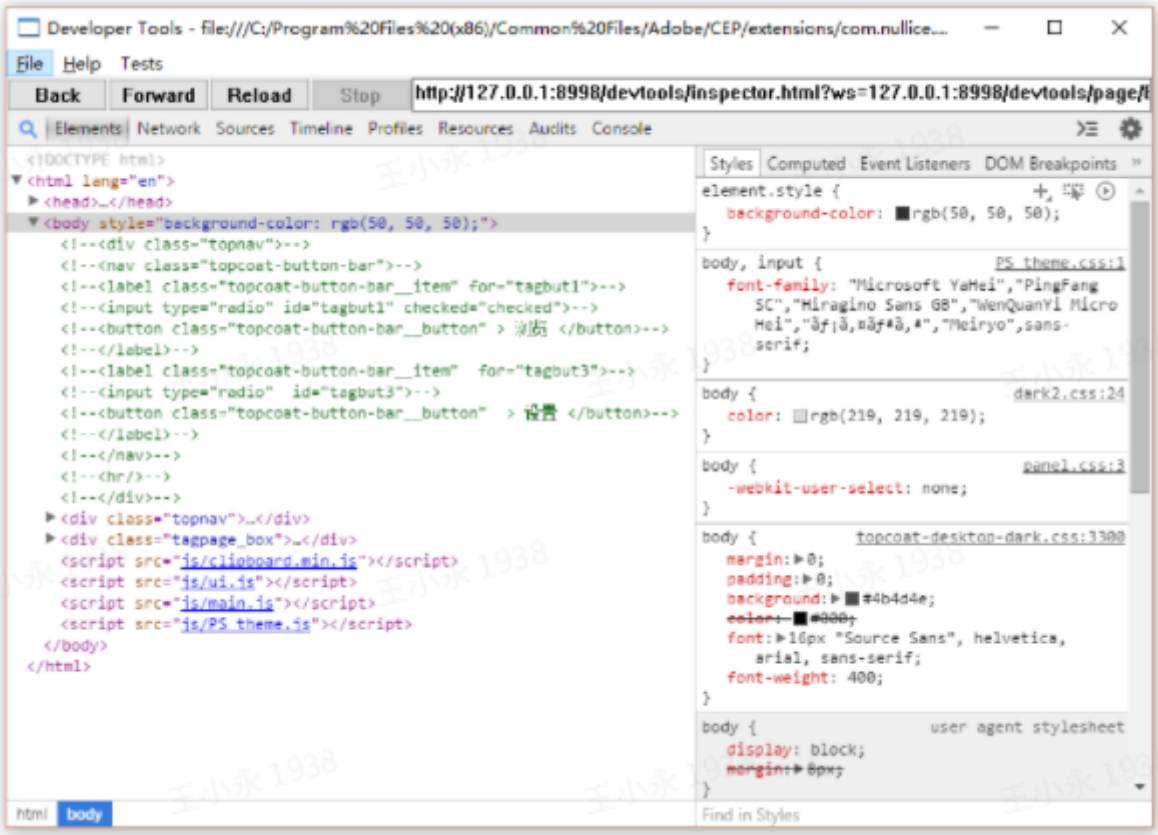
开发环境

界面代码

- vs code等网页开发软件。

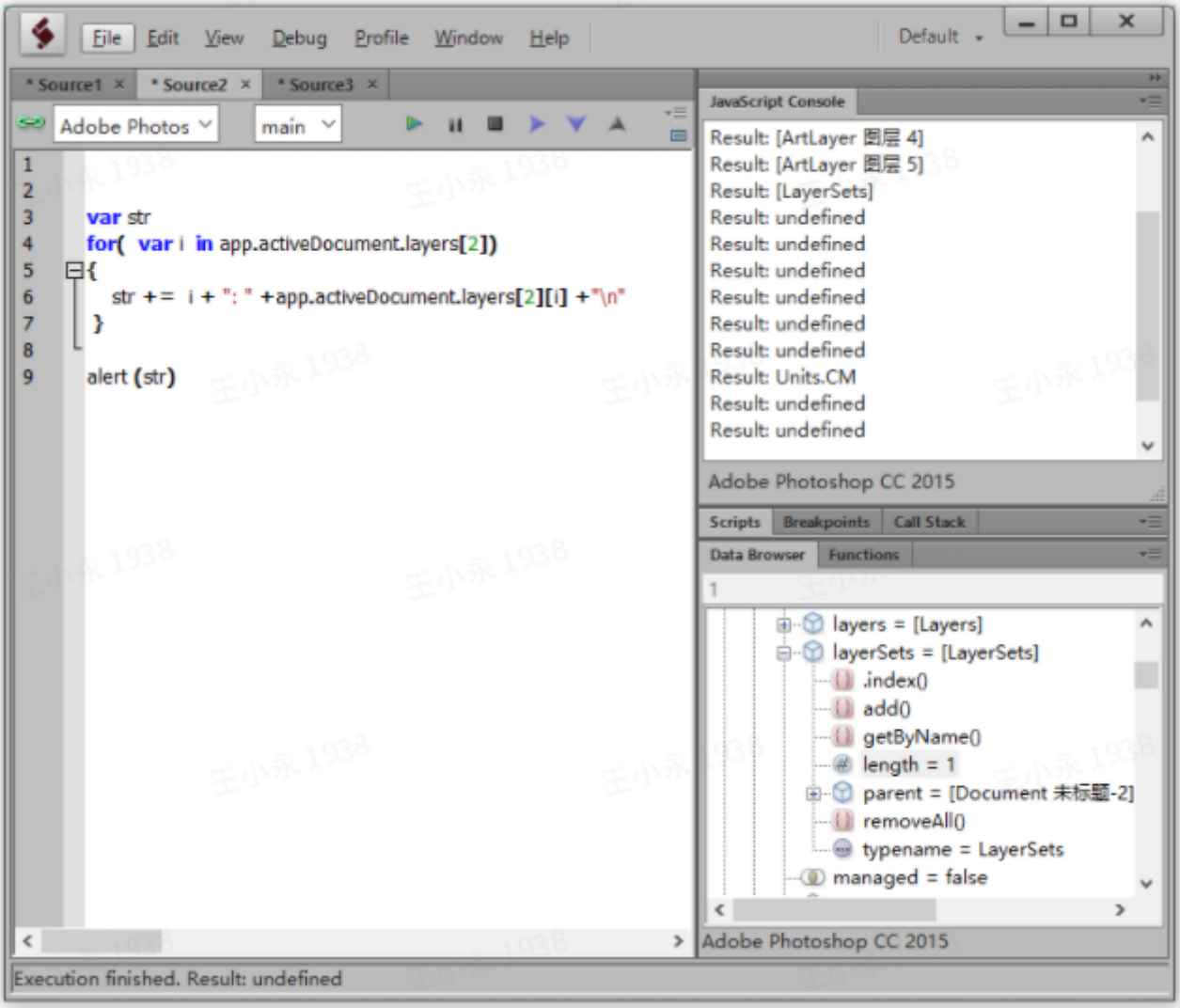
调试相关

> cep扩展支持远程调试，可以在浏览器中打开远程调试页面，不过cep 6.1开始，用主流版本的chrome调试bug比较多，所以需要下载cef clent。
>
> https://github.com/Adobe-CEP/CEP-Resources/tree/master/CEP_10.x/Cefclient_v74



关于extendscript

- 使用 Adobe ExtendScript Toolkit CC
- 用于测试extendscript版本



Node.js/IO.js

> 安装node.js或io.js不是必须的，因为cep的宿主程序自己带有node.js或io.js，开发测试时可以直接使用。

配置开发环境

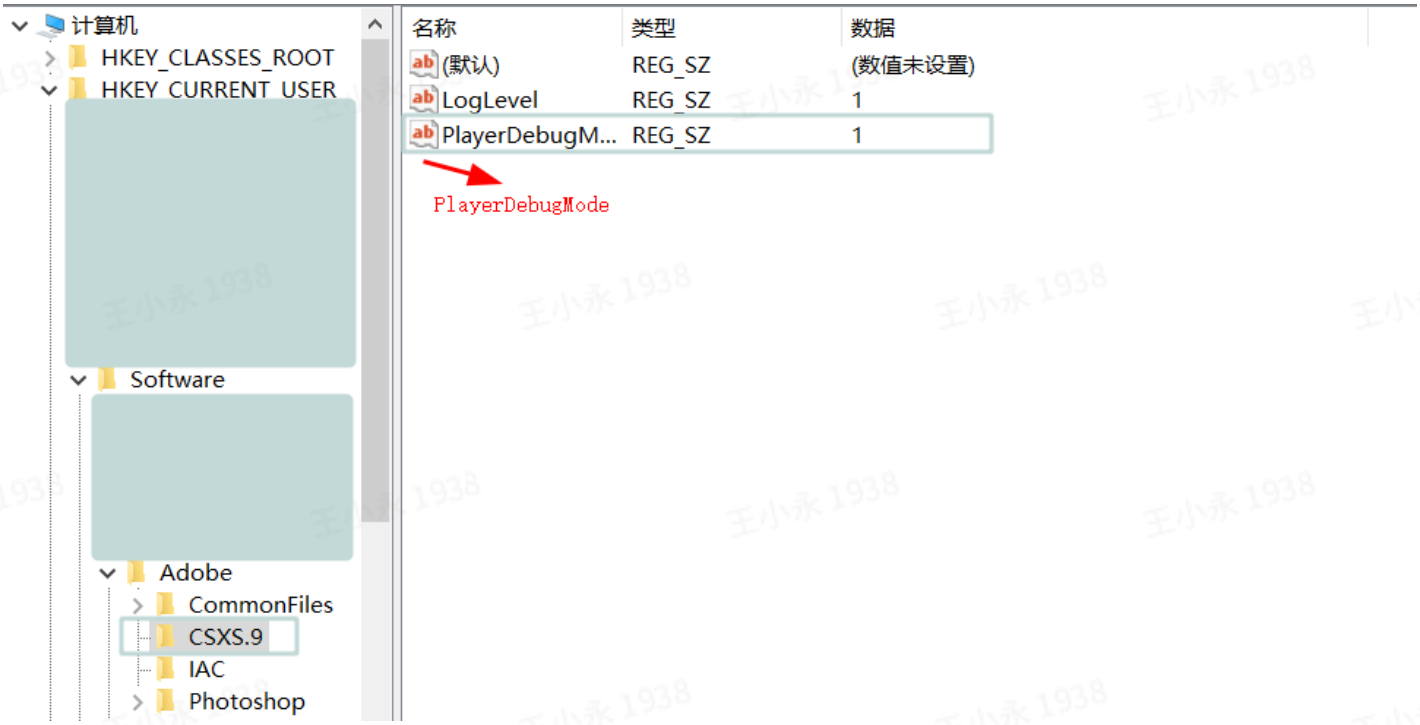
1. 通常情况下，宿主程序不会运行未经签名的扩展，只有打包并签名才可以运行
2. 开发的时候可以开启开发者模式，通过修改注册表来开启

XML

```
//cc,cc 2014
HKEY_CURRNET_USER\Software\Adobe\CSXS.5
//cc 2015
HKEY_CURRNET_USER\Software\Adobe\CSXS.6
//cc 2015.5
HKEY_CURRNET_USER\Software\Adobe\CSXS.7
//我的ps cc 2019
HKEY_CURRNET_USER\Software\Adobe\CSXS.9

--------------------------------------------------------------------------

//添加字段 1为打开，0为关闭。
PlayerDebugMode = 1
```



工作目录

1. 建立一个工作目录，放在宿主程序特定位置，在宿主程序启动时，会被载入
2. 路径如下：

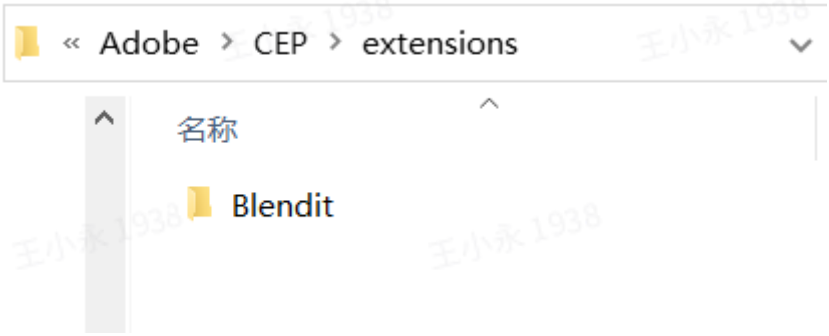
XML

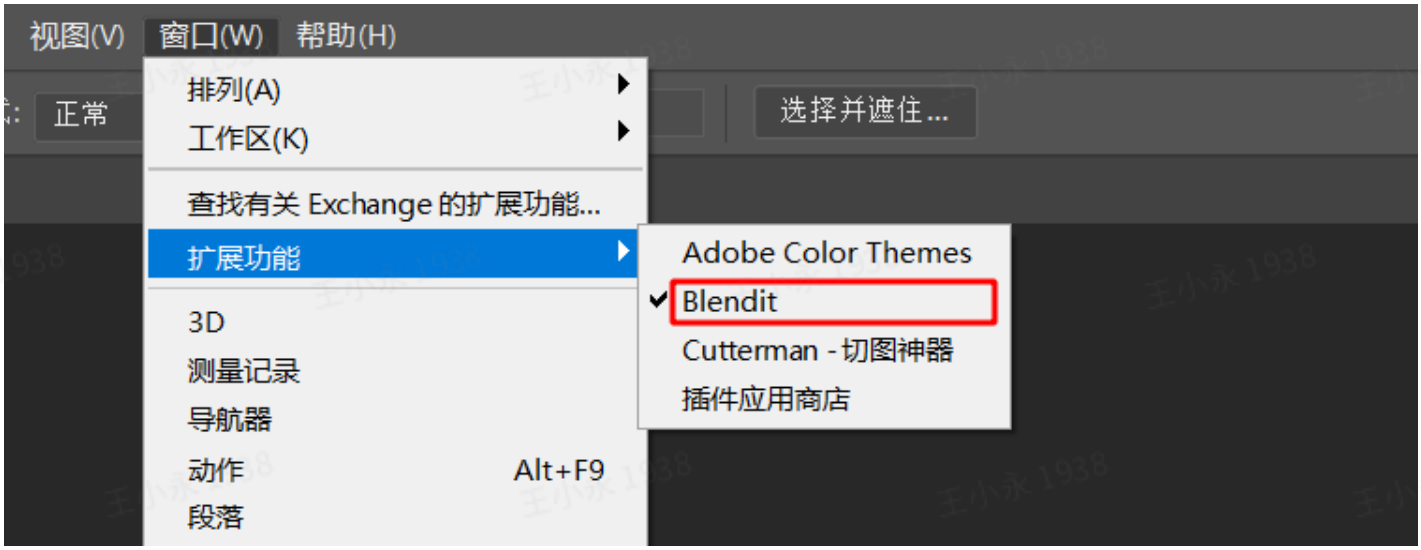
```
//windows 32
C:\Program Files\Common Files\Adobe\CEP\extensions\

//windows 64
C:\Program Files (x86)\Common Files\Adobe\CEP\extensions\

//windows通用位置
C:\Users\系统用户名\AppData\Roaming\Adobe\CEP\extensions\
```

3. 效果示例如下：





CEP插件示例教程

目录结构

> 在`C:\Program Files (x86)\Common Files\Adobe\CEP\extensions`路径下建立了一个liebao_browser,结构如下：

> 其中，比较关键的是csxs里面的manifest.xml文件，这个是必须要有的文件。

manifest.xml

> manifest.xml里面写的是扩展的相关配置信息，包括
>
> 1. 扩展名称
> 2. 版本信息
> 3. 允许运行的ps的版本
> 4. 入口文件
> 5. ...
> 6. 详细如下：

XML

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ExtensionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ExtensionBundleId="liebao_browser" ExtensionBundleVersion="1.0" Version="6.0">
  <!-- MAJOR-VERSION-UPDATE-MARKER -->
  <ExtensionList>
    <Extension Id="liebao_browser" Version="1.0"/>
    <!-- 设置扩展 ID-->
  </ExtensionList>

  <ExecutionEnvironment>
    <HostList>
      <!-- 设置扩展能在 11.0 版本之后 PhotoShop 中运行-->
      <Host Name="PHXS" Version="[11.0,99.9]"/>
      <Host Name="PHSP" Version="[11.0,99.9]"/>
    </HostList>

    <LocaleList>
      <Locale Code="All"/>
    </LocaleList>

    <RequiredRuntimeList>
      <RequiredRuntime Name="CSXS" Version="6.0"/>
    </RequiredRuntimeList>
  </ExecutionEnvironment>

  <DispatchInfoList>
    <Extension Id="liebao_browser">
      <!-- 为 liebao.browser 设置属性-->
      <DispatchInfo>
        <Resources>
          <MainPath>./index.html</MainPath>
```

```
            <MainPath>./index.html</MainPath>
            <!-- 指定起始载入的网页-->
            <ScriptPath>./jsx/main.jsx</ScriptPath>
            <!-- 指定用到的 JSX 文件-->
          </Resources>

          <Lifecycle>
            <AutoVisible>true</AutoVisible>
            <!-- 设置扩展面板为可视-->
            <StartOn>
            </StartOn>
          </Lifecycle>

          <UI>
            <Type>Panel</Type>
            <!-- 设置扩展显示为面板模式-->
            <Menu>Liebao browser</Menu>
            <!-- 设置扩展标题-->
            <Geometry>
              <Size>
                <!-- 设置扩展面板尺寸-->
                <Height>300</Height>
                <Width>600</Width>
              </Size>
              <MaxSize>
                <Height>600</Height>
                <Width>1200</Width>
              </MaxSize>
              <MinSize>
                <Height>300</Height>
                <Width>300</Width>
              </MinSize>
            </Geometry>
            <Icons>
              <!-- 设置扩展面板尺寸-->
              <Icon Type="Normal">./img/icon1.png</Icon>
              <Icon Type="DarkNormal">./img/icon1.png</Icon>
            </Icons>
          </UI>

        </DispatchInfo>
      </Extension>
    </DispatchInfoList>
</ExtensionManifest>
```

Html

> cep的界面是html，在manifest.xml中定义了html的入口文件为index.html

HTML

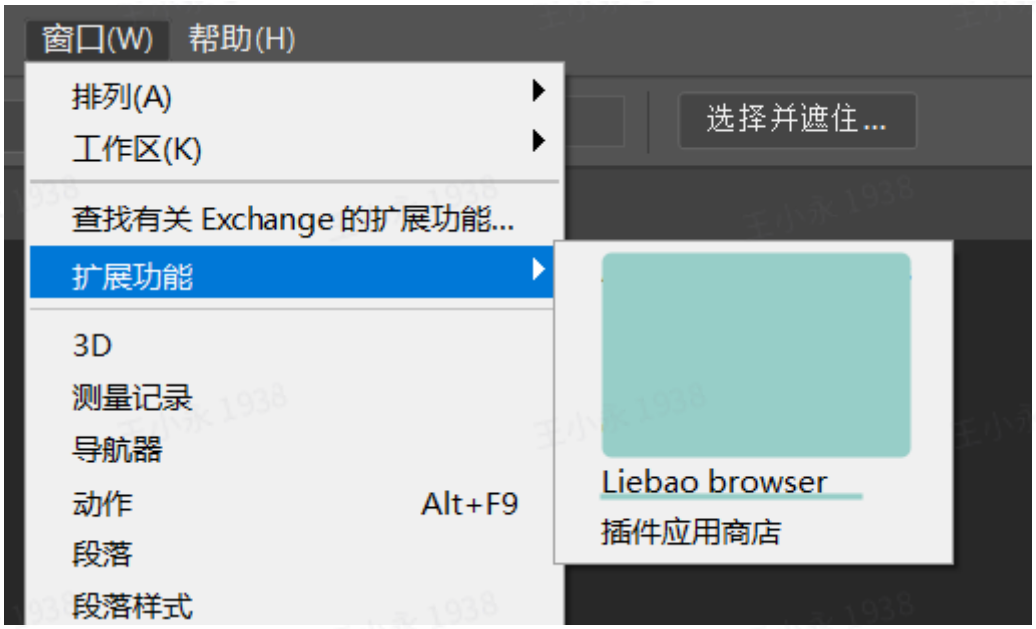
```HTML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html" charset="UTF-8">
  <link href="./css/styles.css" type="text/css" rel="stylesheet">
      <script type="text/javascript" src="./js/liebao.js"></script>
      <script type="text/javascript" src="./js/CSInterface.js"></script>
      <script type="text/javascript" src="./js/Vulcan.js"></script>
      <script type="text/javascript" src="./js/AgoraLib.js"></script>
  </head>

  <body style="background-color: #a2a1a3; text-align: center;" >
  <span style="font-family: '微软雅黑'; font-style: normal; font-weight: normal; font-size: 16pt; color: white">Liebao</span>
  <br>
    <span >
      <span style="font-family: 'Castellar'; font-style: normal; font-weight: normal; font-size: 34pt;">liebao browser</span>
    </span>
  <br>
  <br>
  <input type="button" value="liebao browser" onclick="pop()">

  </body>
</html>
```

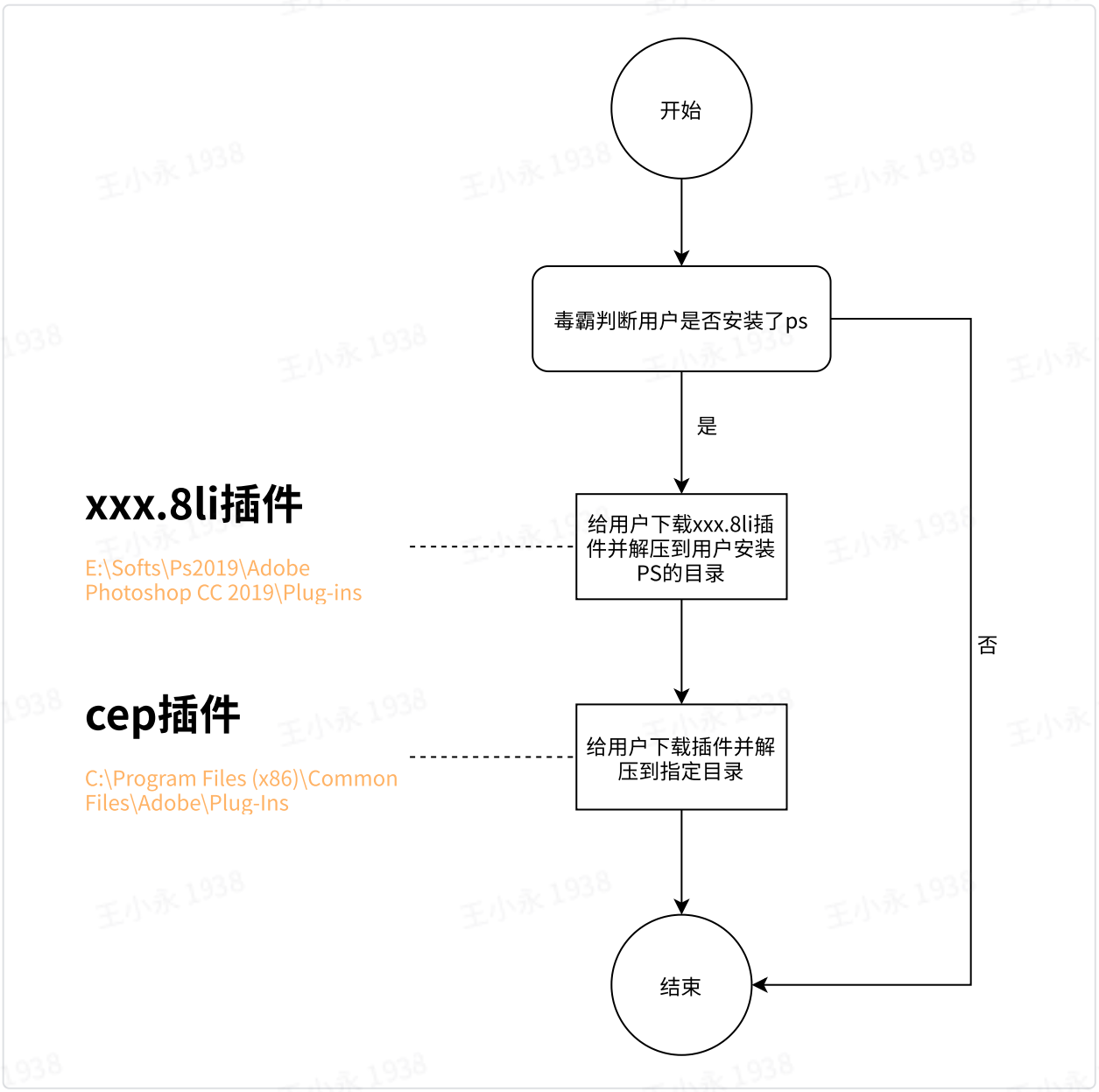
> 并在js目录下定义了用到的liebao.js、
>
> 同时，从下面地址拿到了CSInterface.js，这个是比较新的版本。
>
> https://github.com/Adobe-CEP/CEP-Resources/tree/master/CEP_10.x/Cefclient_v74
>
> CSInterface.js的作用是作为中间的桥梁，让运行在cep vm里面的js代码能够间接调用PS的功能

快捷工具栏效果


窗口(W)
帮助(H)
排列(A)
工作区(K)
查找有关 Exchange 的扩展功能...
扩展功能
3D
测量记录
导航器
动作
Alt+F9
段落
段落样式
选择并遮住...
Liebao browser
插件应用商店

关于静默安装

> 如果可以的话。



与C++通信的介绍

1. 可以利用adobe的sdk（pluginsdk）开发以xxx.8li，xxx.8bx之类的后缀结尾的插件，这种插件其实就是dll。这一类插件的开发需要依赖adobe发布的sdk，可以在官网下载，sdk里头也提供了许多的sample，可以提供参考。
   a. adobe ps cc 2017 sdk地址
2. 该种插件需要手动放到PS安装目录的plug-Ins目录下，启动PS之后就会被自动加载。
3. 该插件无法实现界面，需要单独的面板来提供支持。可以在帮助-系统信息-可选的和第三方增益工具这里看到成功读取到的插件。

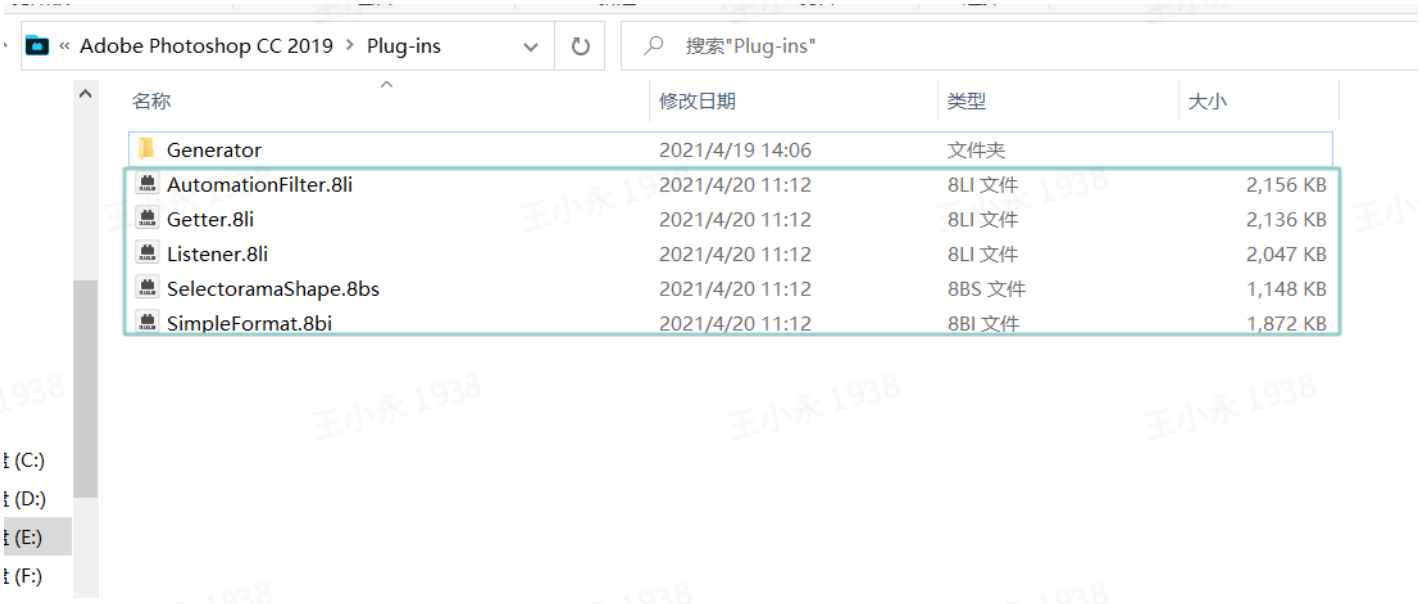
用SDK开发插件

插件的格式

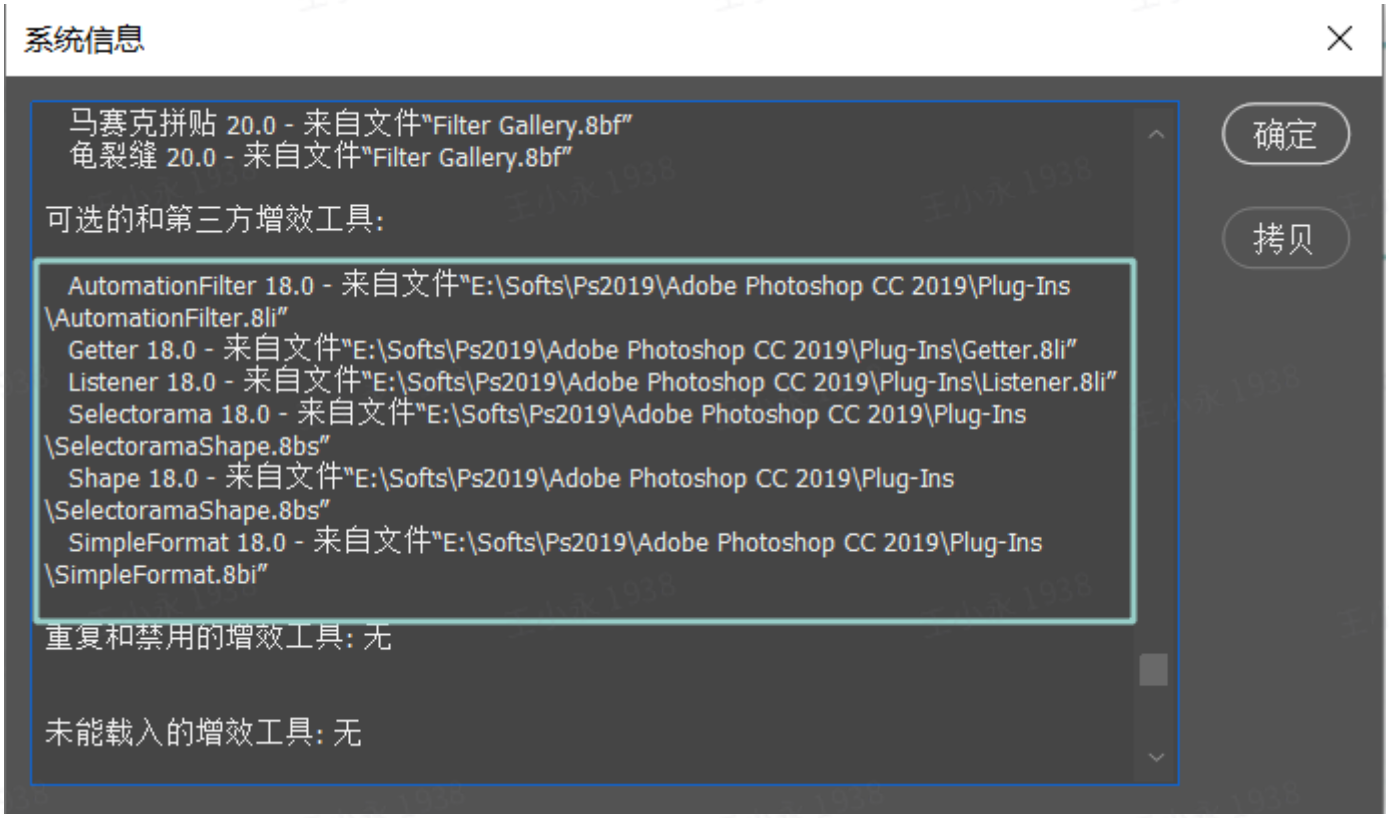
8li 8ly	Automation
8ba	Import
8bc	Color picker
8be	Export
8bf	Filter
8bi	File format
8bp	General
8bs	Selection*
8bx	Extension
8by	Parser*

插件的目录

> 这种自己编出来的插件需要要放在ps安装的目录下对应的plug-lns目录下，这样ps启动的时候，它就可以被读到。比如我的PS安装在了E盘，那么我们编出来的xxx.8li,xxx.8bx插件，就要放在E:\Softs\Ps2019\Adobe Photoshop CC 2019\Plug-ins 这个路径。



在ps中的体现：

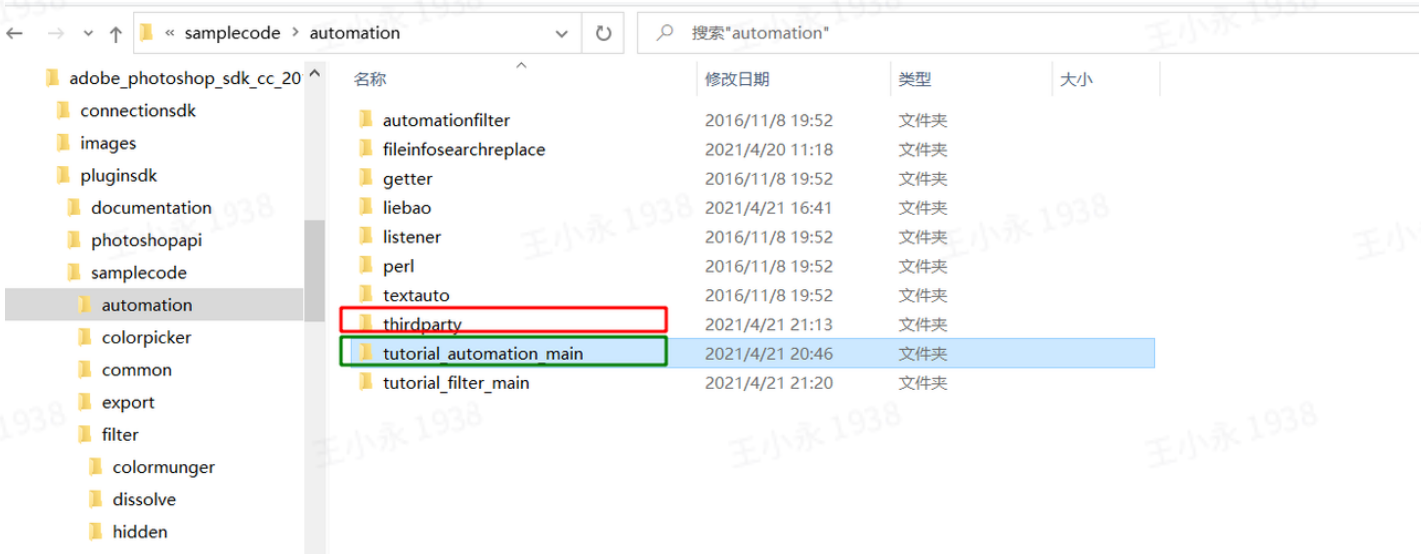


8li插件的实现

> 将工程创建到了sdk里面的samplecode目录下

thirdpart

> thirdpart放到这个路径是为了不改tutorial_automation_main里面对应的批处理的路径



tutorial_automation_globals.h

1. TUTORIAL_AUTOMATION_PLUGINNAME：插件的名字
2. TUTORIAL_AUTOMATION_UUID：在PS调用插件相关接口时，会用到这个uuid来标识插件，生成对应的接口，例如：Play267c8093-d35c-4fb7-b0ae-7b224c7fc1ce(/*...*/)
3. EXPORT_LAYERS_CSXS_EVENT_ID：定义的事件ID，从ecp那里发出，在cpp这里监听
4. DONE_CSXS_EVENT_ID：定义的事件ID，当需要从dll这里往cep那里通信时，需要用到的。

HTTP

```
#define TUTORIAL_AUTOMATION_PLUGINNAME "test_auto"
#define TUTORIAL_AUTOMATION_PLUGINDESC "An example automation plug-in for Adobe Photoshop."
#define TUTORIAL_AUTOMATION_UUID "267c8093-d35c-4fb7-b0ae-7b224c7fc1ce"
#define TUTORIAL_AUTOMATION_RESOURCE_ID 18601
#define TUTORIAL_AUTOMATION_SUITE_ID 'exam'
#define TUTORIAL_AUTOMATION_CLASS_ID TUTORIAL_AUTOMATION_SUITE_ID
#define TUTORIAL_AUTOMATION_EVENT_ID 'ExAm'
#define TUTORIAL_AUTOMATION_VENDORNAME "memyselfandi"

#define EXPORT_LAYERS_CSXS_EVENT_ID "liebao.browser.aet.exportlayersevent"
#define DONE_CSXS_EVENT_ID "liebao.browser.aet.doneevent"
```

tutorial_automation_main.cpp

> 部分内容如下。

1. 关于这个入口函数，不同的插件对应的入口函数是有区别的，这里是automation插件，它的入口函数是AutoPluginMain。在PS里面对tutorial_automation.8li插件的相关操作，都会进入到这里来
2. EXPORT_LAYERS_CSXS_EVENT_ID 就是它监听的事件ID，监听到这个事件了之后，会调用CSXSEventExportLayersCB这个函数
3. 我们在cep面板上做了相关操作之后，走到js这一层，而js不能直接处理PS的相关接口，所以又通过CSInterface()传到jsx这一层，到了jsx这里后，它就是直接面向宿主程序ps的，我们的这个 EXPORT_LAYERS_CSXS_EVENT_ID 事件，就是在jsx这里dispatch的。dll里监听到 EXPORT_LAYERS_CSXS_EVENT_ID 事件后，就会调用与之绑定的接口

C++

```
//...

void CSXSEventExportLayersCB(const csxs::event::Event *const event, void *const context)
{
```

```
    BOOL status = EnumWindows(getPSMainWindowCB, (LPARAM)NULL);
    assert(status != 0);
    MessageBoxA(globalPSMainWindowHwnd, "lb_browser", "From 8li",
MB_OK|MB_ICONINFORMATION);

    // Send a message back to the CEP Panel.
    csxs::event::Event pluginDoneEvent;
    pluginDoneEvent.type = DONE_CSXS_EVENT_ID;
    pluginDoneEvent.scope = csxs::event::kEventScope_Application;
    pluginDoneEvent.appId = CSXS_PHOTOSHOP_APPID;
    pluginDoneEvent.extensionId = TUTORIAL_AUTOMATION_PLUGINNAME;
    pluginDoneEvent.data = "Hello back from C++!";
    globalSDKPlugPlug->DispatchEvent(&pluginDoneEvent);

    return;
}

DLLExport SPAPI SPErr AutoPluginMain(const char* caller,
                                     const char* selector,
                                     void* message)
{
    SPErr status = kSPNoError;

    SPMessageData *basicMessage = (SPMessageData *)message;

    sSPBasic = basicMessage->basic;

    if (sSPBasic->IsEqual(caller, kSPInterfaceCaller)) {
        //...
    }
    else if (sSPBasic->IsEqual(caller, kPSPhotoshopCaller)) {
        if (sSPBasic->IsEqual(selector, kPSDoIt)) {
            PSActionsPlugInMessage *tmpMsg = (PSActionsPlugInMessage *)message;
            // BOOL status = EnumWindows(getPSMainWindowCB, (LPARAM)NULL);
            // assert(status != 0);
            if (!globalSDKPlugPlug) {
                globalSDKPlugPlug = new SDKPlugPlug;
                status = globalSDKPlugPlug->Load();
                if (status != kSPNoError) {
                    delete globalSDKPlugPlug;
                    return status;
                }
            }

            if (!globalEventHandler) {
```

```
48              if (!globalEventListenerRegistered) {
49                  csxs::event::EventErrorCode csxsStat;
50                  csxsStat = globalSDKPlugPlug-
    >AddEventListener(EXPORT_LAYERS_CSXS_EVENT_ID,
51
    CSXSEventExportLayersCB,
52                                                                  NULL);
53                  if (csxsStat !=
    csxs::event::EventErrorCode::kEventErrorCode_Success) {
54                      return kSPLogicError;
55                  }
56
57                  globalEventListenerRegistered = true;
58              }
59          }
60
61
62      }
63
64      return status;
65  }
```

tutorial_automation_pipl.r

> 这里是PIPL文件。

1. Kind决定了本插件会显示到哪个菜单项下面，例如如果是Filter，就是在滤镜下方。本例会出现在文件-自动这个地方。
2. Category是菜单项的子项item名，下面还会介绍如何在PS UI中隐藏一个插件的方法，就是改这一项的值。

```
///@ingroup·PiPLGeneralKeys
/**·Type·or·kind·of·plug-in;·key·value·is·'kind'.·
*·This·property·key·reflects·the·\c·Kind·property·in·the·PiPL·resource·file.
*·The·property·data·has·type·@ref·PIType,·with·expected·values·as·follows:
*·-·'ARPI'=Adobe·Illustrator
*·-·'SPEA'=Adobe·Suite·Pea
*·-·'8BXM'=Accelerator·extension
*·-·'8BAM'=Import·module
*·-·'8BEM'=Export·module
*·-·'8BFM'=Filter·module
*·-·'8BIF'=Format·module
*·-·'8BSM'=Selection·module
*·'-·8BYM'=Parser·module
*/
#define·PIKindProperty→ → → 0x6b696e64L
```

R

```
//....

resource 'PiPL' ( TUTORIAL_AUTOMATION_RESOURCE_ID, TUTORIAL_AUTOMATION_PLUGINNAME, purgeable)
    {
        {
        Kind { Actions },
        Name { TUTORIAL_AUTOMATION_PLUGINNAME },
        Category { "AdobeSDK" },
        Version { (latestActionsPlugInVersion << 16) | latestActionsPlugInSubVersion },

        Component { ComponentNumber, TUTORIAL_AUTOMATION_PLUGINNAME },

        #ifdef __PIMac__
            CodeMacIntel64 { "AutoPluginMain" },
        #else
            #if defined(_WIN64)
                CodeWin64X86 { "AutoPluginMain" },
            #else
                CodeWin32X86 { "AutoPluginMain" },
            #endif
        #endif

        EnableInfo { "true" },

        HasTerminology
            {
            TUTORIAL_AUTOMATION_CLASS_ID,
            TUTORIAL_AUTOMATION_EVENT_ID,
            TUTORIAL_AUTOMATION_RESOURCE_ID,
            TUTORIAL_AUTOMATION_UUID
            },

        Persistent{},

        // Only relevant if Persistent is set.
        Messages
        {
            startupRequired,
            doesNotPurgeCache,
            shutdownRequired,
            acceptProperty
```

```
        },
        }
    };

//...
```

在ps ui中隐藏某个插件

> 修改PIPL文件。

```
resource 'PiPL' ( TUTORIAL_FILTER_RESOURCE_ID, TUTORIAL_FILTER_PLUGINNAME, purgeable )
{
    {
        Kind { Filter },
        Name { TUTORIAL_FILTER_PLUGINNAME },
        Category { "**Hidden**"},
        Version { (latestFilterVersion << 16) | latestFilterSubVersion },

        Component { ComponentNumber, TUTORIAL_FILTER_PLUGINNAME },

        #ifdef   PIMac
```

关于build.bat

1. 如PhotoshopSDKRoot，CnvtPiPLExePath，ZXPSignCmdExe，ZXPCert，ZXPCertPassword等各项需要按自己的相关开发环境重设
2. 完整的代码以及这里需要用到或可能用到的工具都会放在压缩包里

SQL

```
@echo off
setlocal

echo Build script started executing at %time% ...
set BuildType=%1
if "%BuildType%"=="" (set BuildType=release)

set PhotoshopSDKRoot=C:\Users\Kingsoft\Desktop\adobe_photoshop_sdk_cc_2017_win\pluginsdk
set CnvtPiPLExePath="%PhotoshopSDKRoot%\samplecode\resources\Cnvtpipl.exe"
set PhotoshopPluginsDeployPath=%~dp0deploy

set ZXPSignCmdExe=C:\Users\Kingsoft\Desktop\ZXPSignCmd.exe
set ZXPCert=C:\Users\Kingsoft\Desktop\automation.p12
set ZXPCertPassword=password123456

call "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat" x64

set BuildDir=%~dp0msbuild
if not exist %BuildDir% mkdir %BuildDir%
pushd %BuildDir%

set ProjectName=tutorial_automation

set EntryPoint=%~dp0src\%ProjectName%_main.cpp
set ResourcePiPL=%~dp0src\%ProjectName%_pipl.r
set ResourceRC=%BuildDir%\%ProjectName%_pipl.rc
set ResourceRES=%BuildDir%\%ProjectName%_pipl.res

set ThirdPartyDirPath=%~dp0..\thirdparty

set OutBin=%BuildDir%\%ProjectName%.8li
```

在CEP面板里触发DLL的弹窗

> 在html定义了一个liebao browser按钮，点击之后调到liebao.js

```
<input type="button" value="liebao browser" onclick="ExportLayersCB()">
```

```
function ExportLayersCB() {
    var cs = new CSInterface();
    cs.evalScript("ESPSExportLayers();");
    return;
}
```

> liebao.js里面又调了ExtendScript里面定义的ESPSExportLayers
>
> ESPSExportLayers里面发出了一个事件

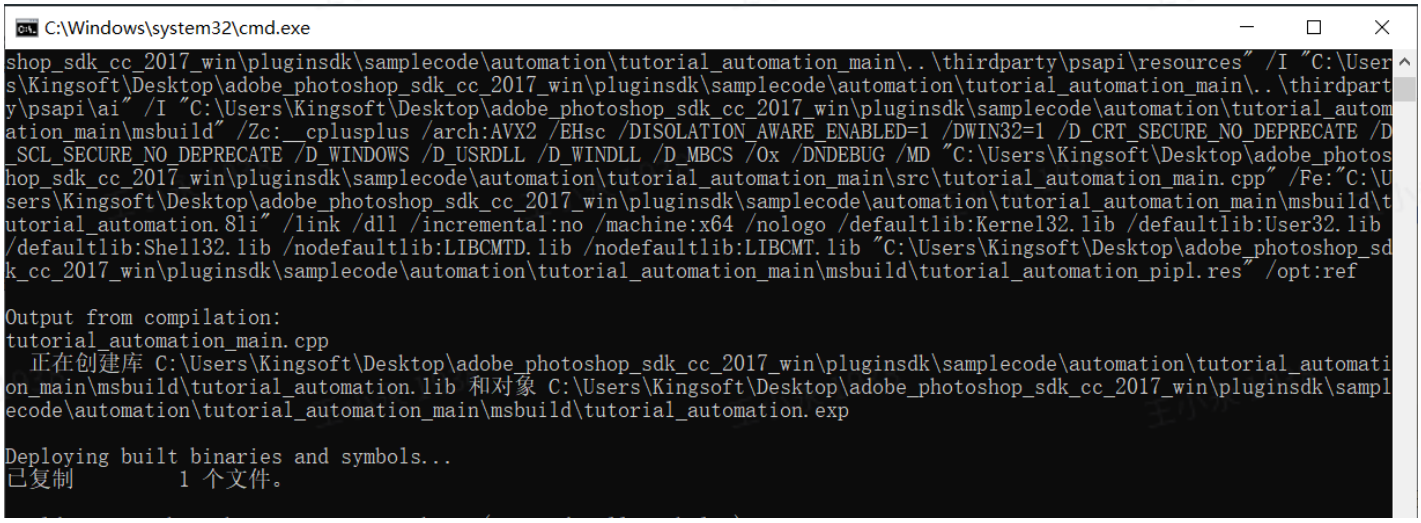
```
function ESPSExportLayers() {
    // The library might not exist in some Photoshop versions.
    try {
        var xLib = new ExternalObject("lib:\PlugPlugExternalObject");
    } catch (e) {
        alert(e);
        return;
    }
    if (xLib) {
        var eventObj = new CSXSEvent();
        eventObj.type = EXPORT_LAYERS_CSXS_EVENT_ID;
        eventObj.data = true;
        eventObj.dispatch();
        return;
    }
    return;
}
```

> 这个事件，在cpp文件里被监听到之后，就会调用与之绑定的接口

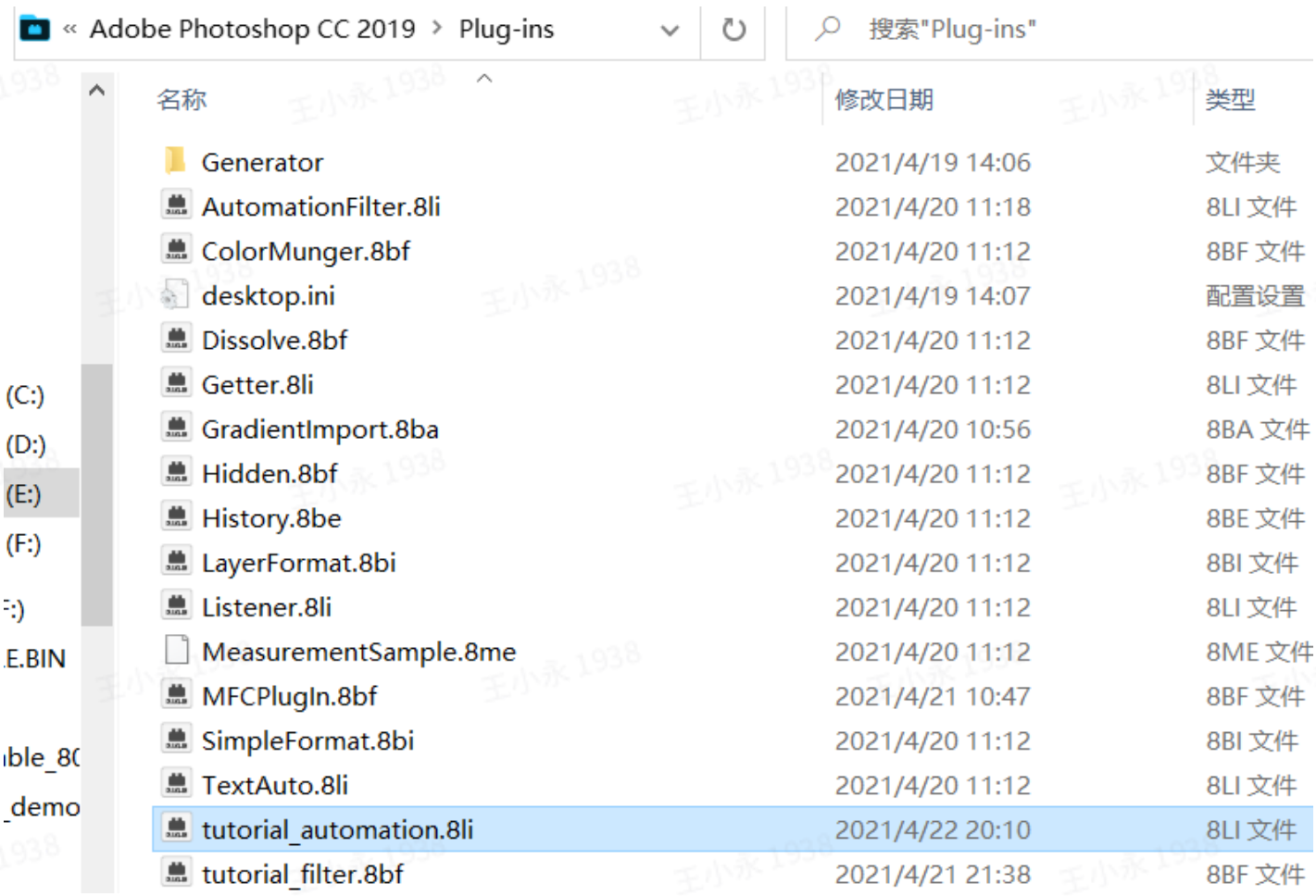
```
csxs::event::EventErrorCode csxsStat;
MessageBoxA(globalPSMainWindowHwnd, "before CSXSEventExportLayersCB", "liebao1", MB_OK|MB_ICONINFORMATION);
csxsStat = globalSDKPlugPlug->AddEventListener(EXPORT_LAYERS_CSXS_EVENT_ID,
                                               CSXSEventExportLayersCB,
                                               NULL);
MessageBoxA(globalPSMainWindowHwnd, "after CSXSEventExportLayersCB", "liebao1", MB_OK|MB_ICONINFORMATION);
if (csxsStat != csxs::event::EventErrorCode::kEventErrorCode_Success) {
    return kSPLogicError;
}
```

弹窗效果与触发

> 点击build.bat生成插件。

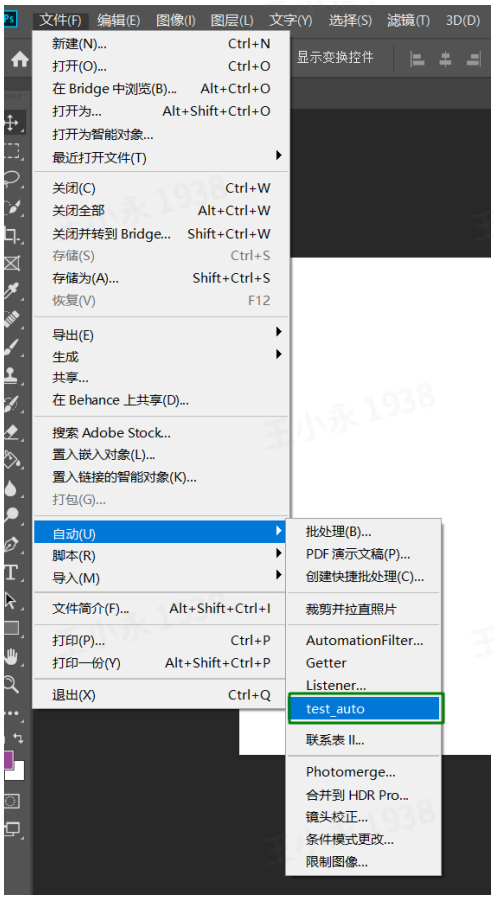


> 复制到安装PS的路径下，如：
>
> E:\Softs\Ps2019\Adobe Photoshop CC 2019\Plug-ins

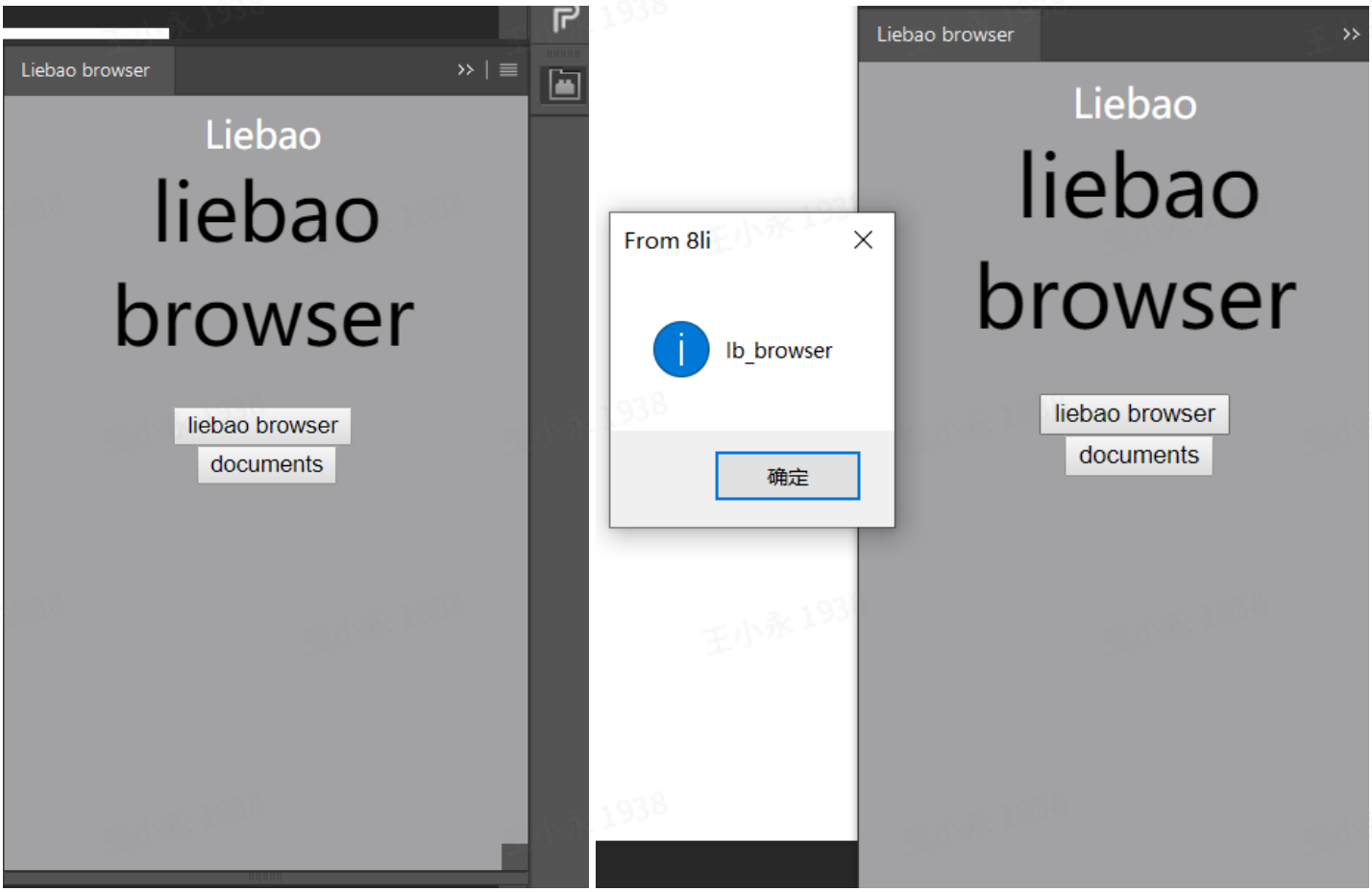


> 进入PS，创建一个文档（问题1：为什么要创建一个文件？）。

> 点击test_auto启动插件（问题2：为什么要手动点击启动？）。由于我把测试消息框去掉了，所以点击之后不会弹出来提示框。



启动liebao browser ecp插件，并点击liebao browser按钮。弹出了dll的提示对话框。



问题

问题1:为什么要创建一个文件?

> 因为没有打开的文件的话，插件是处于不可点击的灰色状态。

问题2：为什么要手动点击启动?

> 因为这种插件是动态加载，触发的时候才去加载。（如果是随宿主程序启动加载，那假如我的插件目录下有很多插件，那它刚开始的时候就要加载很多东西，而我还没做什么事情呢）

相关方案

1. 能否在点击cep插件上的按钮或其他东西的时候，就加载这个插件，省去手动启动那一步的动作。

> 在jsx这一层的时候，可以通过CSInterface.js有直接面向PS相关接口的能力了。也就是说，应该可以在jsx里面，发出事件之前，用代码去做这件事情。

```
function ESPSExportLayers() {
    if (app.documents.length !== 0) {

    }

    //app.doAction();runMenuItem

    try {
        var xLib = new ExternalObject("lib:\PlugPlugExternalObject");
    } catch (e) {
        alert(e);
        return;
    }
    if (xLib) {
        var eventObj = new CSXSEvent();
        eventObj.type = EXPORT_LAYERS_CSXS_EVENT_ID;
        eventObj.data = true;
        eventObj.dispatch();
        return;
    }
    return;
}
```

> 红框这里是，判断了PS当前有没有打开的文件。

> 如果没有打开文件的话，根据PS对这个插件隐藏的逻辑，就不能去加载它的功能。
>
> 当有打开的文件的时候，就走到这个判断里面去了，这时候，如果用相关接口或调用脚本或其他方式，模拟类似点击了**文件-自动-test_auto**这样一个步骤，应该问题就解决了。