2020/07/22_Windows编程_第4课_注册表的使用_进程的创建

笔记本:	Windows编程
创建时间:	2020/7/22 星期三 10:35
作者:	ileemi
标签:	进程的创建, 注册表的使用

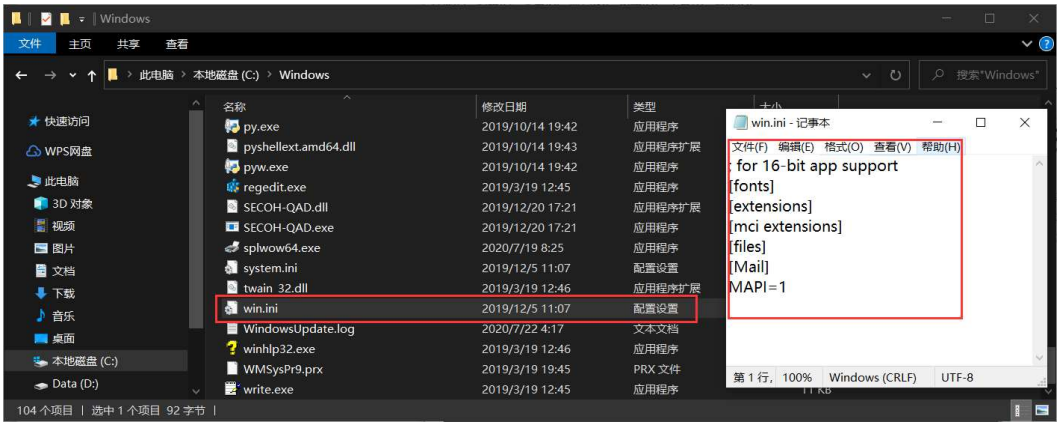
回顾

> 配置文件.ini 存在缺陷， 当使用.ini进行存储软件的配置信息的软件卸载后，对应的配置文件也一并删除，当该软件重新安装后，原来的配置信息也就没有了，相关的配置需要重新设置。(可以将配置文件写到系统目录)

注册表

将配置文件存放到系统目录的.ini文件中，配置信息称为共有的，且.ini文件的存储类型较为单一，只能存储一些简单的数据类型。且结点名容易重复，一旦重复就会将原

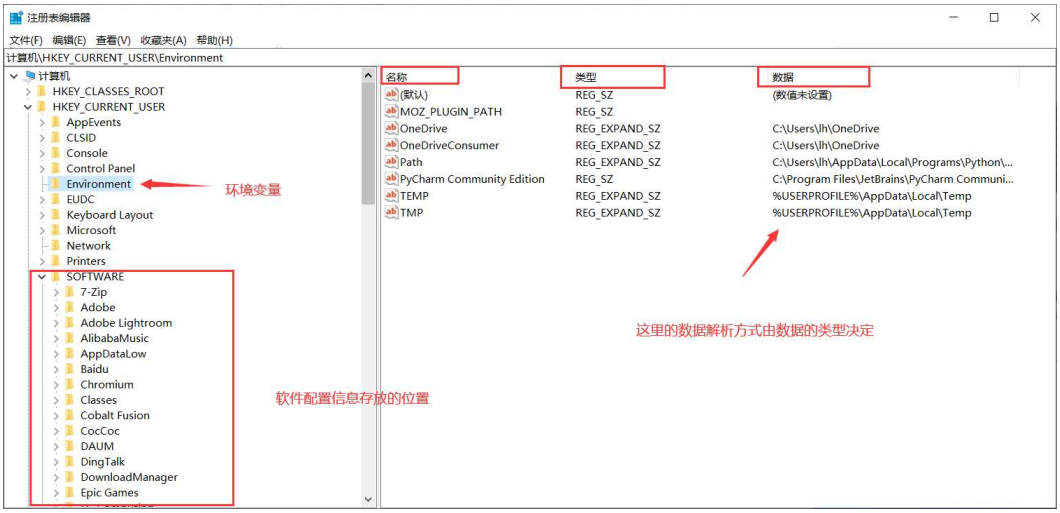
来软件的配置信息给擦除，导致其特软件的配置信息不能正常读取。

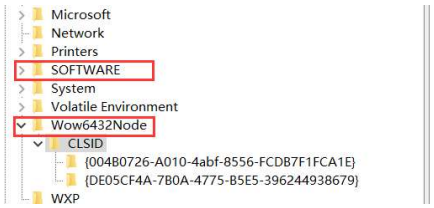


> 根据 .ini 文件的局限性，我们就需要一种全局的，保存数据类型较多的一种存储方式，Winodws 提供了注册表（软件卸载后，相关的配置中信息不会被删除）。

注册表（全局的）：
不但保存了软件的配置信息，还保存了操作系统所有的设置信息。例如可以将软件的路径写入到注册表的指定位置，就可以使该软件开即自动运行。

打开注册表：
Win + R --> **regedit**，整体构造以目录为结构。以路径为单位。



> Wow6432Node:
> 当软件为64位时配置信息会保存到 SoftWare 下，当为32位的时候其配置会被重定位保存到 Wow6432Node 这个文件夹中。
>
> 
>

为了安全性，在 Windows10中写入注册表，软件需要以管理员身份运行进行写入。

注册表的使用

相关API：**RegCreateKeyEx**

Registry Functions

The following functions are used with the registry.

Function	Description
RegCloseKey	Releases a handle to the specified registry key.
RegConnectRegistry	Establishes a connection to a predefined registry handle on another computer.
RegCreateKeyEx	Creates the specified registry key.
RegDeleteKey	Deletes a subkey.
RegDeleteValue	Removes a named value from the specified registry key.
RegDisablePredefinedCache	Disables the predefined registry handle table of **HKEY_CURRENT_USER** for the specified process.
RegEnumKeyEx	Enumerates subkeys of the specified open registry key.
RegEnumValue	Enumerates the values for the specified open registry key.
RegFlushKey	Writes all the attributes of the specified open registry key into the registry.
RegGetKeySecurity	Retrieves a copy of the security descriptor protecting the specified open registry key.
RegLoadKey	Creates a subkey under **HKEY_USERS** or **HKEY_LOCAL_MACHINE** and stores registration information from a specified file into that subkey.
RegNotifyChangeKeyValue	Notifies the caller about changes to the attributes or contents of a specified registry key.
RegOpenCurrentUser	Retrieves a handle to the **HKEY_CURRENT_USER** key for the user the current thread is impersonating.
RegOpenKeyEx	Opens the specified registry key.
RegOpenUserClassesRoot	Retrieves a handle to the **HKEY_CLASSES_ROOT** key for the specified user.
RegOverridePredefKey	Maps a predefined registry key to a specified registry key.
RegQueryInfoKey	Retrieves information about the specified registry key.
RegQueryMultipleValues	Retrieves the type and data for a list of value names associated with an open registry key.
RegQueryValueEx	Retrieves the type and data for a specified value name associated with an open registry key.

添加注册表

> API：
> RegCreateKey -- 创建指定的注册表项
> RegCreateKeyEx（扩展版本）

RegCreateKey 定义：
HKEY hkey;

```
LONG RegCreateKey(
HKEY hKey,   // 添加的注册表存档的目录
LPCTSTR lpSubKey,    // 添加的注册表子项的名称 -- 子键
PHKEY phkResult // &hkey
);
```

为了以后方便操作添加的注册表目录，会返回一个HKEY的句柄，**返回的句柄就代表这个目录。**
参数3：指向一个变量的指针，该变量接收打开或创建的键的句柄。当不再需要返回的句柄时，调用RegCloseKey函数来关闭它。

> RegCloseKey(要关闭的句柄); -- 为了防止句柄泄漏，使用完需要使用 RegCloseKey 来进行关闭。

> 注册表值可以存储各种格式的数据。当您在注册表值下存储数据时，例如通过调用RegSetValueEx函数，您可以指定以下值之一来指示所存储的数据类型。在检索注册表值时，RegQueryValueEx等函数使用这些值指示检索的数据类型。

使用goto时，goto语句中尽量不要定义变量，定义的变量都应放在goto之外。

代码示例：

```
// 添加注册表
void CRegeditDlg::OnBnClickedAddreg()
{
    // 添加注册表子项到注册表中去
    HKEY hkey1 = NULL;
    HKEY hkey2 = NULL;
    LONG result;
    DWORD dwValue = 999;
    const char* strPenStyle = "实心";

    // 在指定的目录下添加指定的注册表项
    result = RegCreateKey(HKEY_CURRENT_USER,
    _T("SOFTWARE\\RegeditTest\\Pen"),
    &hkey1);
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    // 在子项中添加内容，或者在子目录在添加一个子项，在该子项中添加配置
    result = RegSetValueEx(hkey1,
        "", 0, REG_DWORD,
        (BYTE*)&dwValue,
        sizeof(dwValue));
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    result = RegSetValueEx(hkey1,
    "PenStyle", 0, REG_SZ,
    (BYTE*)strPenStyle, sizeof(strPenStyle));
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    // 再创建一个目录
    result = RegCreateKey(HKEY_CURRENT_USER,
    _T("SOFTWARE\\RegeditTest\\Brush"),
```

```
    &hkey2);
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    result = RegSetValueEx(hkey2,
    "BrushStyle", 0,
    REG_SZ, (BYTE*)"波浪", 4);
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    AfxMessageBox("注册表子项写入成功");
SAFE_EXIT:
    if (hkey1 != NULL)
    {
        RegCloseKey(hkey1); // 关闭句柄
    }
    if (hkey2 != NULL)
    {
        RegCloseKey(hkey2); // 关闭句柄
    }
}
```

读取注册表

1. 打开指定的注册表项（查询的是根目录的话不需要打开）
2. 查询打开的注册表项关联的指定值名称的类型和数据

API：**RegOpenKeyEx**，**RegQueryValueEx**

代码示例：

```
// 读取注册表
void CRegeditDlg::OnBnClickedRedreg()
{
    // 打开指定的注册表项
    HKEY hkey = NULL;
    LONG result;
    DWORD dwDataType; // 存储读取数据的类型
```

```
    char szBuff[256] = { 0 };
    DWORD cbData = sizeof(szBuff); // 存储数据所占用的字节数

    // 打开指定注册表中的项
    result = RegOpenKeyEx(HKEY_CURRENT_USER,
    "SOFTWARE\\RegeditTest\\Pen",
    0, KEY_ALL_ACCESS, &hkey);
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败，显示错误信息
        ShowErrorMsg();
        goto SAFE_EXIT;
    }

    // 查询打开的注册表项关联的指定值名称的类型和数据
    result = RegQueryValueEx(hkey,
    "PenStyle", NULL, &dwDataType,
    (BYTE*)szBuff, &cbData);
    if (result != ERROR_SUCCESS)
    {
        // 添加注册表失败
        ShowErrorMsg(); // 显示错误信息
        goto SAFE_EXIT;
    }
    // 判断所查询项对应的数据类型，将其值进行弹出
    switch (dwDataType)
    {
        case REG_DWORD:
        {
            CString csFmt;
            csFmt.Format("%d", *(DWORD*)szBuff);
            AfxMessageBox(csFmt);
            break;
        }
        case REG_SZ:
        {
            AfxMessageBox(szBuff);
            break;
        }
    }
SAFE_EXIT:
    if (hkey != NULL)
    {
        RegCloseKey(hkey);
    }
}
```

删除注册表

> 删除的项，需要其没有子目录，删除时，需要先将其子项进项删除，才能删除当前项（防止误删）。
> API: RegDeleteKey -- 删除子键

代码示例：

```
// 删除注册表中指定的子项
void CRegeditDlg::OnBnClickedDelreg()
{
    LONG result;
    // 删除子项前，需要优先删除子项的子项
    RegDeleteKey(HKEY_CURRENT_USER, "SOFTWARE\\RegeditTest\\Pen");
    RegDeleteKey(HKEY_CURRENT_USER, "SOFTWARE\\RegeditTest\\Brush");
    // 删除子项
    result = RegDeleteKey(HKEY_CURRENT_USER, "SOFTWARE\\RegeditTest");
    if (result != ERROR_SUCCESS)
    {
        AfxMessageBox("删除子项失败");
        return;
    }
    AfxMessageBox("删除子项成功");
}
```

```
300  //删除注册表中指定的子项
301  void CRegeditDlg::OnBnClickedDelreg()
302  {
303      LONG result;
304      //删除子项前，需要优先删除子项的子项
305      RegDeleteKey(HKEY_CURRENT_USER,"SOFTWARE\\RegeditTest\\Pen");
306      RegDeleteKey(HKEY_CURRENT_USER,"SOFTWARE\\RegeditTest\\Brush");
307      //删除子项
308      result = RegDeleteKey(HKEY_CURRENT_USER,"SOFTWARE\\RegeditTest");
309      if (result != ERROR_SUCCESS)
310      {
311          AfxMessageBox("删除子项失败");
312          return;
313      }
314      AfxMessageBox("删除子项成功");
315  }
```

遍历注册表项

> RegEnumKeyEx -- 枚举指定打开的注册表项的子键。该函数在每次被调用时检索关于一个子键的信息。
> RegEnumValue -- 枚举指定打开的注册表项的值。每次调用该函数时，该函数为键复制一个索引值名称和数据块。

这里使用 `RegEnumKeyEx` 枚举指定打开的注册表项的子键。枚举前，由于不知道子键中有多少子项，所以需要使用 API函数 -- `RegQueryInfoKey` 检索关于指定注册表项的

信息。

检索错误信息：OutputDebugString
在 MFC 中 可以使用 TRACE 宏和DebugView。

代码示例：

```
// 遍历注册表的子键
void CRegeditDlg::OnBnClickedErgodicreg()
{
    DWORD dwSubKeys; // 保存检索的注册表子键的子项项数
    char szBuff[0x1000] = { 0 }; // 保存子项的名称
    DWORD dwlpcNameData = sizeof(szBuff); // 存储子键所占的字节大小
    LONG result;
    RegQueryInfoKey(HKEY_CURRENT_USER,
        NULL, NULL, NULL,
        &dwSubKeys, NULL,
        NULL, NULL, NULL,
        NULL, NULL, NULL);

    for (size_t i = 0; i < dwSubKeys; i++)
    {
        dwlpcNameData = sizeof(szBuff);
        result = RegEnumKeyEx(
            HKEY_CURRENT_USER,
            i, szBuff, &dwlpcNameData,
            NULL, NULL, NULL, NULL);
        if (result != ERROR_SUCCESS)
        {
            ShowErrorMsg();
        }
         ("[51asm] %d %s", i + 1, szBuff);
    }
}
```

遍历注册表的子键中的数值

> API：RegOpenKey、RegQueryInfoKey、RegEnumValue

代码示例：

```
// 遍历注册表的子键中的数值
void CRegeditDlg::OnBnClickedErgodicregvalue()
{
    HKEY hkey;
    LONG result;
    DWORD dwlpcValues; // 保存查询子键中值的数量
```

```
char szValueName[MAX_PATH]; // 保存值的名称
char szDataBuff[0x1000]; // 保存数据

DWORD wdlpValueName;
DWORD wdData;
DWORD dwDataType; // 数据的类型
// 遍历指定的子键中的值
RegOpenKey(HKEY_CURRENT_USER, "Environment", &hkey);

// RegQueryInfoKey 查询子键中值的数量
RegQueryInfoKey(hkey, NULL,
    NULL, NULL,
    NULL, NULL,
    NULL, &dwlpcValues,
    NULL, NULL,
    NULL, NULL);

for (size_t i = 0; i < dwlpcValues; i++)
{
    wdlpValueName = sizeof(szValueName);
    wdData = sizeof(szDataBuff);

    result = RegEnumValue(hkey, i,
        szValueName, &wdlpValueName,
        NULL, &dwDataType,
        (BYTE*)szDataBuff, &wdData);
    if (result != ERROR_SUCCESS)
    {
        ShowErrorMsg();
        goto SAFE_EXIT;
    }
    switch (dwDataType)
    {
        case REG_DWORD:
        {
            TRACE("[51asm] %d %s %d", i + 1,
                szValueName,
                *(DWORD*)szDataBuff);
            break;
        }
        case REG_EXPAND_SZ:
        case REG_SZ:
        {
            TRACE("[51asm] %d %s %s", i + 1,
                szValueName,
                szDataBuff);
```

```
                break;
            }
        }
    }

SAFE_EXIT:
    if (hkey != NULL)
    {
        RegCloseKey(hkey);
    }
}
```

树控件添加图标

CTreeCtrl CListCtrl
SetImageList

CImageList image;
image.Add(LoadIcon(IDxxx));
CTreeCtrl tree;
tree.SetImageList(&image);
Tree.InsertItem(xxx, xxx, 0, xx);

进程(Process)

进程和和程序的区别

程序：一堆二进制代码
进程（分配资源 -- 内存，文件，窗口...）：程序运行起来就就叫进程，一个程序运行两次，就有两个进程。

操作系统管理这个应用程序的资源分配问题。

进程：很多东西的抽象，操作系统会用一个结构体来表达进程相关的数据（操作系统不会将这个结构体告诉我们），操作系统根据**进程句柄**（**数组下标**）就可以找到进行相关的资源数据。操作系统保存进程相关的信息是用数组存储的。

为了使用进程方便，操作系统为进程分配一个ID号（进程ID）,一般操作进程的时候，首先获取进程ID，通过ID在去获取进程的句柄。

调度：单核CPU，一个CPU每次只运行一份代码，或者每次只运行一个程序，当需要运行两份代码或者两个程序的时候，CPU不能做到同时运行，当运行这个程序的时候，CPU就从运行的程序中到需要运行的程序中去，来回切换。

时间片：一个程序运行多长的时间

调度

进程的创建

一般双击运行的程序，对应的进程都是由操作系统创建的，同时自己通过API也可以实现自己创建对应的进程。

> 创建进程的API
> **CreateProcess** -- 具有交互性
> **ShellExecuteA** -- 没有交互性
> **WinExec** -- 没有交互性，这个进程给你没有关系
>
> MSDN 官方文档

WinExec

可以打开系统内置的程序，或者打开绝对路径中的可执行程序

> WinExec("calc", SW_SHOWNORMAL);

ShellExecuteA

可以打开指定的磁盘目录，支持系统内置的程序对指定目录中的文件等进行编辑操作。

```
// 打开系统内置的可执行程序
ShellExecute(NULL, "open", "calc", NULL, NULL, SW_SHOWNORMAL);
// 打开系统指定的磁盘目录
ShellExecute(NULL, "explore", "D:\\", NULL, NULL, SW_SHOWNORMAL);
// 在指定的磁盘中查找数据文件
ShellExecute(NULL, "find", "D:\\", NULL, NULL, 0);
// 使用系统默认程序打开磁盘指定的文件进行编辑
ShellExecute(NULL, "edit", "D:\\Config.ini", NULL, NULL, SW_SHOWNORMAL);
```

CreateProcess

通过窗口句柄打开程序或者进行关闭。

> **ExitProcess** -- 结束进程（在任意函数内）
> **GetCommandLine** -- 获取命令行参数
> **GetEnvironmentStrings** -- 获取环境变量
> **GetCurrentProcess** -- 获取当前进程
> **SetProcessAffinityMask** -- 指定当前进程在哪核CPU运行
> **SetProcessAffinityMask**(**GetCurrentProcess**, 1); -- 指定当前进程在1核

> CPU运行
> **GetWindowThreadProcessId** -- 检索创建指定窗口的线程的标识符，以及创建窗口的进程的标识符(可选)
> **OpenProcess** -- 打开一个现有的进程对象
> **GetCurrentProcess** -- 获取当前进程句柄

创建系统进程，代码示例：

```
#include <iostream>
#include <stdio.h>
#include <Windows.h>

int main()
{
    printf("Hello World\r\n");

    /*
    创建进程的三种方法
    1. WinExce
    2. ShellExecute
    3. CreateProcess
    */

    //WinExec("calc", SW_SHOWNORMAL);

    // ShellExecute 的使用
#if 0
    // 打开系统内置的可执行程序
    ShellExecute(NULL, "open", "calc", NULL, NULL, SW_SHOWNORMAL);
    // 打开系统指定的磁盘目录
    ShellExecute(NULL, "explore", "D:\\", NULL, NULL, SW_SHOWNORMAL);
    // 在指定的磁盘中查找数据文件
    ShellExecute(NULL, "find", "D:\\", NULL, NULL, 0);
    // 打开磁盘指定的文件进行编辑
    ShellExecute(NULL, "edit", "D:\\Config.ini", NULL, NULL,
SW_SHOWNORMAL);
#endif // 0

#if 1
    // 两个结构体为传出参数
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    // 使用 ZeroMemory 初始化结构体成员
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
```

```
    /*
    创建系统进程，参数1路径为空即可
    创建自己的进程，参数1路径填写绝对路径
    */
    if (!CreateProcess(
        NULL, // 可执行模块名称
        (LPSTR)"calc", // 命令行参数
        NULL, // 不继承进程的句柄
        NULL, // 不继承线程的句柄
        FALSE, // 新进程不继承进程句柄
        0, // 默认创建
        NULL, // 使用父环境块
        NULL, // 使用父目录
        &si, // 指向窗口的结构体
        &pi // 指向进程信息的结构体
    ))
    {
        printf("创建进程失败\r\n");
        return 0;
    }

    // 获取进程ID,以及进程句柄
    printf("pid:%d handle=%p\n", pi.dwProcessId, pi.hProcess);

    system("pause");

    // 强制结束进程
    TerminateProcess(pi.hProcess, 0);

    // 关闭进程和线程句柄
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
#endif // 0

    return 0;
}
```

将打开现有的进程强制关闭

> 相关API:
> **FindWindow、GetWindowThreadProcessId、OpenProcess、TerminateProcess、CloseHandle**
> 代码示例:

```cpp
#include <iostream>
#include <stdio.h>
#include <Windows.h>

int main()
{
    //指定当前进程在哪核CPU运行
    //SetProcessAffinityMask(GetCurrentProcess(), 1);

    // 获取打开进程的句柄
    DWORD dwPID; // 存储进程的PID
    HWND hwnd = FindWindow(NULL, "计算器"); // 获取窗口句柄
    GetWindowThreadProcessId(hwnd, &dwPID); // 检索创建指定窗口的线程的标识符

    // 打开一个现有的进程对象，返回值为程序打开的句柄
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPID);

    // 强制结束指定的进程
    TerminateProcess(hProcess, 0);

    // 关闭一个打开的对象句柄
    CloseHandle(hProcess);
    return 0;
}
```