

# Window DPI

## 一、DPI定义

DPI全称是dots per inch，也就是每英寸的点数，在显示器上就是每英寸的像素个数，Window上一般默认是96 dpi 作为100% 的缩放比率，但是要注意的是该值未必是真正的显示器物理值，只是Windows里我们的一个参考标准。

## 二、为什么设置DPI会使我们看到的程序变大

简单来说我们可以直接理解为系统的机制，我们暂时不讨论程序对DPI的适配，只讨论系统在设置125%以后程序会放大，如果打个比方的话就像系统使用一个放大镜一样让我们的程序看起来变大了，但是实际上程序还是那么大，只不过视觉上看起来按照设置的DPI比例放大了而已。

如果讨论程序对DPI的适配，也就是说，程序感知到了桌面的dpi变为125%，那么程序在感知到桌面的dpi变化之后，程序把自身大小调成125%的大小，这个改变是与系统无关的，程序只是感知系统的DPI了而已。

举个例子理解，在Win10系统下我们原生DPI为100%，然后去量一个窗口的大小为100\*100，如果此时我们把DPI设置为200%，我们会发现视觉上窗口变大了两倍，但是用GetWindowRect量出来的窗口大小依旧为100\*100，这就是因为系统使用了**DWM (Desktop Window Manager) 虚拟化机制**，就是刚刚所说的放大镜。但是实际上你窗口本身的大小坐标并没有变化。可以理解为每个单元被放大镜变大了，而不是单元变多了。所以实际工作的逻辑坐标根本就没有变化，因为只是每个坐标单元变大了。

DPI比例与每英寸点数对应关系：

屏幕比例	每英寸点数
100%	96
125%	120
150%	144
175	192

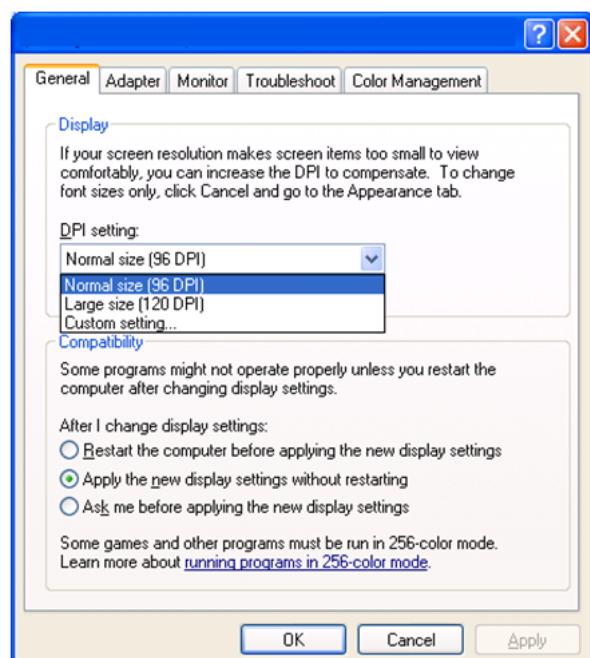
注：以上的每英寸点数为代码中关于DPI问题经常使用到的数字

## 三、不同系统下的DWM机制

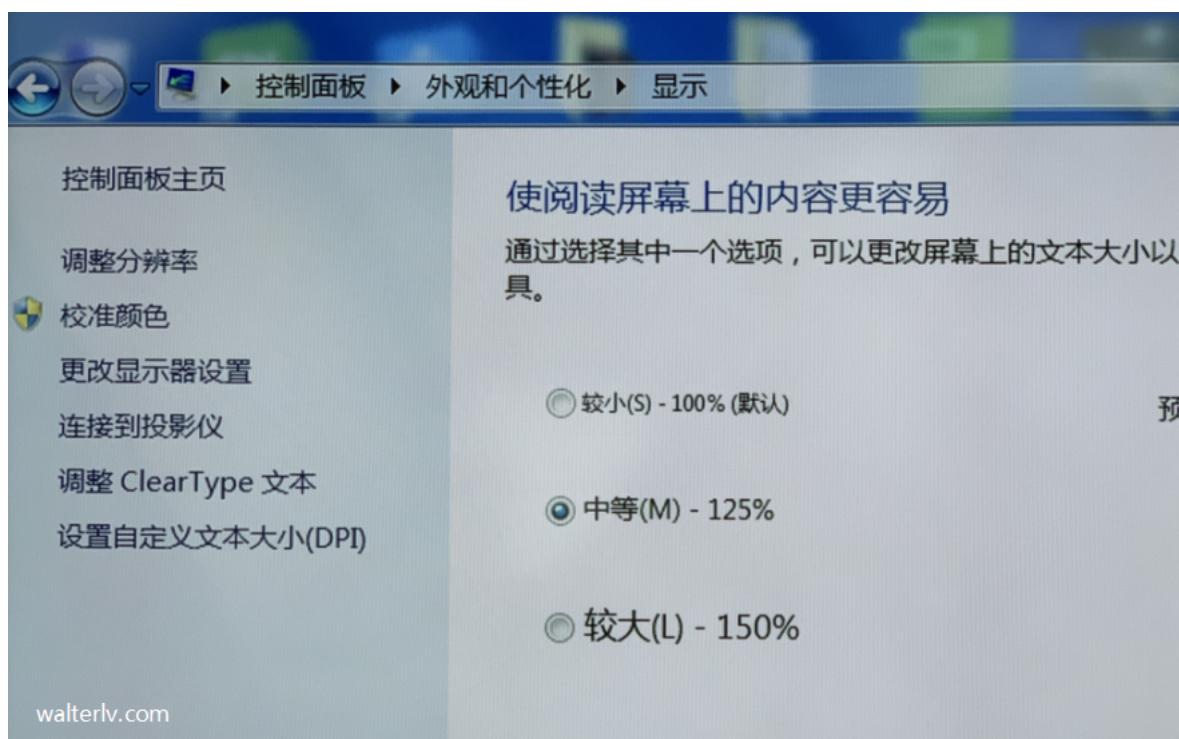
对于系统dpi的改善，可以分为三个时间段

- Windows XP阶段：

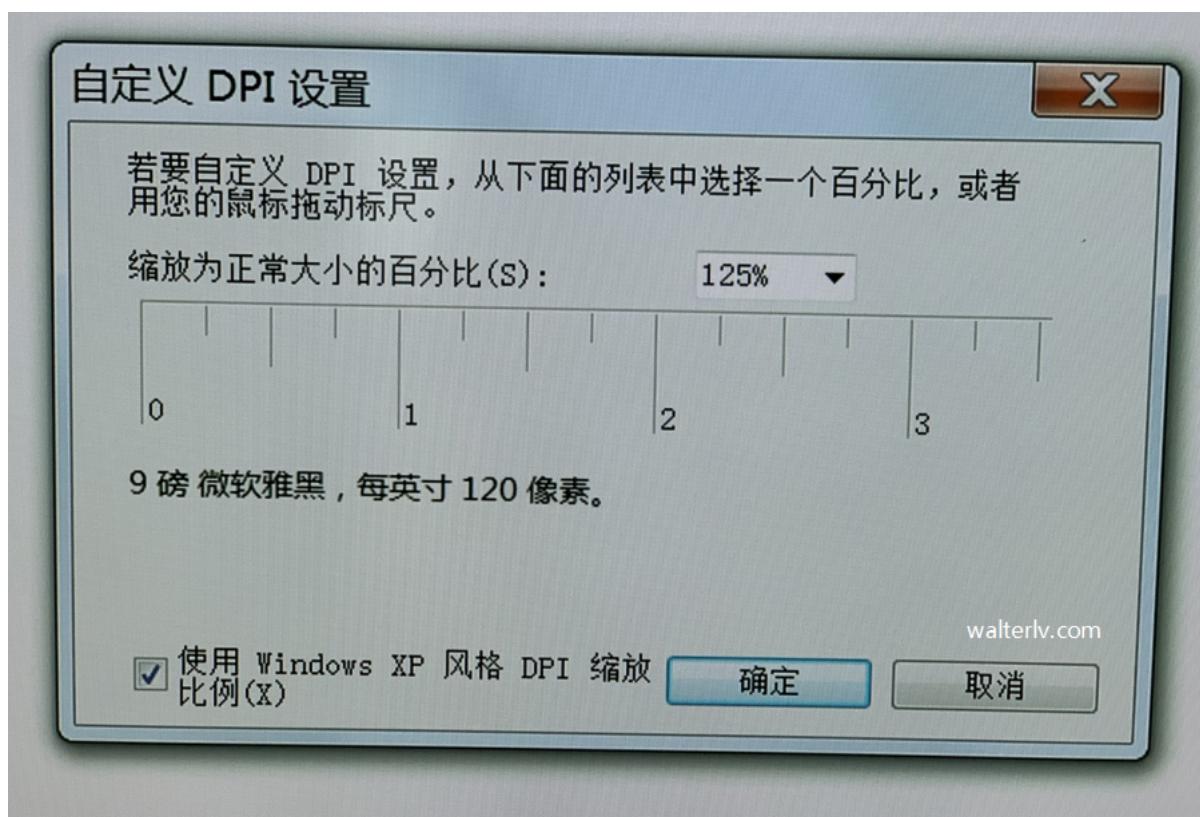
对高DPI的支持比较差劲，大部分情况下就是字体的放大，当然我们程序也可以通过GetDeviceCaps(hDC, LOGPIXELSX)获取DPI后自己对绘画的内容进行缩放。因为现在已经用不到了，不作过多讨论。



- Windows Vista/Windows 7/Windows 8阶段：  
在 Windows Vista / 7 / 8 中，操作系统提供了真正的 DPI 的设置：



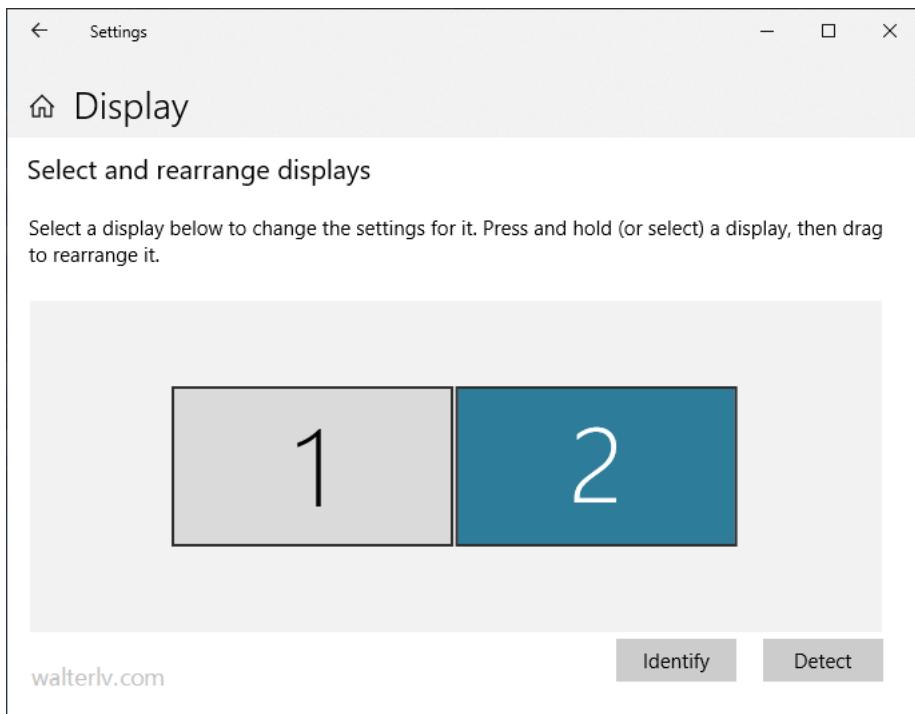
Windows 7 中还额外提供了传统 Windows XP 风格 DPI 缩放比例的选项（此选项在 Windows 8 之后就删掉了），这也是在修改 DPI 值，只不过可以选择非 1/4 整数倍的 DPI 值。



以上系统下我们可以禁止DWM，该模式我们称之为Basic模式，这种模式下的高DPI效果和XP一样。  
如何禁止：在服务管理器中找**Desktop Window Manager Session Manager**这个选项就可以禁止  
但是在不禁止DWM模式，我们就可以看到以上已经讨论过的“放大镜”效果了

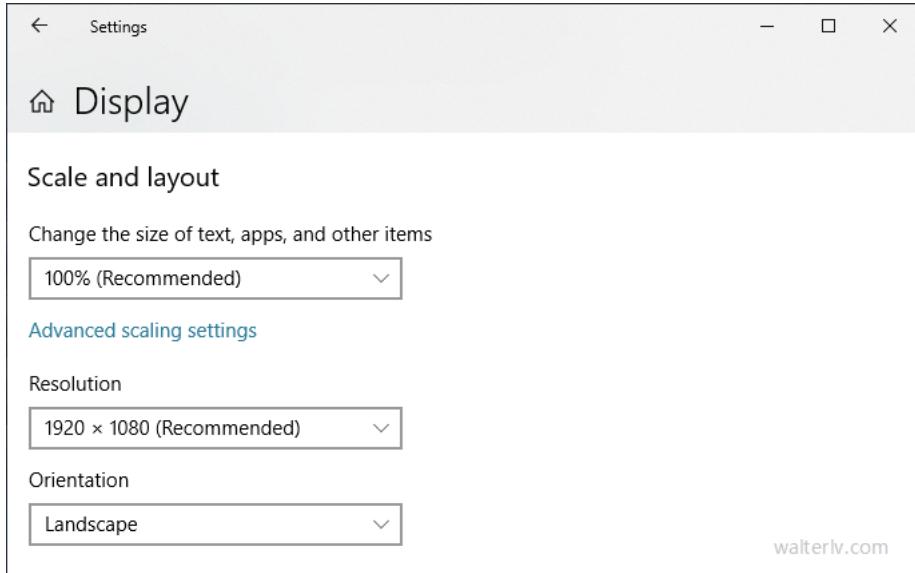
- Windows 8.1以后阶段

Windows 10 中的多个屏幕选择



▲ Windows 10 中的多个屏幕选择

Windows 10 中针对每个屏幕的 DPI 设置



▲ Windows 10 中针对每个屏幕的 DPI 设置

如果用户在设置中更改了系统 DPI 值或屏幕 DPI 值，那么 Windows 系统会提示需要注销才会应用修改。

对于 Windows 8.1 以下的系统，注销是必要的。因为系统 DPI 值如果不注销就不会改变，应用需要在系统重新登录后有了新的 DPI 值时才会正常根据新的系统 DPI 值进行渲染。否则就是系统进行的位图缩放。

对于 Windows 8.1 及以上的系统，注销通常也是必要的。虽然屏幕 DPI 值已经更新，并且已向应用窗口发送了 Dpi Change 消息，但系统 DPI 值依然没变。**应用必须处理 Dpi Change 消息才会正常渲染。如果应用不支持屏幕 DPI 感知（后边讨论），那么使用的就是系统 DPI 值，于是一样的会被系统进行位图缩放（放大镜）。**

但事情到 Windows 10 (1803) 之后，事情又有了转机。现在，你可以通过在设置中打开一个开关，使得无需注销，只要重新打开应用即可让此应用获取到最新的系统 DPI 的值。

Windows 10 (1803) 中新增的“不模糊”设置项

# 岔 Advanced scaling settings

## Fix scaling for apps

Some desktop apps might look blurry when your display settings change. Windows can try to fix these apps so they look better when you open them the next time. This only works for apps on your main display, and it won't work for all apps.

Let Windows try to fix apps so they're not blurry



walterlv.com

方法是：打开“设置”->“系统”->“显示器”->“高级缩放设置”，在“高级缩放设置”上，打开“允许 Windows 尝试修复应用，使其不模糊”。

额外的，对于 Windows 8.1 及以上的系统，系统 DPI 值等于主屏在系统启动时的屏幕 DPI 值。

## 四、对 Windows 应用而言的 DPI 感知级别 (Dpi Awareness)

上面已经说过系统对于DPI的操作，我们再讨论一下应用程序，对于系统dpi改变后的适配方式。

现在Windows 的程序 DPI 感知级别经过历代升级，主要是有四种级别。

### • DPI Unaware

顾名思义，就是程序对系统的DPI不进行感知，对于应用程序本身，永远是96DPI，但是在系统级别的DPI改变时，系统直接对该感知级别的程序进行DWM放大，就是视觉上模糊的位图扩大，坐标位置还是不变。这是操作系统进行的操作。

简单理解就是应用程序告诉系统，本程序对于DPI改变没有适配，你操作系统帮我给用户适配一下吧。

### • System DPI Awareness

这个级别的程序，在系统启动的时候，系统所设置的DPI，程序会去识别这个DPI，然后根据这个DPI来选择自己程序编好的DPI进行识别，这个是程序层面的DPI适配，在初始化期间，他们使用该系统DPI值适当地布置其UI（大小控制，选择字体大小，加载资源等）。

这种适配不是模糊的看起来扩大，而是程序真真切切的变大了，这是程序级别的操作。

但是假设我们是双屏的屏幕，一个主屏幕是125%，另一个屏幕是150%，然后程序启动感受到了125%的桌面DPI，然后初始化的时候使用程序125%的初始化方式，但是当程序从125%的屏幕移动到150%的屏幕时候，程序不会再进行150%的初始化，而是让系统对其进行模糊的位图放大，这个放大就是操作系统层面的。

### • Per-Monitor

此模式下的程序是告诉系统，不要对我使用DWM放大，当系统DPI更改时，给程序发送WM\_DPICHANGED消息，应用程序自己负责为新DPI处理自身的大小调整。桌面应用程序使用的大多数UI框架（Windows通用控件（comctl32），Windows窗体，Windows Presentation Framework等）都不支持自动DPI缩放，需要开发人员自行调整其窗口内容的大小并重新放置其内容。

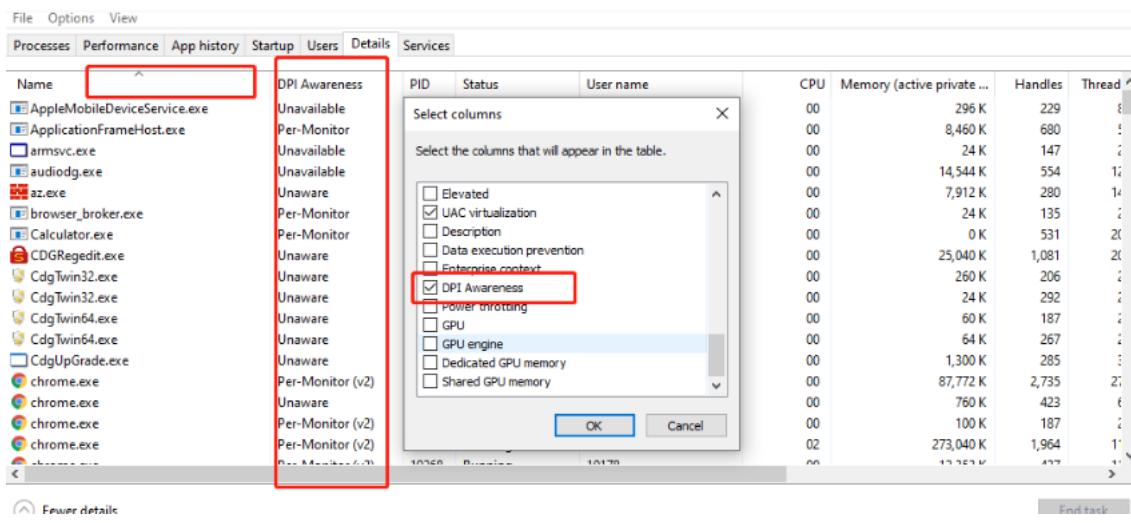
### • Per-Monitor (V2)

与Per-Monitor的模式相同，主要差别为以下几点：

- 1.当DPI更改时（顶级HWND和子HWND）通知应用程序
  - 2.该应用程序看到每个显示器的原始像素
  - 3.Windows永远不会缩放应用程序
  - 4.Windows自动缩放非客户区域（窗口标题，滚动条等）
  - 5.Win32对话框（来自CreateDialog）由Windows自动缩放DPI
  - 6.公共控件（复选框，按钮背景等）中以主题绘制的位图资产将以适当的DPI比例因子自动呈现
- 在Per-Monitor v2模式下运行时，当DPI更改时，将通知应用程序。如果应用程序没有为新的DPI调整大小，则应用程序UI将显得太小或太大（取决于先前DPI值和新DPI值的不同）。

对于应用程序的DPI感知级别可以在Win10下的任务管理器中查看

打开任务管理器，右键点击窗口栏，就可以弹出程序显示情况，勾选DPI感知，就可以在详细里面看到每个程序的DPI感知模式。



[Power details](#)

其他更深入的知识请移步[MSDN High DPI Desktop Application Development on Windows](#)

## 五、如何设置程序的DPI感知以及相关的API使用

理解了上面程序的DPI感知级别，我们再讲一下如何去设置程序的DPI感知级别，主要有以下两个办法：（方法名是我在MSDN上复制过来的，翻译过来很生硬，大家自行理解）

- 1) through an application manifest setting
- 2) programmatically through an API call

在此主要解释一下第二种方法，使用windows提供的API进行设置，就是以编程方式设置程序的DPI感知，虽然Windows不建议我们这么做，但是在我们的开发过程中，这些API还是挺实用的，以及其他相关程序DPI感知行为的API解释。

这些API有三个版本的迭代，以SetProcessDpiAwarenessContext系列为例

API	Windows 最低支持 版本	DPI Unaware	System DPI Aware	Per Monitor DF
SetProcessDpiAware	Windows Vista	N/A	SetProcessDpiAware()	N/A
SetProcessDpiAwareness	Windows 8.1	PROCESS_DPI_UNAWARE	PROCESS_DPI_SYSTEM_DPI_AWARE	PROCESS_DPI
SetProcessDpiAwarenessContext	Windows 10, version 1607	DPI_AWARENESS_CONTEXT_UNAWARE	DPI_AWARENESS_CONTEXT_SYSTEM_AWARE	DPI_AWARENE

SetProcessDpiAwareness ()函数的参数枚举：

```
typedef enum PROCESS_DPI_AWARENESS {
    PROCESS_DPI_UNAWARE,
    PROCESS_SYSTEM_DPI_AWARE,
    PROCESS_PER_MONITOR_DPI_AWARE
} ;
```

SetProcessDpiAwarenessContext()函数的参数枚举：

```
#define DPI_AWARENESS_CONTEXT_UNAWARE          ((DPI_AWARENESS_CONTEXT)-1)
#define DPI_AWARENESS_CONTEXT_SYSTEM_AWARE       ((DPI_AWARENESS_CONTEXT)-2)
#define DPI_AWARENESS_CONTEXT_PER_MONITOR_AWARE   ((DPI_AWARENESS_CONTEXT)-3)
#define DPI_AWARENESS_CONTEXT_PER_MONITOR_AWARE_V2 ((DPI_AWARENESS_CONTEXT)-4)
#define DPI_AWARENESS_CONTEXT_UNAWARE_GDISCALED    ((DPI_AWARENESS_CONTEXT)-5)
```

这DPI系列的相关函数还有获取程序DPI感知等，详细可去[MSDN](#)上查看

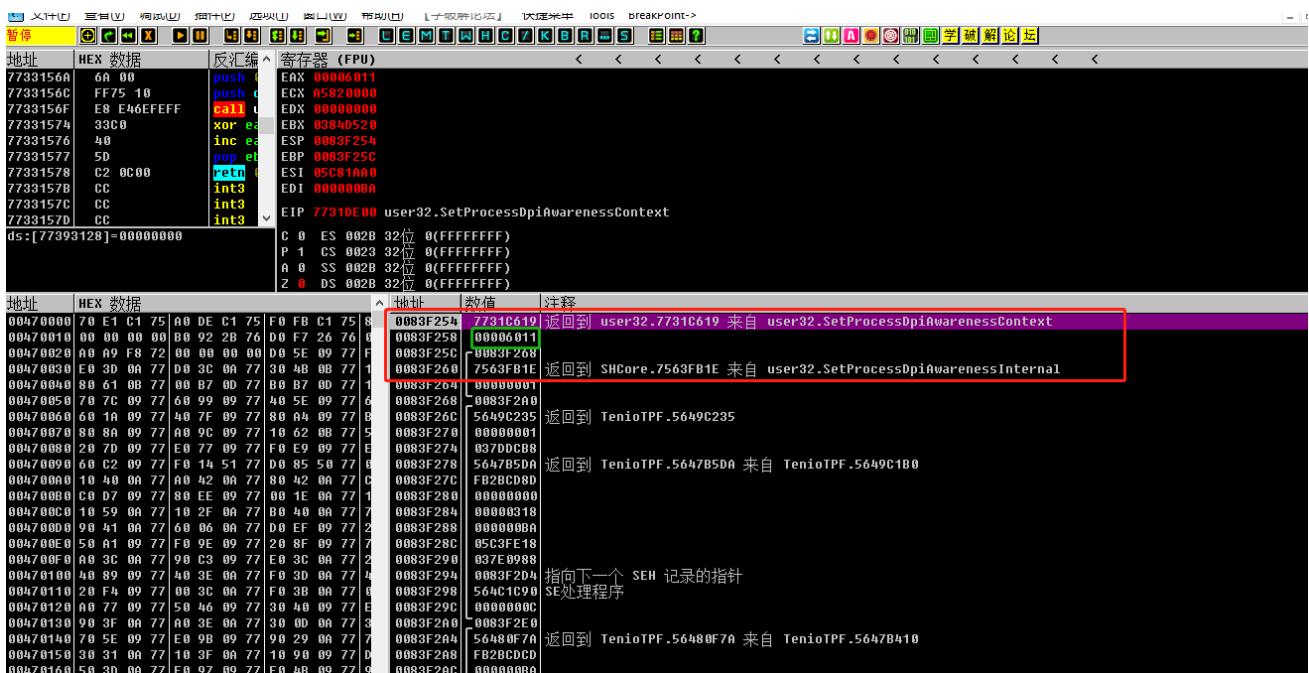
```
UINT GetDpiFromDpiAwarenessContext(
    DPI_AWARENESS_CONTEXT value
);
```

## 六、Wegame相关的DPI测试结果以验证以前的猜想

1. 使用win10打开Wegame发现其DPI感知方式为System，关闭Wegame

Name	DPI Awareness
vmware-hostd.exe	Unavailable
vmware-usbarbitrator64.exe	Unavailable
vsls-agent.exe	Unaware
vsls-agent.exe	Unaware
watchctrl.exe	Unaware
WeChat.exe	System
WeChatApp.exe	System
WeChatWeb.exe	System
wegame.exe	System

2. 在Win10下使用再OD打开Wegame找到SetProcessDpiAwarenessContext函数，下断点，发现参数为绿色框框内的数值，把其参数改为UNWARE启动



3. 去再任务管理器中查看Wegame 的感知方式就会发现变了

Name	DPI Awareness	PID
vmware-usbarbitrator64.exe	Unavailable	4244
vsls-agent.exe	Unaware	6160
vsls-agent.exe	Unaware	8868
watchctrl.exe	Unaware	8244
WeChat.exe	System	10596
WeChatApp.exe	System	1260
WeChatWeb.exe	System	15992
wegame.exe	Unaware	20840

我们在125%的分辨率下这两种感知方式打开后的Wegame可以看到明显的差别

UNWARE感知模式下的Wegame



SYSTEM感知下的Wegame



可以明显看到UNWARE模式下的Wegame是模糊的，也就是系统对其100%的大小进行了位图缩放，其GetWindowRect依旧是100%

## 七、其他神奇的DPI检测代码分析以及小坑坑

有时我们用系统给的API获取系统桌面的DPI发现始终得到的数值为96  
比如说

```
GetDpiForSystem()
```

比如说

```
HDC hdc = GetDC(0);
int dpi = GetDeviceCaps(hdc, LOGPIXELSY);
DeleteObject(hdc);
```

可以猜想到，是因为我们编写程序的时候，给程序设置的DPI感知模式位UNWARE所以他就始终感知都是100%，如果使用程序编辑设置为系统感知DPI(System DPI Awareness)的模式，就可以获取系统正确DPI了  
在编写程序时，使用上述SetProcessDpi系列函数就可以

```
SetProcessDpiAware();
SetProcessDpiAwareness(PROCESS_SYSTEM_DPI_AWARE);
SetProcessDpiAwarenessContext(DPI_AWARENESS_SYSTEM_AWARE);
```

## 通过逻辑宽度与物理宽度的比值来获取桌面DPI

```
void getdpi()
{
    // 获取窗口当前显示的监视器
    // 使用桌面的句柄。
    HWND hWnd = GetDesktopWindow();
    HMONITOR hMonitor = MonitorFromWindow(hWnd, MONITOR_DEFAULTTONEAREST);

    // 获取监视器逻辑宽度与高度
    MONITORINFOEX miex;
    miex.cbSize = sizeof(miex);
    GetMonitorInfo(hMonitor, &miex);
    int cxLogical = (miex.rcMonitor.right - miex.rcMonitor.left);
    int cyLogical = (miex.rcMonitor.bottom - miex.rcMonitor.top);

    // 获取监视器物理宽度与高度
    DEVMODE dm;
    dm.dmSize = sizeof(dm);
    dm.dmDriverExtra = 0;
    EnumDisplaySettings(miex.szDevice, ENUM_CURRENT_SETTINGS, &dm);
    int cxPhysical = dm.dmPelsWidth;
    int cyPhysical = dm.dmPelsHeight;

    // 缩放比例计算 实际上使用任何一个即可
    double horzScale = ((double)cxPhysical / (double)cxLogical);
    double vertScale = ((double)cyPhysical / (double)cyLogical);
    assert(horzScale == vertScale); // 宽或高这个缩放值应该是相等的
}
```