Windows 程序设计精髓：从 API、UI、结构到商业产品

当下，Android、Linux、Mac 等设备也大行其道，PC 软件似乎已经日薄西山。然而，我们大多数人平常工作、学习、娱乐的环境都是Windows平台，仍然在使用 PC 机器上各种软件，即使从事linux开发的技术人员，也很多使用像xshell/SecureCRT这类支持SSH协议pc软件远程登录到linux机器上来操作，实际工作的平台仍然是Windows，使用的软件仍然是pc软件。

我接触 Windows 开发有十多年了，先后担任过像股票行情资讯系统、手机助手等多 款 PC 软件的主程工作，也“客串”过 PC 端软件的产品经理，对用户使用 PC 端软件的行为习惯和用户心理有一定程度的了解。

本文将 PC 软件开发中涉及到的 Windows API 编程技术、界面库、程序库、框架结构设计以及商业 PC 产品设计思路等方面来分享我的经验，内容包括 Windows 软件开发需要掌握的技术和产品理念，希望对从事 Windows 软件工作的开发人员和产品人员有一定的帮助和启示。

这里说明一下，这篇文章讨论的内容不是Windows程序的具体编码细节，而是Windows平台上从程序设计到产出一款商业软件的方法和思想理念，当然这种思路也适合移动端等其他客户端软件。如果您需要学习Windows程序方方面面的技术原理和编码细节，您可以关注我的live。

一、Windows程序的特点

1. 严谨的接口设计

按操作系统的技术栈发展来看，相比较类unix操作系统比较短小精悍的命名风格，windows操作系统提供的函数接口以及各种函数参数的命名都是很清晰而已容易看懂的。虽然古怪的匈牙利命名法（下文将介绍）让Windows程序看起来有点“中世纪风格”，但这种命名方法增加了windows程序的可读性和可理解性，后来者也不断模仿之。

首先，Windows提供的函数名称、结构体类型风格都非常统一，诸如CreateProcess、CreateThread、CreateMutex、CreateSemaphore等（当然也有例外，比如堆创建函数HeapCreate，这个据《The Old Things of Windows》（中文名《Windows编程启示录》）的作者Raymond Chen说，当时设计这个堆系列的接口函数的是微软另外一个部门，所以风格出现了不一致--！），扩展的函数和结构体在原有的基础上使用Ex，例如从CreateWindow 到 CreateWindowEx ， RegisterClass 到 RegisterClassExWNDCLASS 到

WNDCLASSEX。新的扩展的接口不仅兼容旧的接口的功能，还增加了另外一些新功能新特性，如果您用不到新的功能和特性，仍然可以使用旧的接口函数。

Windows在新旧系统兼容性方面做的也非常优秀，即使在最新的Windows操作系统上使用很老的Windows api仍然没什么问题，虽然很多api或者做法不推荐在新版系统中使用，但是你使用它们仍然不会有什么大问题。反观，现在的一些操作系统比如Android，低版本使用的API一个个被弃用，原来老的做法在新的系统上可能会导致程序直接crash掉。

再者，windows将大多数资源抽象成句柄（HANDLE），例如socket、进程对象、线程对象、画笔画刷对象，甚至连像Windows服务这样的东西也抽象成对象。这种抽象带来的好处就是可以用同一套逻辑去处理，比如操作一个对象必须先打开或者创建，然后使用，使用完了之后然后关闭。当权限不足时也不会因为越权而带来安全隐患。

2. 匈牙利命名法

在驼峰式（例如myBestFriend）、帕斯卡（例如MyBestFriend）或者连字符式（例如my_best_friend）等诸多命名法当中，匈牙利命名法是Windows程序的一大特色。虽然显得很奇怪，但是带来的便捷性和可读性不得不提。匈牙利命名法是比尔盖茨雇佣的第一代程序员查尔斯·西蒙尼发明的（Excel的主要开发者），这是一个传奇性的人物。匈牙利命名法的主要思想是给程序变量加上它的类型信息，例如一个表示数值的整形变量，可以叫nNum或iNum，表示屏幕尺寸的整型变量可以叫cxScreen、cyScreen，以NULL结束的字符串指针叫pszStr（sz的意思是String terminated with Zero，以0结束的字符串）。

当我们在代码中看到这样的变量时我们无需查看其类型定义。网上有很多论调说这种命名法已经过时。但笔者以为，这些观点难免有点以偏概全。在后来的很多系统设计上，很多地方可以看到匈牙利命名法的影子。这种命名法虽然看起来很拖沓，但是表达意思却是非常清晰。尤其是在阅读别人的项目代码或者维护一些旧的项目时，能读懂是维护的前提。

3. 消息机制

之所以把Windows消息机制单独列出来是因为基本上可以认为它是以后所有的操作系统界面的模型的先驱，同时也是我们普通程序开发者应该学习和模仿的典范。它本质上就是一个消息队列，Windows消息队列的实现可以参考开源版本的”Windows”——ReactOS。ReactOS开发团队立志要做一个与微软Windows一模一样的操作系统，这里说的“一模一样”不仅指的是ReactOS的界面和使用习惯与Windows一模一样，甚至连提供的操作系统接口函数的名称、签名都必须一模一样，而且参与开发的人员必须没有阅读过任何Windows源码哪怕是微软泄露出来的代码也不行，如果发现这种情况，立马开除，并且立即重写被开除的人的所有代码。所有关于ReactOS的实现都必须通过逆向Windows来获得。所以ReactOS这个项目最大程度地仿真了Windows，并可以免费开源。

回到正题上来，这里要提一下的是，消息队列是软件产品中非常常用的技术，所以如何设计出一个好的消息队列属于开发者的基本功了。Windows可以参考，现在流行的很多专门的消息队列框架（如RabbitMQ）都是不错的学习资料。

4. 统一的用户界面使用习惯

Windows程序除了一些自绘的界面以外，大多数界面风格、菜单位置、使用习惯等都是一致的，例如如果某个菜单项后面有...（三个点），点击该菜单项一般都会弹出一个对话框；对于文本的编辑操作（例如复制、剪切、粘贴）等一般统一放在”编辑”菜单下面。这也是Windows和linux的不同，Windows的哲学是假设你不会操作，Windows教你如何操作；而linux是假设你会操作。所以，广大产品人员在设计UI的时候千万不要违背这种微软培养出来的多年用户习惯，否则您的用户很可能因为觉得您的产品操作非主流而放弃使用。

二、Windows下的界面库与自绘技术

一些应用软件为了追求一些特殊的界面效果，需要展现出不同于传统的Windows程序界面的样子（比如游戏界面）。由于Windows操作系统提供了最大化的界面自绘接口，所以市面上出现了很多开源的或非开源的、收费的或非收费的界面库。核心思想其实就是调用Windows GDI或GDI+函数进行自绘，GDI提供的自绘接口在一些追求界面细节的精细程度上做的不够且GDI接口都是C接口不符合现在开发软件使用的面向对象模型的理念，所以后来微软又专门推出来一套基于GDI的纯面向对象的绘制接口GDI+（GDI Plus），更不用说专门用于图形要求更高的领域的opengl、direct3D了。

比较著名的开源的界面库像DUILIB，这个库最先由一位德国人开发，后面中国的一些开发者接手，产生了种类繁多的分支。微信和百度云盘的pc客户端都是基于DUILIB。收费的界面库像迅雷的bolt。这里不再列举各种界面库的名字了，无论哪种界面库其核心技术都是自绘。这里不得不说一下这里的DUI思想，做Windows界面开发，这是一种必须理解的界面绘制思想。所谓DUI，即Directly Draw on Parent，即子控件对象直接绘制在父窗口上，也就是相当于只有父窗口一个窗口句柄。原来可能每个子控件都是窗口，windows消息机制决定，窗口越多，窗口的消息泵也就越多。为了减少这些消息泵，子窗口直接绘制在父窗口上，然后利用PtInRect这类API函数去检测鼠标是否位于这些绘制出来的控件上，然后利用PostMessage/SendMessage等API产生消息，交给父窗口处理，以此模拟这些无窗口句柄（hwndless）的控件可以响应消息的效果。

衡量一个界面库的优劣不仅要看这个界面库的功能，也要看它的流畅性（性能）和用户友好性程度。腾讯QQ从2009版本开始，采用了这种思想，到今天的2017版本，其界面的设计是一个经典的范例，而2009版本也是QQ应用DUI思想，UI产生质的飞跃的一个转折点。当然个人觉得界面库就该做好界面库自己的工作，现在一些界面库的作者因为一些利益的驱使，在其发布的界面库里面包含了方方面面的功能，核心的界面功能不去优化，一些与界面有关的类对象，因为继承链的关系，体积已经达到几十k甚至更大，并号称“旗舰版”。这很容易对一些初学windows界面开发的新手造成误导。界面是客户端软件的脸蛋，也是用户体验好与差的第一衡量标准，一个用户体验好的软件界面是开发人员、产品人员和美工人员共同努力的结果。

三、Windows上的程序库

Windows上的程序库最经典的当属mfc（Microsoft Foundation Class）。网上有大量的言论说mfc已经过时，mfc很臃肿。这里我有必要澄清一下，根据我个人的学习成长历程来看，学习Windows程序设计正确的顺序是先学习Win32 API，然后学习mfc框架。mfc可能不是一个最好的框架，但一定是最好用的框架。当然学好mfc是一个先易后难再易的过程。mfc的很多设计理念和思想都是非常优秀的，比如消息路由、C++对象的序列化与反序列化、动态创建、视图/文档对象模型等等。在后来者的各种软件框架当中都能看到似曾相识的影子。从另一个角度来讲，每个开发者可能都想成为程序架构师，而一般都是从学习模仿别人的框架开始，也就是说你的脑中应该有一些经典的框架设计。那么mfc可能是你学习成本最低、收获最大的框架。君不见Qt为啥也要抽象出QtApplication这样的对象？桃李不言，下自成蹊。

有人说mfc很臃肿，这点mfc真的很冤枉。mfc之所以臃肿是因为它是作为一个面面俱到功能型的程序框架，最大程度的去让使用框架的人减少重复劳动并快速生成一套可以使用的软件。功能多了，生成的库文件也就相对来说大一点。当然在现今的磁盘容量、内存容量、网络带宽条件下，mfc那几兆的库文件其实已经微不足道了。

当然，如果你不需要mfc这种重量型的框架，你可以使用WTL。WTL是基于C++模版的，程序库都是一些头文件，嵌入到你的程序中进行编译，本身没有任何二进制库文件。WTL没有mfc那样的方方面面的功能，只是将一些与窗口（包括控件）相关的东西封装起来（当然也包括一些常用的工具类）。市面上有很多软件都是基于WTL做的扩展，例如360安全卫士、金山卫士等等，我的开源即时通讯软件flamingo pc客户端也是基于WTL做的一套自绘界面库。

近些年，随着移动设备安卓、IOS的流行，Mac机器和linux平台也开始受到很多用户的欢迎。很多软件开发商为了减少开发成本、缩短开发周期，一套程序可以同时运行在多个平台上，Qt这样的跨平台框架流行起来。但是Qt这种庞然大物以及它的学习成本，笔者实在不敢恭维。

总结起来，不管哪种程序库，一般适合自己的软件项目才是最好的，没有必要为了使用框架而使用框架。在我带队的公司最近几个pc端项目都是我仿照mfc的思想用Win32 API从底层开发的，没有使用任何现成的框架。鉴于此，广大开发者或者公司决策人员，在做产品技术选型时，要综合考虑多方面因素去选择一款合适的程序框架。

四、PC端软件的组织结构、架构设计

关于PC端软件的组织结构和框架设计，最核心的其实就是界面逻辑和数据处理，数据处理可能是一些复杂计算或者网络通信。最经典的也就是三层结构，即UI层、数据加工层和网络通信层。关于它们的具体细节，我在我的另外一篇文章《客户端软件的结构思考》中详细地介绍了：。文中有具体的细节和具体的案例分析，有兴趣的可以参考一下。

五、商业PC产品设计思路

这个话题如果展开来说，是一个非常大的话题。这里我就具体而微的来说一下我的经验。一款成功的商业软件，在开发之前要做可行性分析和竞品分析。如果你的软件目前市面还没有，那么你在决定开发之前要对它做可行性调研，诸如开发技术难度、开发周期、是否有市场需求、盈利点等等都应该是重要的考虑因素；如果你的软件目前市面上已经有了，那么您需要做竞品分析时，您要吸纳采用同类软件的优点，反思同类软件的不足用于改进或者完善自己的产品。我曾经带队开发过一款邮件采集与加工系统，用户是某知名基金公司的研究员们，他们会将原创的各种金融研究报告汇总至指定邮箱。

这款软件就去从该邮箱读取这些邮件，并解析出来，然后生成对应格式的报告在金融平台上展示，程序需要自动识别邮件的标题、发件人、分组、邮件主题等内容，还要自动处理有附件的邮件。这款软件的业务其实并不复杂，但是在海量数据的处理的性能上，以及自动处理和加工邮件的内容是一个重难点。所以我在和产品人员沟通时，也会重点就这些方面的细节做好一一沟通落实，在开发人员的安排上也会往这些方面做一定的倾斜。

商业软件的另外一个问题就是程序的发布方式，这包括两个方面：

1.程序的发版问题

对于一些安装体验要求不是很高的软件，目前像NSIS或者InnoSetup这样现成的打包工具已经足够用了，当然如果您需要独特的安装程序，您可能需要自己去开发。安装程序本质上是检测软件所在的运行环境、释放程序文件到指定位置以及写入一些初始化信息或配置信息去注册表和配置文件中。

2.程序的自动升级问题

考虑到程序的bug修复和新功能的后续更新，大多数客户端软件都可以自动升级。当然苹果的App是行业内的一个奇葩。所以为了你后续的商业利益，自动升级这个功能一定是要特别重视的。只要自动升级功能不出问题，无论您的软件当前如何，您都有“补偿”或者改进的机会。

商业软件的另外一个需要注意的是，就是用户体验问题。良好的用户体验，能让用户对您的软件产生依赖。反观一些软件，一些常用的快捷键要么没有，要么非常难记难用。这都是经典的反面教材。

限于笔者水平和经验有限，文中难免有偏颇之处，欢迎提出批评指正意见。