桌面应用软件，在操作系统中，通过操作系统桌面窗体展现应用内容，并实现用户交互操作。编程语言有很多种，但并不是每一种语言都具有支持UI开发的能力，很多语言支持UI开发能力，是有人专门为之实现的UI开发支持平台。简单地说，需要有专门的语言UI库支持，如C++中的Qt库、MFC库，Java的Swing、AWT相关依赖库，否则语言是没办法进行快速UI开发的（可以通过编程语言调用系统函数，但很复杂）。

本文仅讨论常用的编程开发语言进行桌面应用软件开发的相关内容，对于诸如Matlab的m语言、Labview以及其他的类似语言，并不适合广泛的桌面应用软件开发的情况，将不涉及。

# 1. 有哪些语言支持桌面应用开发

桌面应用开发的方式有很多种，一般分成三大类：  
一是原生编译运行的开发技术，这一类技术，通过编译->可执行文件方式，可执行文件直接调用系统API，完成UI绘制等。这类开发技术，有着较高的运行效率，但一般来说，开发速度较慢，技术要求较高。  
二是二次编译或解释运行方式，这一类技术和将要说的第三类技术类似，但又稍有一些不同，这一类语言是一开始就有本地开发和UI开发的。一次编译后，得到中间文件，通过平台或虚机完成二次加载编译或解释运行。运行效率低于原生编译，但平台优化后，其效率也是比较可观的。就开发速度方面，比原生编译技术要快一些。  
三是机箱模式，总是需要一个运行外盒，才能够运行。这一类虽然类似于基于平台或虚机，但是这一类一开始的目标不是桌面应用，而是Web应用开发，包括了常见的RIA开发技术和主流Web前端开发技术。这一类技术桌面应用开发中，不外乎两种方式，一种是单独开发了基于桌面的机箱，来提供运行平台。另一种就是对机箱打包嵌套，给机箱再加个外壳，让机箱运行在外壳上。运行效率上，经过不断地革新和优化，Flash引擎进化、V8引擎的出世，总体上能够满足要求。开发效率上，速度较快，毕竟平台支持比较高级，很多东西直接拿来用。  
**目前比较流行的：**

原生编译：  
C++/MFC（Win）  
Objective-C（Mac）

托管平台：  
C#/.NET Framework（Win）  
Java/Swing、AWT \\*

富客户端：  
Java/JavaFX \（Sun出的，富客户端）  
ActionScript/Flex、Flash（Adobe Air、Adobe Flash） \（Adobe出的，富客户端）  
SilverLight（微软出的，富客户端）（Win）  
Open Laszlo（挺渣的，我用过，效率低的不得了）

Web式：  
HTML5/H5 + WebKit 、Node.Js（NW.JS）、Electron\\*

其他：  
Python/PyQT、GTK+ \\*  
Ruby TK\\*  
Go \\* （部分方言支持，入手要慎重）

\*\* 有年头的，但也有一些使用量的：\*\*  
VB、Delphi

# 2. 选择语言的评判标准

* 运行效率
* 开发效率  
  貌似，运行效率和开发效率是一对矛盾的值，但随着各类渲染技术、引擎、优化的加深，这两个值也开始同步提高。实际上，下述的评判内容，基本包含在了开发效率中，只有各种依赖、支持良好，才能快速开发，除此，还有语言的易用性。
* UI库支持、渲染美化效果程度
* 网络库支持程度（是否支持网络，网络库支持协议（TCP/IP？HTTP? SMTP？POP3？RTMP？））
* I/O支持
* 数据库操作支持
* 算法库支持程度
* 其他第三方库支持程度，如JSON、XML支持，加密支持，压缩文件支持等等
* **跨平台能力**（跨Windows、Linux、Macintosh OS X等操作系统）
* 新开发技术学习成本
* 开发平台成本

# 3. 具体分析

以下将分析各种UI开发技术的一些内容。但主体趋势无外乎以下方向：

运行效率：原生 > 托管平台 > 富客户端、Web

开发效率：原生 < 托管平台 < 富客户端、Web

就学习成本来说，很难说出谁高谁低，但是总体来说，富客户端和Web学习成本比较低，原生的学习成本比较高，因为原生的所面向开发应用场景比较固定，局限性大，所以考虑到技术人员的发展，成本算高的。

### C++

基于C++语言进行桌面应用软件开发的框架和平台较多，包括了Windows平台流行的MFC、支持多平台的Qt、GTK+等，都提供了GUI交互支持。

###### 运行效率

C++开发的应用软件，需要针对运行目标平台进行单独编译、生成可执行文件，运行效率极高，具有较好的性能。

###### 开发效率

C++开发要求开发人员具有较好的C++基础和应用能力，否则将会极大的降低开发效率。C++对内存管理要求较高，需要耗费较多精力处理。

###### UI效果

UI效果与具体的GUI交互支持方式有关。  
MFC采用贴图方式美化外观，且交互界面运行效率高，在Windows平台中，UI渲染有多种先进的图形库和渲染引擎支持，能够获得极佳的UI性能和效果。

###### 网络支持

C++语言对网络的支持较好，具有多种成熟的开源网络库可使用，如重量级的ACE，Boost的ASIO、陈硕的Muduo，仅支持Linux的libev、主要支持Linux的libevent。

###### I/O支持

C++语言作为编译型原生运行程序开发语言，本身就具有对I/O的天然良好支持，通过I/O流能够快速完成本地文件操作等等一系列I/O操作。

###### 数据库支持

C++语言对于数据库支持，主要依赖于数据库驱动程序提供的接口支持程度，目前对绝大对数主流数据库都有着良好的支持。

###### 其他第三方库支持

C++作为目前使用量较大的开发语言，其各个方面均有较好的支持

###### 跨平台能力

C++开发的程序跨平台能力较差，需要针对目标平台进行单独编译。当所使用的框架、类库不支持目标平台时，将导致失败的跨平台开发。  
MFC目前不支持跨平台，仅支持Windows平台。  
Qt和GTK+对较多平台均支持，包括Windows、Linux、Mac OS X。

###### 成本考虑

C++语言开发，一直是成本居然不下，前期开发架构搭建、业务逻辑堆砌成本高，后期测试、维护，每一项环节都占据了较高的成本使用。C++的开发、维护，都要求参与者有着较高的开发素养，因此人工成本也是一个大项。  
基于成本考虑，C++项目适合于运行效率要求高

### C#语言

C#语言是Windows中的当红语言，在Web、系统服务、桌面应用、富客户端等多有涉及。C#基于.NET Framewrok开发的桌面应用软件，也是Win系中的主流方式。  
此外，随着微软的努力，感谢Visual Studio Core和.NET Code，实现了跨平台开发和运行。

###### 运行效率

C#桌面应用基于托管式运行，效率相比于C++较低。

###### 开发效率

C#开发效率较高较快，入门快、上手快，但后期优化困难，需要较为深入的研究。

###### UI效果

C#的UI效果通过皮肤控制。由于C#消耗资源较多，在低配机器上，UI流畅度较低。

###### 网络支持

C#网络支持良好。

###### I/O支持

C#本地化支持良好，面向对象操作的封装，使得各种I/O操作都很便捷。

###### 数据库支持

C#对SQL Server数据库支持良好。对其他数据库，需要特殊的支持。

###### 其他第三方库支持

C#本身做了较为完整的工具类封装，不仅在基本的JSON之类的支持良好，在加密、编码等方面也有着良好支持。

###### 跨平台能力

不太好，但是微软正在致力于跨平台。但微软不应该如此好心，他们从来不会让客户和用户离开win系平台的，这也许只是他们放出的一个糖衣。

###### 成本考虑

Windows平台采用C#开发简单的桌面应用成本较低。大中型、复杂应用需要慎重考虑。

### Java

Java是目前服务器端的主流语言之一。Java语言用于桌面应用开发的产品较多，但更多的体现在Sun公司、Oracle公司自主研发产品，如Eclipse。一些对性能有较高要求的应用，并不适合用Java开发，而更适合用C/C++。但Java最为突出的有点是Write once， Run anywhere，跨平台性无与伦比。

###### 运行效率

基于JVM的运行方式，使得Java逊于C++。

###### 开发效率

有着Java基础的开发人员，能够快速开发。

###### UI效果

依赖皮肤效果。控件丰富度，需要第三方支持，原生控件种类较少，颇简陋。

###### 网络支持

支持良好，同时拥有众多第三方库支持。

###### I/O支持

支持良好。

###### 数据库支持

支持良好。

###### 其他第三方库支持

Java的第三方库支持众多。

###### 跨平台能力

Java有着良好的跨平台能力。

###### 成本考虑

Java开发成本较低，但其UI性能较弱，需要考虑后期项目膨胀后的应对办法。

### Objective-C

与C++同宗，但又有着很多不同的一门语言，随着苹果系列产品的畅销，OC开发也逐渐主流化。Objective-C由苹果公司推出，主要面向Mac系列系统使用，跨平台能力较差。

###### 运行效率

运行效率较高，属于原生编译运行。

###### 开发效率

熟悉OC语言，能够快速开发。

###### UI效果

UI效果借助贴图，控件样式较多。

###### 网络支持

支持网络，随着OC火爆，很多第三方库纷纷支持。

###### I/O支持

支持本地化操作。

###### 数据库支持

支持Mac上的数据库，如SQLite、MySQL等。

###### 其他第三方库支持

OC越来越流行，越来越多的第三方库出现，支持越来越好。

###### 跨平台能力

不具有跨平台能力。

###### 成本考虑

仅支持Mac OS X。

### 富客户端

富客户端一度是浏览器的霸主，但随着H5的兴起，更多更绚烂的效果由H5实现，甚至H5页肩负起了RIA的任务，老牌富客户端开发技术主键平淡，但不可否认的是，以Flash为主的富客户端依然对Web RIA拥有着不可撼动的主流地位。本文主要讨论桌面应用，富客户端中，很大一部分是可以实现桌面应用开发，当前最流行的Flex/Flesh，是典型的代表。富客户端技术开发语言根据平台不同而不同，包括了Flash/Flex的ActionScript/OpenLaszlo、JavaFX的Java、SliverLight的C#等等。富客户端技术并不是一个单纯技术，而是以其他开发语言和技术支撑起来的复合型开发技术。

###### 运行效率

富客户端运行基于其特定的环境和运行平台，如Flash播放平台、JVM、.NET Framework插件等，从而导致了相比于原生编译程序，其效率较为低下，从而不适合进行复杂功能和高强度运算处理应用的开发。  
随着技术不断的革新，富客户端在运行效率上也有着较大的提升，但综合来看效率普遍偏低，因此注重高效率的程序开发可以绕过这一块另择通路。

###### 开发效率

富客户端开发技术，往往是基于其他技术而形成的综合技术。  
Flash/Flex的ActionScript是一类面型对象的脚本开发语言，该语言语法与一般面向对象开发语言类似，在语法格式上，其变量声明格式与传统的编程语言略有不同（类型后置写法），但一旦熟悉之后，基本无碍。除此，Flash/Flex的UI界面开发，通过特殊的皮肤机制实现，需要投入一定的精力研究。  
JavaFX目前支持Java语言开发，对于一般Java开发者来说，需要熟悉JavaFX带来的新内容，能够较快上手。  
SliverLight基于C#语言，C#入门极易，但优化困难，所以入门级开发较快，后期优化比较扎手。

###### UI效果

富客户端的UI效果，一般靠皮肤支撑。JavaFX在3D图形支持上稍弱，有待加强。

###### 网络支持

富客户端天然具有网络支持能力，通常具有较强的多媒体支持能力。Flash/Flex支持更多的多媒体网络协议，如RTMP流媒体传输协议等，很多在线直播平台均采用该网络技术。

###### I/O支持

富客户端桌面应用开发中，借助本地化操作支持，能够较好实现I/O操作。

###### 数据库支持

富客户端开发中，不同类型对数据库连接和操作的支持不一致。ActionScript对这一类支持较差，并且缺乏第三方库支持。JavaFX天然使用Java语言，在各类数据库支持方面较好。SilverLight采用C#语言，对SQL Server数据库支持良好。

###### 其他第三方库支持

由于Adobe开发的Flash/Flex ActionScript开发平台，尚处于闭源状态，因此第三方拓展较少，但其本身已经具有较多的支持，如JSON支持等。  
JavaFX能够使用Java开发中的各类第三方库，支持绝对良好。

###### 跨平台能力

富客户端都具有较好的跨平台能力。

###### 成本考虑

富客户端开发成本，需要根据不同类型RIA区别对待。  
综合考虑，JavaFX开发成本较低。Flash/Flex开发成本稍高。

### Web

Web语言实现桌面应用开发，实际上和RIA类似，主要的套路是利用浏览器引擎完成UI渲染，目前比较主流的是IE浏览器引擎和WebKit引擎。随着H5前端技术、NODE.JS等的大肆兴起，桌面应用逐渐冷淡。但基于浏览器引擎开发桌面应用的技术，却因此而得到发展，如基于node.js的NW.js、基于io.js的Electron等等，以及国内很多公司自主研发的开发平台。

###### 运行效率

随着V8引擎的发布，Node.js借助其实现了服务器端JS编程的可能，作为桌面应用，其性能也有大幅度提升，可以与其他桌面应用开发技术相提并论。

###### 开发效率

使用前端技术开发，对于有前端开发经验的人来说，在克服了开发环境和运行环境搭建后，稍微摸索下开发技巧，之后的路将会比较坦荡。

###### UI效果

参考H5的效果，不难想象，UI效果将会达到比较炫酷的级别。

###### 网络支持

NW.js使用了WebKit支持UI渲染，同时使用node.js完成本地化操作，网络支持通过node.js，支持良好。

###### I/O支持

本地操作极为良好，通过JS提供的多种API，能够轻松完成各种本地操作，因此对I/O支持良好。

###### 数据库支持

目前，有很多的JS数据库操作库，这方面支持较为良好。

###### 其他第三方库支持

随着WebKit桌面应用开发的高速发展，同时以node.js等为主的JS后端编程实现，其他支持都已经逐步跟进，达到较好的支持。

###### 跨平台能力

不言而喻，拥有良好的跨平台能力。

###### 成本考虑

开发成本主要在环境搭建和开发入门方面，一旦入门后，有着丰富前端开发经验的人员，能够快速进行开发。同时，随着相应技术的发展，文档、资料、开源代码等逐渐丰富，开发成本逐步降低。  
维护成本方面，并非涉及到全新的开发语言，成本因此能够控制在较低的程度。

### Python

一种胶水语言，目前也支持了UI开发。

###### 运行效率

一般。

###### 开发效率

快。

###### UI效果

局限性较大，控件库丰富度较弱。

###### 网络支持

支持。

###### I/O支持

支持。

###### 数据库支持

支持。

###### 其他支持

支持良好

###### 跨平台能力

良好。

###### 成本考虑

成本较低，但不适合大型应用开发。

### Ruby

Ruby on Rails是当前炒作的比较火爆的一个概念，在Web开发中也有着一席之地。Ruby用于桌面应用开发，基于TK平台实现。Ruby，作为一个简单面向对象的脚本语言，其优点是使用简单，开发快速。

###### 运行效率

一般，脚本语言的效率都不会太高。

###### 开发效率

快，Ruby比较简洁，开发速度较快。

###### UI效果

作为脚本语言，由TK决定，控件库丰富度较弱。

###### 网络支持

支持。

###### I/O支持

基本支持。

###### 数据库支持

良好支持。

###### 其他支持

良好支持

###### 跨平台能力

采用虚拟机运行，跨平台能力良好。

###### 成本考虑

成本较低，但不适合大型应用开发。

### Go

Go语言的一些方言，目前支持UI开发。但即便是神一样的方言设计者，也有力不从心的时候，也无法与大型公司为靠山的开发平台丰富和稳定。但Go语言的开发速度，绝不是盖的。

###### 运行效率

视情况而定，由方言的特性决定。

###### 开发效率

快。

###### UI效果

绝大多数个人贡献，所以，控件库丰富度较弱。

###### 网络支持

视情况而定，由方言的特性决定，一般是支持的。

###### I/O支持

视情况而定，由方言的特性决定，一般是支持的。

###### 数据库支持

视情况而定，由方言的特性决定。

###### 其他支持

视情况而定，由方言的特性决定。

###### 跨平台能力

视情况而定，由方言的特性决定。

###### 成本考虑

成本较低，关键是快速开发效率高。

### 小结

1. 以C++语言为基础的桌面应用开发技术，具有较高效率，但跨平台实现比较困难，需要具体平台库支撑，但Qt和GTK+基本上达到了跨平台的能力。
2. OC专门针对Macintosh OS X，虽然具有较高的效率和较强的UI效果，但是跨平台能力限制了其广泛发展的可能性（除非苹果称霸天下……）。
3. C#是目前Windows系统上桌面应用开发使用最多的开发技术之一（好像把之一去掉，不过还有MFC镇压着，不可造次），C#虽然在效率上低于C++，但是在开发效率上甩C/C++几条街啊。
4. Java在UI、用户交互开发方面，一直是不温不火的状态，Swing、AWT已经是过去式了，但做一些小工具还是可以考虑的，毕竟Java语言会的人多。
5. 富客户端开发技术，在桌面应用开发市场中也占有一席之地，尤其是网络多媒体应用，我鹅的某宠就是这种模式。Flash/Flex是富客户端的大哥大，SliverLight也曾红火一时。JavaFX最近新军崛起，是否能够占据导一份市场，也要拭目以待。
6. WebKit为基础的Web式桌面应用开发，以Web引擎作为UI渲染工具，是一个高明的设计和思路。随着H5、Node.js的兴起，Web式桌面应用也在逐渐火爆，某道翻译软件目前应该是在借鉴NW.JS的基础上使用自主研发的heX完成UI开发。
7. 至于其他的语种，如Python、Ruby，脚本开发语言，效率不高，UI有限，但能够实现快速开发，小型软件方面也有一定优势。Go语言作为程序节的高冷语言，部分方言有对UI的支持，开发速度绝对快，但是要求对函数式编程有一定掌握，稍有些门槛。

### 那么该如何选择

１、 首先判断项目类型，对运行效率需求如何、对一些特殊的技术指标要求。如果有着较高的运行效率要求，那么选择范围也就较低了，多在原生编译和高度优化的平台上选择。如果对UI灵活性、自主性要求较高，也要重点考虑原生编译，因为原生编译更接近底层，自由度更大。  
2、 开发速度要求：如果有快速开发要求，尽量避免原生开发，一个是跨平台比较麻烦，另一个是原生开发，即便是提供了较多的UI支持库，也不可能像基于平台的技术有着良好的封装。  
3、 考虑技术人员的底蕴：如果具有特殊的技术人员，精通某一方面，那么必须要把这个考虑进去，这不但是技术方面的保证，也是开发成本的保证。  
4、 后期考虑：从长远考虑项目的要求，有没有什么特殊的考虑，如以后往Web上靠，那么就考虑NW.JS或者Electron等。往流媒体富客户端，那么要考虑Flex技术。等等。

# 结尾

支持桌面应用软件开发的技术，只要具有相应的UI支持基础就能够实现，因此相关技术并不止上述几类，但上述几大类是目前桌面应用开发的主流开发技术，占据了绝大部分的市场。其他较小众的桌面开发技术由于其小众化、文档匮乏，较难流行。