

## 21 | 图形界面程序的框架

2019-07-02 许式伟

许式伟的架构课

[进入课程 >](#)



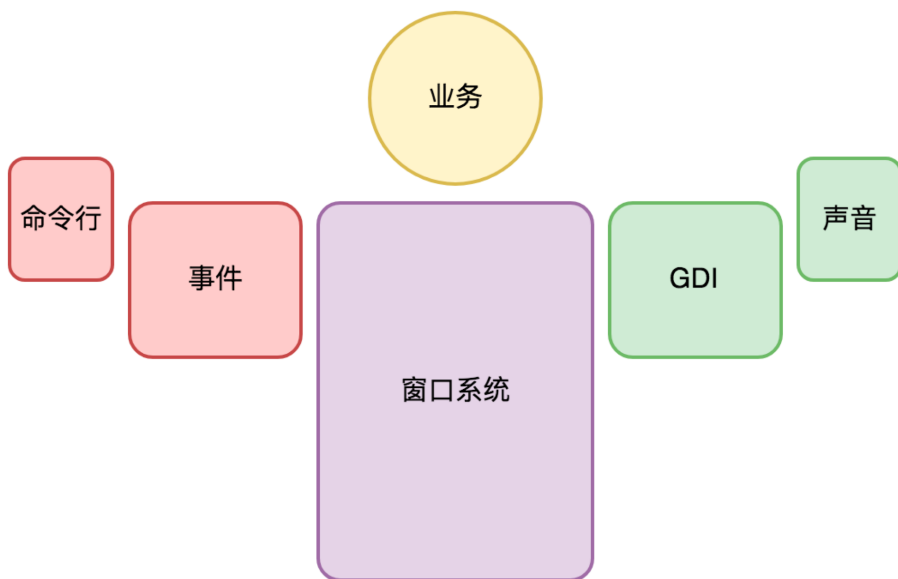
**讲述：姚迪迈**

时长 08:53 大小 8.14M



你好，我是七牛云许式伟。

上一讲我们回顾了交互的演化历程。今天，我们将关注点收敛到现在仍然占主流地位的图形界面程序。它的结构如下图所示。



实现一个图形界面程序，最大的复杂性在于不同操作系统的使用接口完全不同，差异非常巨大。这给开发一个跨平台的图形界面程序带来巨大挑战。

好在，尽管操作系统的使用接口有异，但基本的大逻辑差不多。今天我们从统一的视角来看待，谈谈图形界面程序的框架。

## 事件

无论是什么桌面操作系统，每个进程都有一个全局的事件队列（Event Queue）。当我们在键盘上按了一个键、移动或者点击鼠标、触摸屏幕等等，都会产生一个事件

(Event) , 并由操作系统负责将它扔到进程的事件队列。  
整个过程大体如下。

键盘、鼠标、触摸屏等硬件产生了一个硬件中断；

操作系统的硬件中断处理程序收到对应的事件  
(Event) ；

确定该事件的目标进程；

将事件放入目标进程的事件队列 (Event Queue) 。

## 窗口与事件响应

窗口 (Window) , 也有人会把它叫视图 (View) , 是一个独立可复用的界面元素 (UI Element) 。一个窗口响应发送给它的事件 (Event) , 修改内部的状态, 然后调用 GDI 绘制子系统更新界面显示。

**响应事件的常见机制有两种。**

**一种是事件处理类** (EventHandler, 在 iOS 中叫 Responder) 。通常, 我们自定义的窗口类会直接或间接从事件处理类继承。Windows 平台有些特殊, 为了让窗口类可复用, 且与语言无关, 它将事件处理做成了回调函数, 术

语叫窗口过程（WindowProc）。这只是形式上的不同，并无本质差异。

**另一种是用委托（delegate）。**顾名思义，用委托的意思是事件处理不是收到事件的人自己来做，而是把它委托给了别人。这只是一种编程的手法。比如，在 Web 编程中我们给一个界面元素（UI Element）实现 onclick 方法，这可以理解为是一种委托（delegate）。

有一个事件比较特殊，它往往被叫做 onPaint 或 onDraw。为什么会有这样的事件？我们想象一下，当一个窗口在另一个窗口的上面，并且我们移动其中一个窗口时，部分被遮挡的窗口内容会显露出来。

这个过程我们可能觉得很自然，但实际上，操作系统并不会帮我们保存被遮挡的窗口内容，而是发送 onPaint 事件给对应的窗口让它重新绘制。

另外，不只是窗口可以响应事件，应用程序（Application）也可以。因为有一些事件并不是发送给窗口的，而是发给应用程序的，比如：本进程即将被杀死、手机低电量告警等等。


当然如果我们约定一定存在一个主窗口（Main Window），那么把应用程序级别的事件理解为是发给主窗口的也可以。

## 事件分派

事件是怎么从全局的事件队列（Event Queue）到窗口的呢？

这就是事件分派（Event Dispatch）过程，它通常由一个事件分派循环（Event Dispatch Loop）来完成。一些平台把这个过程隐藏起来，直接提供一个类似 RunLoop 这样的函数。也有一些平台则让你自己实现。

例如，对于 Windows 平台，它把事件叫消息（Message），事件分派循环的代码看起来是这样的：

 复制代码

```
1 func RunLoop() {
2     for {
3         msg, ok := winapi.GetMessage() // 从事件队列中取出一
4         if !ok {
5             break
6         }
7         winapi.TranslateMessage(msg)
8         winapi.DispatchMessage(msg)
9     }
```



大体来说，就是一个简单的取消息（`GetMessage`）然后对消息进行分派（`DispatchMessage`）的过程。其中 `TranslateMessage` 函数你可能比较陌生，它负责的是将键盘按键事件（`onKeyDown`、`onKeyUp`）转化为字符事件（`onChar`）。

窗口有了父子和兄弟关系，就有了窗口系统。一旦界面涉及复杂的窗口系统，交互变得更为复杂。事件分派过程怎么知道应该由哪个窗口响应事件呢？

这就是事件处理链（`EventHandler Chain`）。

不同事件的分派过程并不一样。

对于鼠标或者触摸屏的触摸事件，事件的响应方理应是事件发生处所在的窗口。但也会有一些例外的场景，比如拖放。为了支持拖放，Windows 系统引入了鼠标捕获（`Mouse Capture`）的概念，一旦鼠标被某个窗口捕获，哪怕鼠标已经移出该窗口，事件仍然会继续发往该窗口。

对于键盘事件（onKeyDown/onKeyUp/onChar），则通常焦点窗口先响应，如果它不感兴趣再逐层上升，直到最顶层的窗口。

**键盘从功能上来说，有两个不同的能力：其一是输入文本，其二是触发命令。**从输入文本的角度来说，要有一个输入光标（在 Windows 里面叫 Caret）来指示输入的目的窗口。目的窗口也必然是焦点窗口，否则就会显得很 unnatural。

但是从触发命令的角度来说，命令的响应并不一定是在焦点窗口，甚至不一定在活跃窗口。比如 Windows 下就有热键（HotKey）的概念，能够让非活跃窗口（Inactive Window）也获得响应键盘命令的机会。一个常见的例子是截屏软件，它往往需要一个热键来触发截屏。

到了移动时代，键盘不再是交互主体，但是，键盘作为输入文本的能力很难被替代（虽然有语音输入法），于是它便自然而然地保留下来。

不过在移动设备里，不太会有人会基于键盘来触发命令，只有常见的热键需求比如截屏、调大 / 调小音量、拍照等等，被设计为系统功能（对应的，这些功能的热键也被设计为系统按键）保留下来。

## 窗口内容绘制

在收到 onPaint 或 onDraw 消息时，我们就要绘制我们的窗口内容了，这时就需要操作系统的 GDI 子系统。

从大分类来说，我们首先要确定要绘制的内容是 2D 还是 3D 的。对于 2D 内容，操作系统 GDI 子系统往往有较好的支持，但是不同平台终究还是会有较大的差异。而对于 3D 内容来说，OpenGL 这样的跨平台方案占据了今天的主流市场，而 Vulkan 号称是 NextGL（下一代的 OpenGL），其潜力同样不容小觑。

从跨平台的难易程度来说，不同平台的 GDI 子系统往往概念上大同小异，相比整个桌面应用程序框架而言，更加容易抽象出跨平台的编程接口。

从另一个角度来说，GDI 是操作系统性能要求最高、最耗电的子系统。所以 GDI 优化往往通过硬件加速来完成，真正的关键角色是在硬件厂商这里。由此观之，由硬件厂商来推跨平台的 GDI 硬件加速方案可能会成为趋势。

## 通用控件

有了以上这些内容，窗口系统本身已经完备，我们就可以实现一个任意复杂的桌面应用程序了。



但是，为了进一步简化开发过程，操作系统往往还提供了一些通用的界面元素，通常我们称之为控件 (Control)。常见的控件有如下这些：

静态文本 (Label);

按钮 (Button);

单选框 (RadioButton);

复选框 (CheckBox);

输入框 (Input, 也叫 EditText/EditText);

进度条 (ProgressBar);

等等。

不同操作系统提供的基础控件大同小异。不过一些处理细节上的差异往往会成为跨平台开发的坑，如果你希望一份代码多平台使用，在这方面就需要谨慎处理。

## 结语

总结来说，桌面应用程序通常由用户交互所驱动。我们身处在由操作系统约定的编程框架中，这是桌面编程的特点。

在操作系统的所有子系统中，交互相关的子系统是毫无疑问的差异性最大的子系统。我们这里列了一个简单的对比表格：

类别	Windows	iOS	Android
事件处理	MSG WindowProc	UIEvent UIResponder, delegate	InputEvent delegate
事件分派	GetMessage TranslateMessage DispatchMessage	RunLoop	Looper
窗口	Window	UIView	View
绘制 (2D)	GDI GDI+ Direct2D/DirectDraw	CoreGraphics	Canvas
绘制 (3D)	Direct3D OpenGL Vulkan (非官方)	Metal OpenGL Vulkan (非官方)	OpenGL Vulkan
应用	-	UIApplication	Application

这还不是差异的全部。要做一个跨平台的桌面应用程序并不容易。我们需要面对的平台太多，简单罗列，如下所示。

PC：Windows、MacOS、Linux 等；

PC 浏览器：Chrome、Safri、Firefox 等；

手机 / 平板 / 手表：Android（不同手机厂商也会有细节差异）、iOS 等；

小程序：微信、支付宝、快应用等。

怎么安排不同平台的优先级？怎么规划未来版本的迭代计划？选择什么样的跨平台方案？这些问题在业务架构之外，但极其考验架构师的决策能力。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们将聊聊“桌面程序的架构建议”。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

 极客时间

# 许式伟的架构课

---

从源头出发, 带你重新理解架构设计

---

许式伟  
七牛云 CEO



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 20 | 桌面开发的宏观视角

下一篇 22 | 桌面程序的架构建议

## 精选留言 (9)

写留言



勇闯天涯

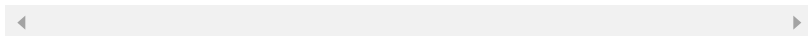
2019-07-03

高屋建瓴的视觉，大部分做应用层开发的同学缺乏这个视角的了解，感谢许老师！

是不是只有从这个角度了解了系统，才能更好的开发一套应用系统出来？还需要哪些知识的了解？

展开 ▼

作者回复: 以上两讲是基础的背景知识，下一讲更为关键一些。



👍 4



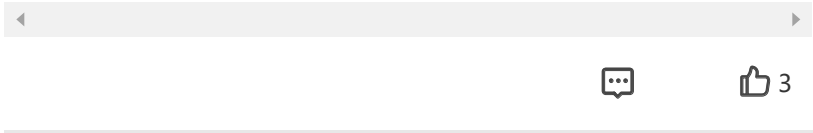
周伟民

2019-07-02

许老师，你好。Qt里面的信号和槽机制非常好用，实现了两个对象的同步通信和异步通信。您觉得信号和槽的性能

和实时性怎么样？

作者回复: qt 的 signal-slot 其实就是一种 delegate 机制



**Geek\_JsonHu**

2019-07-03

醍醐灌顶，底层系统的思想大体是一致的！

展开 ∨



👍 2



**lckfa李钊**

2019-07-02

最近在研究Google的flutter框架，凭着其一统桌面端开发的野心，就值得好好学习下。从开发语言到框架都不同以往。不过本着太阳下没有新事物的原则，从消息循环到事件的响应开始深挖型学习，一切问题都不是事了。

展开 ∨



👍 2



**Geek\_88604f**

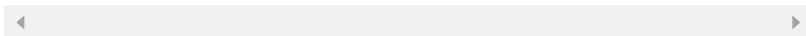
2019-07-06

GUI程序逐渐被WEB应用程序替代，这里有哪些原因，许

老师?

展开 ✓

作者回复: 下一讲会谈这个话题



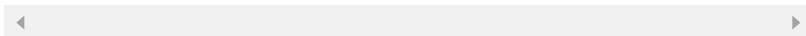
**靠人品去赢**

2019-07-04

老师，如果做一个大的中间层来处理系统差异（类似硬件中我们处理硬件差异，驱动管理一样）。每个系统对应一个驱动来处理跨平台，会不会好一点，这种方案难点在哪？像Java，flutter跨平台的原理？

展开 ✓

作者回复: 后面会谈跨平台



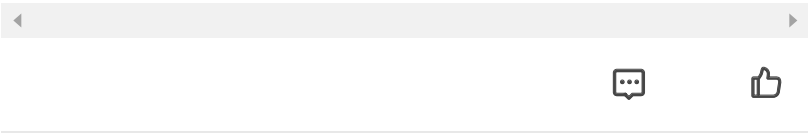
**ljf10000**

2019-07-02

“操作系统的硬件中断处理程序收到对应的事件（Event）；  
确定该事件的目标进程；”

如何确定目标进程呢？中断处理过程中获取当前系统的...  
展开 ∨

作者回复: 以上过程有省略，操作系统一级也是有键盘事件缓存的。有时候系统比较卡的时候我们按键盘主机的内置会有提示音，这是操作系统键盘缓冲满你的按键被丢弃的意思。至于确定目标窗口存在误差，这个从交互来说不是大问题



**随心而至**  
2019-07-02

好的实现方案大家都会相互借鉴，  
展开 ∨



**Aaron Cheung**  
2019-07-02

打卡21 这方面还没有思考总结过  
展开 ∨



