Atiti.ui原理与gui理论

[1. 概论 2](#_Toc17232)

[2. ui的类型 2](#_Toc2921)

[2.1. RMGUI vs IMGUI 2](#_Toc20349)

[2.2. Cli 2](#_Toc19798)

[2.3. Gui 2](#_Toc22523)

[2.4. Nui natural user interface 2](#_Toc17457)

[3. Ui发展的3个阶段 2](#_Toc20753)

[3.1.](#_Toc5148) **[1. 命令行界面 CLI](#_Toc5148)** [2](#_Toc5148)

[3.2. Gui(click 3](#_Toc13163)

[3.3. Nui(touch 3](#_Toc27614)

[4. Gui的原理 解决两个问题，“输入”和“输出” 3](#_Toc20565)

[4.1. Gui的结构 4](#_Toc18090)

[4.2. 界面引擎和图形引擎 4](#_Toc20714)

[4.3.  窗口管理控件系统 5](#_Toc8479)

[4.4. 事件系统 5](#_Toc32654)

[4.5.  图形系统 5](#_Toc25418)

[4.6. 布局系统 5](#_Toc13933)

[4.7. 一些杂项 utility： 5](#_Toc4015)

[4.8. 常用控件集合： 特殊控件： 5](#_Toc25309)

[4.9. 编辑工具： 5](#_Toc1955)

[5. Ui的趋势cli>gui>nui 5](#_Toc19662)

[5.1.](#_Toc30596) **[方便使用的动态机制](#_Toc30596)**[， 5](#_Toc30596)

[5.2.](#_Toc2732) **[方便的控件拓展机制](#_Toc2732)**[， 6](#_Toc2732)

[5.3. 脚本化(native>> html5) 6](#_Toc20875)

[6](#_Toc3421)

# 概论

# ui的类型

## RMGUI vs IMGUI

Immediate Mode GUI (IMGUI)

我来补充一种比较新颖的GUI实现模式，叫做Immediate Mode GUI (IMGUI)。这种类型的更多的适用于显示区域实时刷新的程序里面，例如游戏和CAD等。IMGUI这种实现模式的优势在于他在实现和实用上都会比传统的Retained Mode GUI (RMGUI)，例如Qt, MFC等，要简单不少。我自己两种模式都实现过，我认为IMGUI比较适合想学习自己写GUI的人上手。贴一个我自己实现的一个轻量级GUI库的效果图和程序，想玩玩的戳链接：[EDXGui](//link.zhihu.com/?target=http://www.edxgraphics.com/uploads/8/5/2/1/8521459/edxgui.zip" \t "_blank)。可以看出IMGUI也可以实现较为复杂的控件，给自己的杂七杂八的程序用绰绰有余了。

## Cli

## Gui

## Nui natural user interface

# Ui发展的3个阶段

人机界面的发展，主要经过了三个阶段

## **1. 命令行界面 CLI**

命令行的用户行为模式，是 记忆->输入，用户要背下各个命令，单后输入同机器交互，使用门槛高，现在只有专业人士和好莱坞电影里要表现黑客们的牛逼才使用

作者::  ★(attilax)>>>  绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ） 汉字名：艾龙，  EMAIL:1466519819@qq.com

转载请注明来源： <http://blog.csdn.net/attilax>

## Gui(click

苹果推出，windows发扬光大，其用户行为模式，是 识别->选择，用户不用背命令了，拿着软件就可以鼠标乱点进行探索，这个时候大部分软件的用户手册都没人看了，交互门槛降低，但是我们没接触过电脑的父辈还是会觉得有距离感，有学习门槛。  
GUI实际上是使用标准的一套图形界面语言（按钮，列表，文本框等），所有的应用需求都转化为这套界面语言呈现，信息内容要依附于界面语言形式表现

## Nui(touch

iphone及ipad的出现，将机器交互门槛大大降低，基本所有人包括老人小孩都能拿着就用。因为其交互语言发生了变化，不再是用标准化控件来设计界面，这时内容>形式，形式要依附信息内容的使用场景。符合人对内容表现和交互的自然预期，这样，界面引擎仅提供标准的控件就无法满足要求。

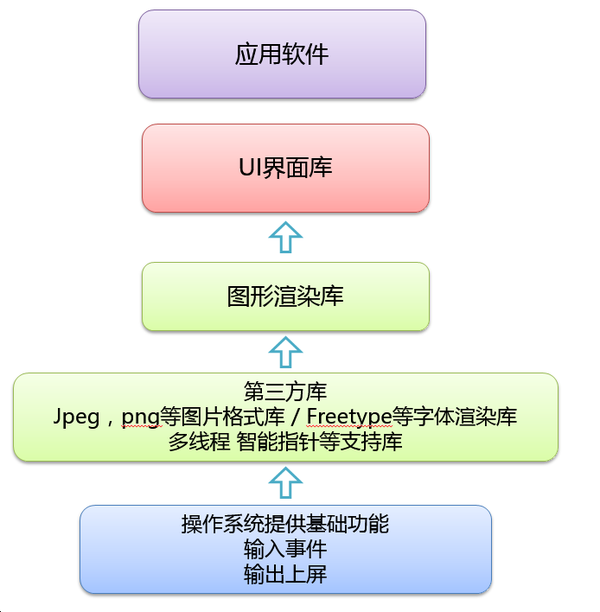
# Gui的原理 解决两个问题，“输入”和“输出”

图形界面，就是解决两个问题，“输入”和“输出” “输入”就是用户通过键盘，鼠标，触摸屏 定制设备按钮等各种硬件装置进行的输入操作，通过这些设备的驱动，最后被操作系统转化为各种事件消息，图形界面要响应并管理好这些消息； “输出”就是显示，让用户看到的内容，图形界面最终要在屏幕上显示出来，需要操作系统提供这个接口，本质还是显示芯片的驱动提供这个功能。不考虑效率的话，给个画点的函数就够了。但画点肯定太慢，所以一般会是一个拷屏的上屏操作，用缓冲防止闪烁； “获取输入消息”，“输出上屏”，这两点就是图形界面的“轮子”和“火”。两个方式都依赖操作系统提供的接口，而且不同操作系统的接口也不定相同，甚至同一操作系统也有不同的方法实现。对于跨平台图形界面引擎，这两部分的基本代码是不跨平台的，针对不同平台会有不同实现，但是这两部分代码不会很多，封装的好也不影响上层开发。

## Gui的结构

GUI库，有几个基本的子系统：

## 界面引擎和图形引擎



图形引擎和基本库</b><br>首先要说明的是，虽然经常说图形界面引擎，但界面引擎和图形引擎，这是两个不同的东西，就如同MFC和GDI，android UI库和skia的关系，图形引擎任务是提供基本的绘制方法，如何画位图，缩放位图，如何画线条 多边形等矢量图形，还有很重要的，如何绘制文字，这些依赖一些基本库，除非你想自己解析jpg png等图片格式，自己读取字体对文字进行矢量渲染。好在大部分这些库都是开源并且商业友好的（免费）<br>还有一些支持库，比如多线程，文件IO，数学算法。因此，一个图形界面引擎，有以下基本组成部分

## 窗口管理控件系统

## 事件系统

## 图形系统 渲染机制

## 布局系统

## 动画

## 一些杂项 utility：

## 常用控件集合： 特殊控件：

</b>基本框架稳定了，就要做一些比较常用的基本控件供应用层直接使用，按钮，列表，滑动条，进度条等<br><b>

特殊控件：</b>一些比较特殊的控件，比如模式对话框，弹出菜单，和常规控件的设计会有差异，他们基本需要全局管理。<br><b>

## 编辑工具：

</b>是否需要图形界面的编辑工具，对于静态界面，就如windows的对话框，各种按钮，列表框，有一个所见即所得的编辑工具会提高界面开发效率

# Ui的趋势cli>gui>nui

UI引擎，虽然会提供一套标准控件，但是更重要的，要满足两个需求

## **方便使用的动态机制**，

使用者不用考虑定时器，多线程，异步处理等，能实现界面元素的各种属性动画

## **方便的控件拓展机制**，

要更方便的定义一个新的控件，让使用者更专注于**定义控件行为**，而不用管控件的消息处理，绘制处理这两个层面的事情。之前的老HMI引擎，每次生成一个新控件，都要重载Draw和DoMessage两个方法，新的NUI引擎通过类似android库drawable层的封装，并引入信号槽机制，已可以像搭积木一样搭建控件，而不必再重写这两个方法

## 脚本化(native>> html5)

市面上许许多多的js库，其实就代表未来界面库的发展方向：不再使用传统的控件体系，而是用比控件更小粒度的原子控件---就比如迅雷界面库里的各种原子控件，又如js库里所使用的那些html元素。你会发现，只使用基本的几个元素，然后强化他们的拼装方式，能更方便快捷的实现产品提出的那些奇怪需求。  
有了原子控件，你拼装出异性控件更加方便，你要实现控件各种动画也更加简洁。当你还是觉得有点不足，  
  
现在的NUI引擎基本达到这两个目标。开发门槛低，

# 自己的ui 引擎

其实没有想象的复杂，只要抓住输入和输出两个线索。操作系统只需要给程序一个无边距窗口，窗口内允许贴图，窗口可以接受消息或事件，那么就可以发展出一套自己的GUI。

然后，你得定义一种逻辑概念，对应窗口，控件，Sprite，无所谓你自己决定....你的界面元素都是从这个概念逐层派生的，然后用Tree组织起来的。我的平台是用Sprite这个名称。

Sprite对象自带一张Bitmap，当然这个Bitmap也是我自己定义的数据结构以尽量与系统无关。最后显示实际上就是从背景的Cavas不断的贴上各层Sprite的bitmap得到一个总的Bitmap，然后调用系统的显示图片的函数(Windows下可用DirectX或者StreetchDIB)显示给用户。  
  
那么RenderOneTime里面显然就做两个事情，  
1，通知顶层Sprite，根据时间戳和自身状态更新自己的Bitmap，而顶层Sprite要做到这一点就得不断通知下层的Sprite，遍历。  
2，更新完毕后，得到一个矩形区域表示哪些地方变化了，然后用显示函数显示出来。  
这样，界面输出部分就完成了。系统可以按照设定的帧率更新界面，甚至支持动画。  
-------------------------------------------------  
  
那么输入部分就更简单了，自定义了消息队列和消息分发系统，从操作系统得到消息或者事件后，转化为自定义的消息，然后分发给对应的Sprite去响应就可以了。  
  
注意渲染和消息处理是两个不同的线程，所以同步设计很重要。  
  
-----------------------------------------------------  
字体处理部分还是部分系统相关的，移植时候需要一起改动。  
GDI画线什么的都是自己的函数，因为绘画对象直接是自己定义的内存Bitmap。  
-----------------------------------------------------  
这样一来，需要移植的时候只需要把系统相关的代码包括消息处理，显示处理，以及线程相关的代码等替换到新的系统，就可以完成移植了。目前我的产品可以跨Windows和mac平台。有些小不稳定，但总体还可以勉强达到产品应用水平。

# 

参考资料

如何用 C++ 从零编写 GUI？ - 图形用户界面 - 知乎.html