YYText 是如何運作的

上面的陰影效果是用以下代碼實現的：

可以看到先生成了 YYTextShadow，然後賦值給了 attributedString 的 yy_textShadow，然後再把 attributedString 賦值到 YYLabel 裡面，接著把 YYLabel 加入到 UIView 裡來顯示。跟蹤 yy_textShadow 發現，主要是把 textShadow 綁定到了 NSAttributedString 的 attribute 裡，key 是 YYTextShadowAttributeName，值是 textShadow，也就是先把 shadow 存起來，後來再使用。用 Shift + Command + J 快速跳轉到定義處：

這裡有個 addAttribute，它在 NSAttributedString.h 裡定義：

```
- (void)addAttribute:(NSString *)name value:(id)value range:(NSRange)range;
```

說明可以賦值任意的鍵值對給它。而 YYTextShadowAttributeName 的定義是一個普通的字符串，這意味著先是把 shadow 信息存起來，然後後面再使用。我們全局搜索一下 YYTextShadowAttributeName。

然後我們來到 YYTextLayout 裡的 YYTextDrawShadow 函數：

CGContextTranslateCTM 是說改變一個 Context 裡的原點坐標，所以

```
CGContextTranslateCTM(context, point.x, point.y);
```

是說要把繪製的上下文移動到 point 點。我們還是先搞清楚哪裡調用了 YYTextDrawShadow，發現是在 drawInContext 裡調用的。

在 drawInContext 裡，依次繪製方塊的邊框，然後繪製背景邊框、陰影、下劃線、文字、附屬物、內陰影、刪除線、文字邊框、調試線。

那麼到底哪裡用了 drawInContext 呢？可以看到裡面有個參數 YYTextDebugOption，所以這個函數一定不是系統的回調，而是 YYText 裡面自己調用的。

按住 Ctrl + 1 彈出快捷鍵，發現有四個地方調用了它。

drawInContext:size:debug 仍然是 YYText 自己的調用，因為 debug 的類型是 YYTextDebugOption *，是 YY 自身的。newAsyncTask 不像是系統的調用，addAttachmentToView:layer: 同理，所以極有可能是 drawRect:。

果然是，看右邊的快速幫助，有詳盡的解釋，幫助的下面也說明了是在 UIView 裡定義的。再看 YYTextContainerView，它是繼承了 UIView 的。

所以 YYLabel 是用了 YYTextContainerView 咯？然後讓系統調用 YYTextContainerView 裡的 drawRect: 畫出來？

奇怪，YYLabel 可繼承了 UIView。所以，YYText 裡應該有兩套東西！一套 YYLabel，一套 YYTextView，像 UILabel 和 UITextView 一樣。接著我們再回去看之前的 YYLabel 的 newAsyncDisplayTask，

很長，在中間的位置調用了 YYTextLayout 裡的 drawInContext。newAsyncDisplayTask，它又是在哪裡調用的呢？

在第二行被調用了。所以可以簡單地理解為 YYLabel 用了異步來繪製文本。而 _displayAsync 被上面的 display 調用了。看 display 的文檔，說是系統會在恰當的時間來調用來更新 layer 的內容，你不要直接去調用它。我們也可以給它打個斷點。

說明 display 是在 CALayer 的一次事務中調用的。為何用事務，大概是因為想批量更新，效率高點吧？不像是數據庫裡的回滾需求。

display 的系統文檔還說，如果你想你的 layer 繪製不一樣，那你可以複寫這個方法，來實現你自己的繪製。

所以，我們簡單的有了一點思路。YYLabel 通過複寫 UIView 的 display 方法，來異步繪製自己的陰影等各種效果，陰影效果先保存在了 YYLabel 的 attributedText 裡的 attribute 中，在 display 中繪製的時候再取出來，繪製的時候用了系統的 CoreGraphics 框架。

理清了一些思路後，會發現，真正強大的是什麼？一邊是把這麼多效果、異步調用等組織起來，一邊是對底層 CoreGraphics 框架熟練運用。所以對前面的代碼組織有了些了解後，接著我們深入到 CoreGraphics 框架上去。看看是怎麼繪製上去的。

讓我們重新回到 YYTextDrawShadow。

這裡，CGContextSaveGState 和 CGContextRestoreGState 包圍起了一段繪製的代碼。CGContextSaveGState 的意思是說，把當前的繪圖狀態拷貝一份，放到繪製棧裡。每個繪製的 Context 都維護著一個繪製棧。我也不清楚，裡面棧到底是怎麼操作的。先暫且理解為繪製 Context 前要調用 CGContextSaveGState，繪製 Context 後要調用 CGContextRestoreGState，之後中間的繪製就能有效地出現在 Context 裡。CGContextTranslateCTM 是移動到 Context 移動到相應的位置。先是移動到 point.x 和 point.y ，繪製的相應位置，至於後面移動到 0 和 size.height，倒不清楚了，後續再看看。接著取出了 lines，執行了 for 循環。

lines 是什麼？發現在 YYTextLayout 裡的 (YYTextLayout *)layoutWithContainer:(YYTextContainer *)container text:(NSAttributedString *)text range:(NSRange)range 賦值的。

接著翻到這個函數的定義處：

這個函數非常長，367 到 861 行，500 行代碼！看了頭尾，可見它的用處就是得到這些變量。lines 是怎麼得到的呢？

可以見到在一個大的 for 循環裡把一條一條 line 加入到 lines 裡。那 lineCount 是怎麼得到的呢？

第 472 行創建了一個 framesetter 對象，text 參數是 NSAttributedString，接著在 frameSetter 對象中創建了一個 CTFrameRef，接著從 CTFrameRef 得到了 lines。line 到底是什麼呢？我們給它打個斷點。

發現，shadow 這個字的 lineCount = 2，並不是我們想像中的字母個數。

所以猜測，白色的 Shadow 整個是一條 line，陰影也是一條 line？

YYText 裡有好幾個例子，只顯示其中一種效果，把其它的代碼註釋掉。發現很奇怪，Shadow 的 lineCount = 2，Multiple Shadows 的 lineCount 也是 2，可 Multiple Shadows 還有內陰影啊，應該是 3 條啊？

去找 CTLine 的蘋果文檔，說 CTLine 代表著一行的文本，一個 CTLine 對象包含著一組的 glyph runs。所以就是簡單的行數而已！看上面的斷點截圖，剛剛 shadow 之所以為 2 ，是因為它的文本是 shadow\n\n，看剛剛，\n\n 是故意加的，為了顯示美觀：

所以 shadow\n\n 就是兩行文本。CTLine 就是我們平時說的行。接著回去看我們的 lineCount：

這裡得到 CTLines 數組，從裡面的個數，然後如果 lineCount 大於 0 的話，得到每行的坐標原點。好了，有了 lineCount，我們接著看 for 循環。

從 ctLines 數組裡得到 CTLine，接著得到 YYTextLine 對象，然後加入到 lines 數組中。然後做一些 line 的 frame 計算。YYTextLine 的構造函數很簡單，先保存著位置、是否垂直排版、CTLine 對象：

lines 搞清楚之後，我們再回去之前的 YYTextDrawShadow 中去：

這下代碼簡單了。先獲取到行數，遍歷它，然後取得 GlyphRuns 數組，再遍歷它，GlyphRun 可以理解為一個圖元，或者繪製單元。然後從中得到 attributes 數組，用我們之前的 YYTextShadowAttributeName，獲取我們一開始賦值的 shadow，接著開始繪製陰影：

一個 while 循環，不斷繪製子陰影。調用 CGContextSetShadowWithColor 設好陰影的位移、半徑、顏色。接著調用 YYTextDrawRun 來真正的繪製。YYTextDrawRun 被三個地方調用了：

用來繪製內陰影和文本陰影以及文本。說明它是個通用方法，用來畫 Run 這個對象。

一開始獲取文字的變換矩陣，用 runTextMatrixIsID 來看看它是否原地不變，如果不是垂直排版或沒有設置圖元轉換的話，就直接上來畫。調用 CTRunDraw 來畫 run 對象。接著斷點發現，繪製一開始那個陰影時只進入了 if 裡面，沒有進入 else 裡面。

所以我們的陰影繪製就到此結束了！

總結一下，YYLabel 先把陰影等效果保存在 attributedText 裡的 attributes，複寫了 UIView 的 display 方法，在 display 中進行異步繪製，用 CoreText 框架得到 CTLine、CTRun 對象，從 CTRun 獲取到 attributes，之後再根據 attributes 裡的各屬性，用 CoreGraphics 框架把 CTRun 對象繪製到 Context 中。

理解還是不夠，等後續再來品讀。不覺感嘆 YY 實在太強了！今天理了理思路，讓自己邊寫邊讀代碼，不至於枯燥，同時供大家參考。得去睡覺了。