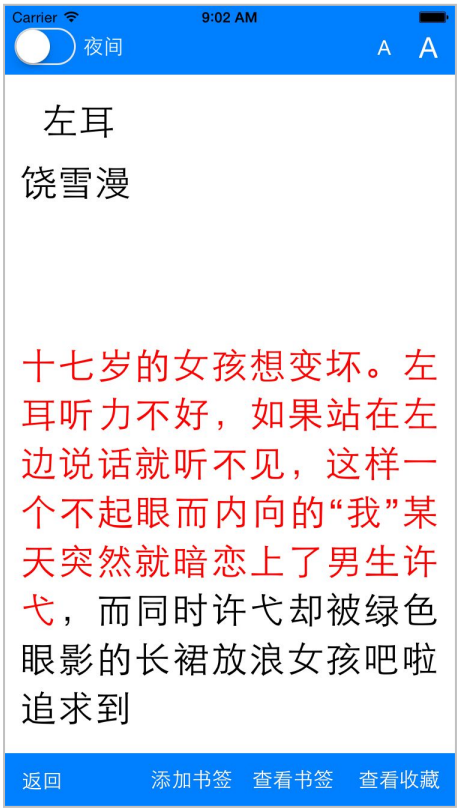
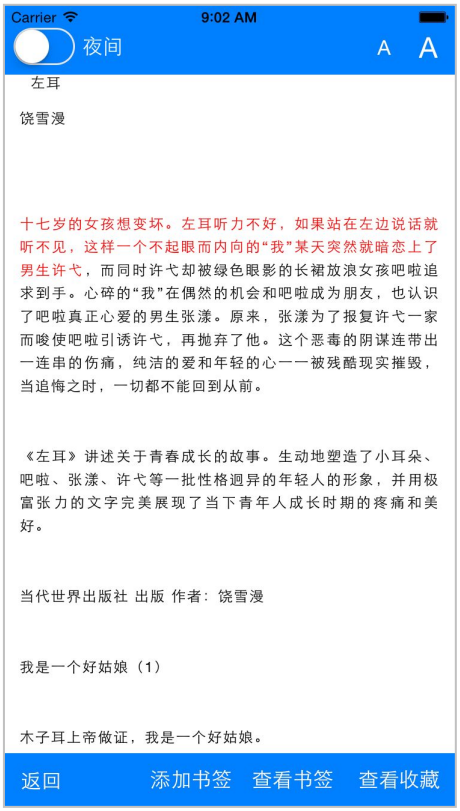
iDoc 书架 Beta

运行截图如下：

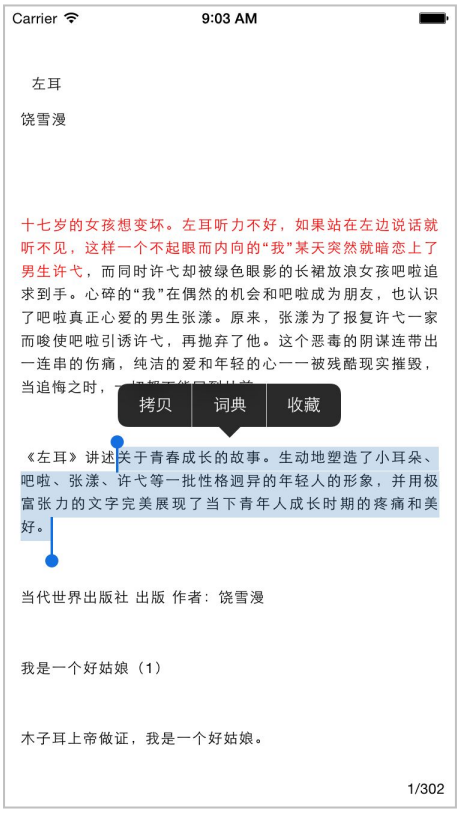
前后翻页功能比较简单，不作演示，这里展示添加书签的功能。点击下方工具栏“添加书签”按钮即可添加书签。

点击右上角“A A”图标即可调整字体大小。

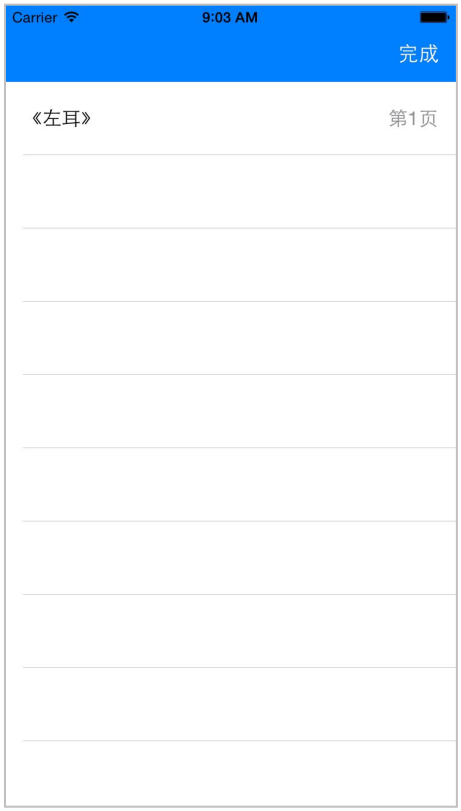




长按文字即可弹出收藏菜单，收藏过的文字会变成红色。上下部工具栏在敲击屏幕底部时会弹出，敲击屏幕中部时消失。

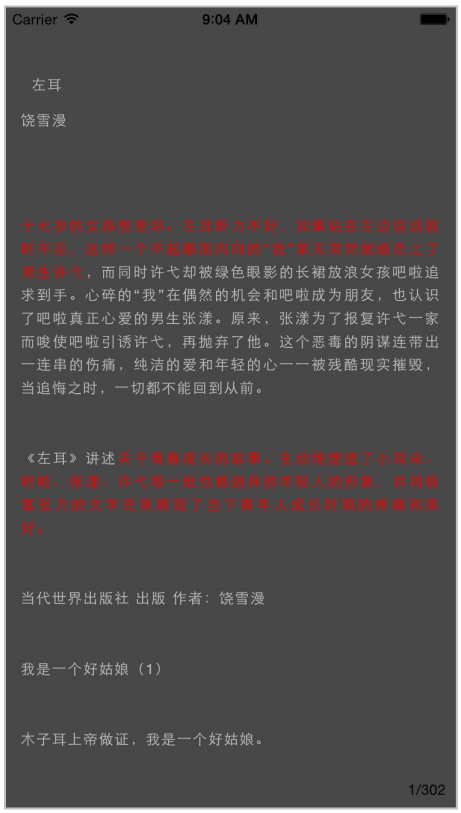


点击底部工具栏“查看书签”即可查看当前阅读的书籍已经添加的书签。在已添加的书签项目上向左滑动可以弹出删除菜单，点击已经添加的书签会跳转到书签指定页面。



点击底部工具栏“查看收藏”即可查看当前阅读的书籍中已经收藏的文字。在已经收藏文字上向左滑动可以弹出删除菜单。

点击左上角工具栏中的“夜间”模式开关可以切换页面颜色。



点击左下角工具栏中的“返回”按钮会返回到书架界面，并且保存当前阅读的书籍和位置。

下次打开时会载入返回时的位置。

已经实现的功能：

1.翻页阅读

2.改变字体大小

3.添加书签

4.保存阅读记录

5.长按划重点（收藏）

6.改变字体大小

已知但不影响体验的 bug：

1.收藏文字在调整字体跨页后不会再显示红色（这是文字算法的问题，待解决）。

2.收藏界面文字的显示偶尔不完整（概率很低）。

3.书签页面会出现多个同一页书签的情况。（这是正常情况，因为书签记录了不同字体下添加的书签，目前书签的跳转还没有十分完善）

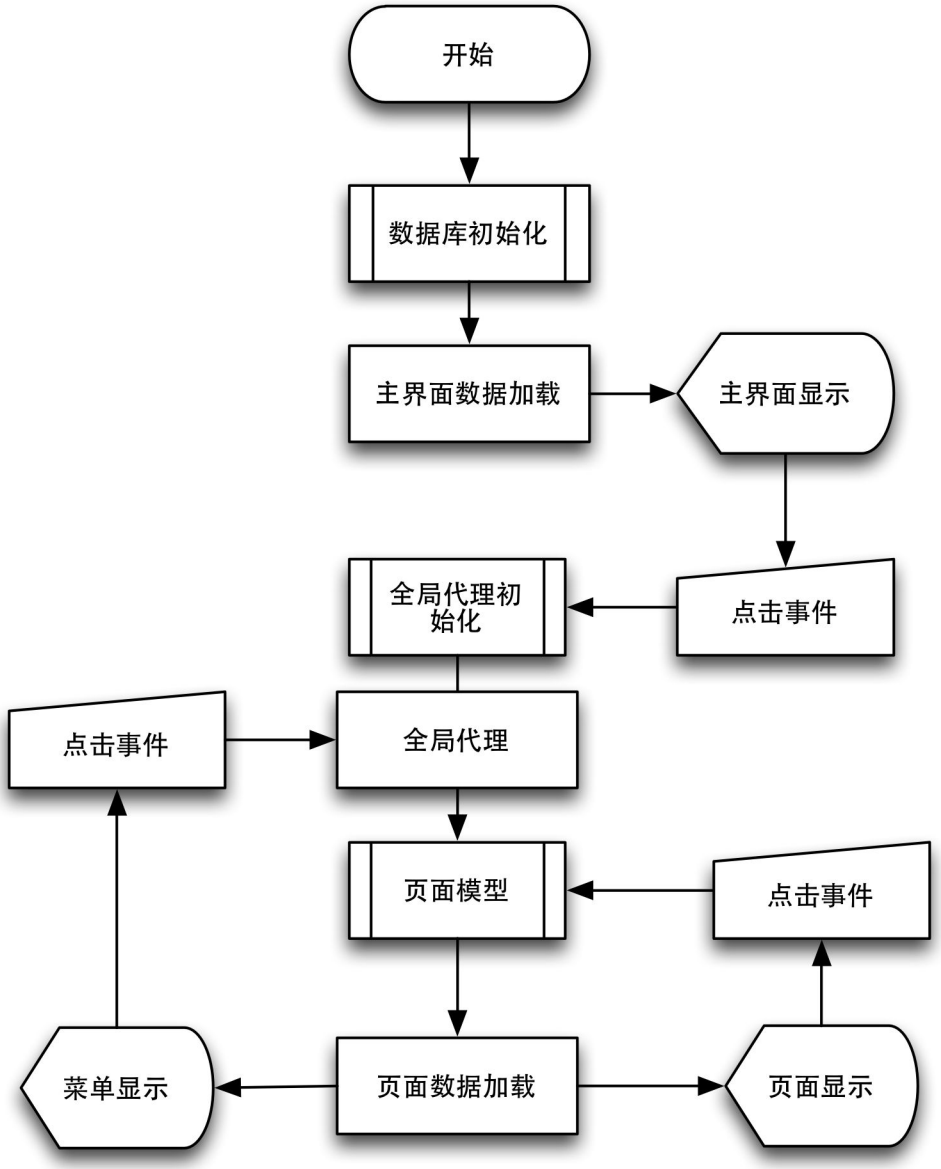
4.调整字体后可能会出现正在阅读的文字跑到前一页或后一页去（分页的算法不够完善的问题）

使用的第三方开源框架：

1.FMDB，轻量级的 SQLite 3 开发框架。

2.MBProgressHUD，添加操作提示

1.运行流程图



2.模型与结构

2.1 几个重要的概念模型：

2.1.1 数据库：

数据库是用来存储书籍有关的各种信息，在 iOS 平台上我选择使用 SQLite 存储数据。建立

了三个表：books，bookMark，reserved。分别用于存储书籍信息，书签信息，收藏信息。

2.1.2 全局代理模型：

全局代理用于管理当前阅读的书籍的所有信息，包括阅读的内容、页码，字体的大小，是否夜间模式等。全局代理能够根据设置更新阅读的页面。

2.1.3 页面模型：

页面模型用于向页面容器提供页面。它会根据全局模型传递过来的数据设置页面的数据，并且把页面提供给页面容器。

2.2 模型和结构

2.2.1 数据库结构（WZLDataUtils）：

Books 表结构

```
CREATE TABLE books (
    id integer PRIMARY KEY AUTOINCREMENT,
    name text,
    font integer,
    page integer,
    lastread text
);
```

bookMark 表结构

```
CREATE TABLE bookMark (
    id integer,
    name text(128),
    page integer(128) NOT NULL,
```

```
    PRIMARY KEY(id, page)
    UNIQUE (name, page)
);
```

Reserved 表结构

```
CREATE TABLE reserved (
    id integer PRIMARY KEY AUTOINCREMENT NOT NULL,
    name text(128),
    content text(128)
);
```

暴露的接口：

sharedDatautils，获得数据工具操作实例。

insertBooks（因为没有用到尚未实现）

getBookByName，通过书名获得书籍的信息字典。

getAllBooks，获得数据库中的全部书籍字典，返回一个数组。

updateBook，传入书籍信息字典更新书籍信息。

getAllBookMark，获得全部书签的信息字典，返回数组。

getBookMarkByName，通过书名获得书籍的所有书签，返回一个数组。

insertBookMark，插入一条书签记录。

deleteBookMark，删除一条指定的书签记录。

getAllReserves，获得全部已经收藏的信息字典，返回一个数组。

getReservedByName，根据书名获得书籍的所以收藏，返回一个数组。

insertReserves，插入一条收藏。

deleteReserves，删除一条指定的收藏。

2.2.2 全局模型（WZLGlobalModel）：

属性

Text，当前阅读的书籍内容

rangeArray，分页的结果

Attributes，字体属性

FontSize，字体大小

currentPage，当前阅读的页码

currentRange，当前阅读的

方法

sharedModel，获得全局模型的实例。

loadText，载入当前阅读书籍的内容，进行分页。

updateFontCompletion，更新字体，执行更新结束后的回调 Block。

updateNightMode，更新夜间模式，执行更新结束后的回调 Block。

2.2.3 页面模型（PageModelController）：

属性

pageData，当前页面的数据范围

Text，当前页面的数据

Attributes，当前页面数据文字的字体属性

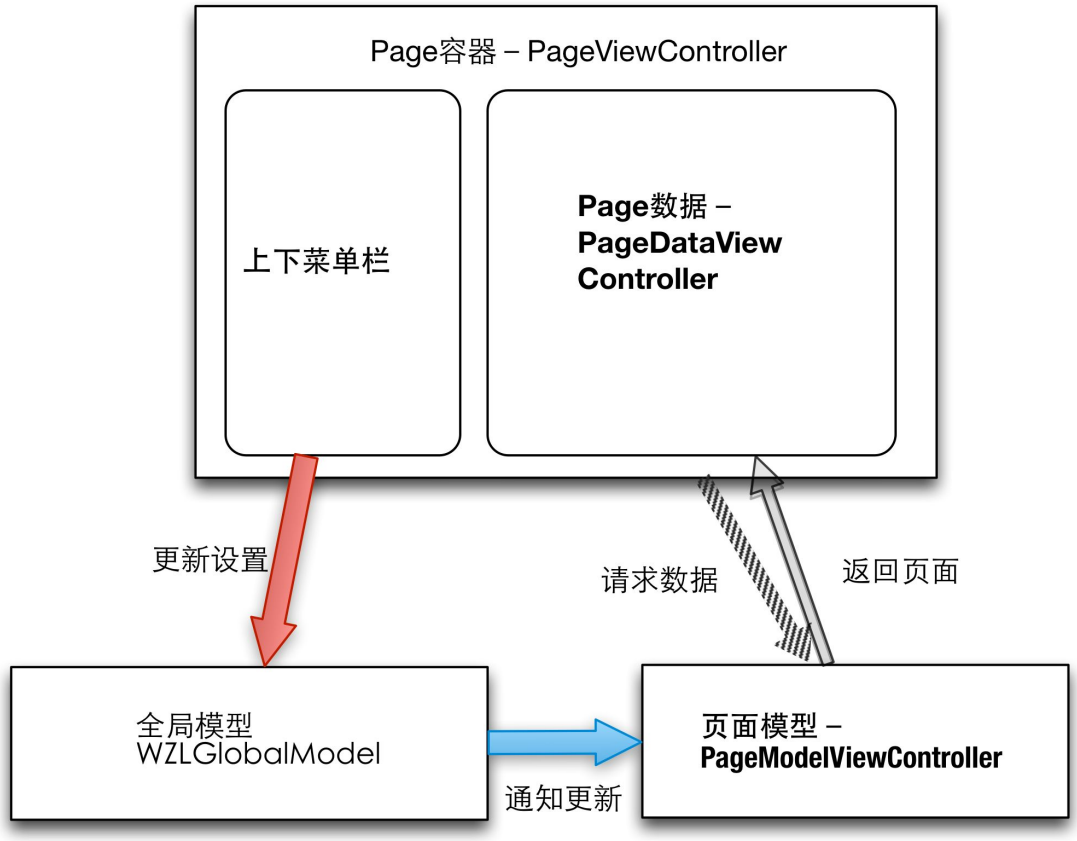
isNight，当前页面是否处于夜间模式

方法

viewControllerAtIndex，返回第 n 个页面。

indexOfViewController，返回页面的页码。

3.运行原理



当书籍被点击之后会跳转到 PageViewController 载入数据。此时，PageViewController 会向页面模型请求数据，页面模型根据全局模型的设置载入页面返回给 PageViewController。上下菜单栏的时间处理会向全局模型通知设置更新，全局模型通知页面模型更改设置重载页面，最后返回页面。

4.算法和实现

4.1 分页算法

4.1.1 CoreText 排版引擎

做电子书阅读器首先遇到的问题就是文字的排版，我需要计算一定数量的文字占用的宽高来确定每页有多少文字。最后把这些文字放进控件容器内用于显示。那么首先想到的两个控件

就是 UILabel 和 UITextView。但是流行的计算文字内容宽高的方式在 iOS7 以后已经被废弃了，所以 UILabel 和 UITextView 显得不太好用。但是，有了 CoreText 排版引擎就可以很好地计算文字的内容宽高。我选用了 UITextView，不选用 UILabel 的原因是 UITextView 对文字选中的支持性更好。

不过，我其实还可以用 UIWebView 来做文字的显示，但是使用 CoreText 效率会更好一点。

CoreText 是用于处理文字和字体的底层技术。它直接和 Core Graphics（又被称为 Quartz）打交道。Quartz 是一个 2D 图形渲染引擎，能够处理 OSX 和 iOS 中的图形显示。

Quartz 能够直接处理字体（font）和字形（glyphs），将文字渲染到界面上，它是基础库中唯一能够处理字形的模块。因此，CoreText 为了排版，需要将显示的文本内容、位置、字体、字形直接传递给 Quartz。相比其它 UI 组件，由于 CoreText 直接和 Quartz 来交互，所以它具有高速的排版效果。

CoreText 和 UIWebView 相比有以下的一些优势：

CoreText 占用的内存更少，渲染速度快，UIWebView 占用的内存更多，渲染速度慢。

CoreText 在渲染界面前就可以精确地获得显示内容的高度（只要有了 CTFrame 即可），而 UIWebView 只有渲染出内容后，才能获得内容的高度（而且还需要用 javascript 代码来获取）

CoreText 的 CTFrame 可以在后台线程渲染，UIWebView 的内容只能在主线程（UI 线程）渲染。

基于 CoreText 可以做更好的原生交互效果，交互效果可以更细腻。而 UIWebView 的交互效果都是用 javascript 来实现的，在交互效果上会有一些卡顿存在。例如，在 UIWebView 下，一个简单的按钮按下效果，都无法做到原生按钮的即时和细腻的按下效果。

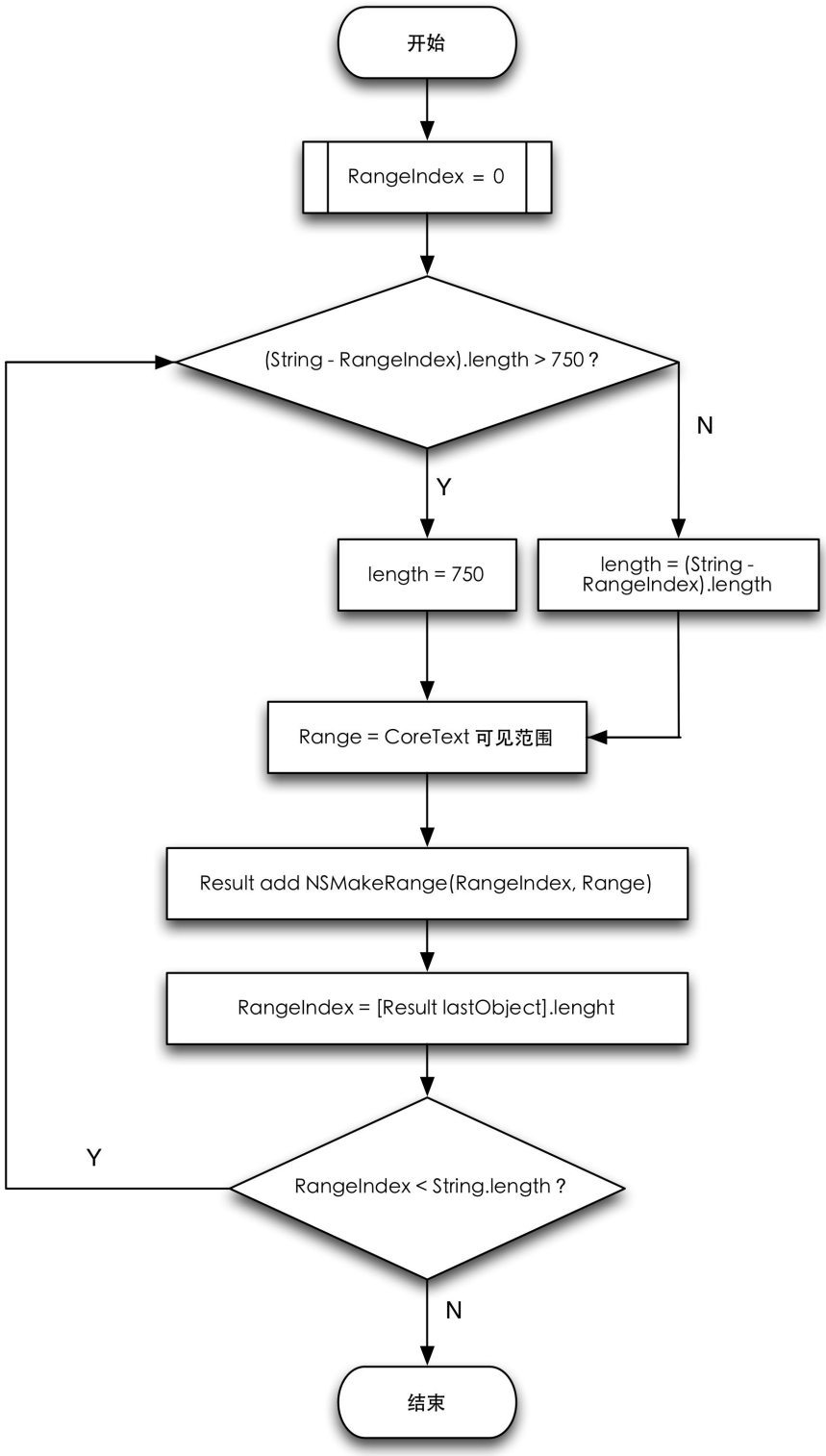
4.1.2 算法思想

基本思想就是使用 CoreText 计算一个 CGSize 的区域可以容纳指定属性文字的数量，由此来确定每一页的文字范围。

4.1.3 算法实现

算法实现的具体代码在 **NSString + WZLPaging** 的 Category 中。

文字描述为：从全部文字中截取一段长约 750 个字符（或更少）的属性文字。使用 CoreText 的 CTFrame，设置其 Frame，内容属性字符串，然后获取 Frame 中可见的文字范围（Range）。返回这个 Range，继续下一次计算。



5.代理模式

5.1 定义代理

字体的调节使用了一个代理来实现。代理主要是当前 ViewController 不方便管理而产生的一个运行模式，其实本质就是一个 Protocol。FontAdjustDelegate 中只定义了一个 optional 的方法。

5.2 使用代理

当前类中需要定义一个 FontAdjustDelegate 对象，使当前类继承 FontAdjustDelegate 代理，实现方法。最后设置 self.delegate = self 即可。

5.3 代理响应

使用 self.delegate respondsToSelector 方法来响应代理，执行代理方法。更新字体的代理中，使用了 GlobalModel 的 updateFontCompletion 方法，在字体更新后的 Block 里面重载页面。