

**P R O  
D U C  
T E R**

**K E V I N   Z H O W**

# Table of Contents

---

1. Introduction
2. Product 设计驱动开发
3. Product Design 设计的感觉
  - i. 设计的感觉 - 从感觉开始
  - ii. 设计的感觉 - Clarity
  - iii. 设计的感觉 - Clean
  - iv. 设计的感觉 - Cool
  - v. 设计的感觉 - 寻找灵感
4. Product Design 产品的起点
  - i. 产品的起点 - 为什么做 小记
  - ii. 产品的起点 - 设计工具的选择
  - iii. 产品的起点 - Sketch 基础
  - iv. 产品的起点 - Framer 基础
  - v. 产品的起点 - 设计 小记
  - vi. 产品的起点 - 交互设计
  - vii. 产品的起点 - 小记 的交互设计
  - viii. 产品的起点 - Hydro 的交互与体验设计
5. Product Code 产品的实现
  - i. 产品的实现 - 成为开发者
  - ii. 产品的实现 - Swift
  - iii. 产品的实现 - iOS App 是什么
  - iv. 产品的实现 - 在年之外
  - v. 产品的实现 - 使用 Git 管理你的代码
  - vi. 产品的实现 - 年 和 月
  - vii. 产品的实现 - 撰写
  - viii. 产品的实现 - 浏览年, 月
  - ix. 产品的实现 - 浏览日记
  - x. 产品的实现 - 编辑日记
6. Product ReDesign 产品的迭代
  - i. 产品的迭代 - 优化
  - ii. 产品的迭代 - 动画库 Facebook POP
  - iii. 产品的迭代 - CocoaPods
  - iv. 产品的迭代 - Crashlytics
  - v. 产品的迭代 - 更通用

7. Product Market 产品的营销

- i. 产品的营销 - 产品的特质
- ii. 产品的营销 - 营销的方法
- iii. 产品的营销 - 营销的细节
- iv. 产品的营销 - 内测你的产品

8. Product Extra 动效

- i. Product Extra 动效 - Motion Graphic
- ii. Product Extra 动效 - Waver 声波效果

9. Product Next 新篇章

# 前言

---

从什么时候开始，设计变成了我生命中的一部分？

似乎开始于一个非常难以捕捉的时间；

或许是我第一次打开 Vimeo 的时候；

或许是第一次用镜头记录风景的时候；

或许是我第一次想要用情书感动一个女孩的时候；

记忆的丢失让人分外伤怀，但是与产品一起度过的每一刻，我都满怀着对美好的期待与感恩。

独立完成一款产品是很多人得梦想，也是我的梦想，但是单单完成一个作品并没有什么价值，真正完成一款优秀的作品的难度远超过了“完成”这个词语。

设计上的锤炼，编程技艺的精进，营销的思考，让很多人都在起步阶段就放弃了。

从来没有一本书，去讲如何完整的完成这件事情。

我觉得应该有一本。

于是断断续续，经历了一年的时间，我终于在 23 岁生日前写完了这本书。

完成一件事情着实考验人的耐心，这本书初稿之后又经历了相当大的修订，这种改进和迭代与产品开发何其相似，正如许多创作一样，想要下笔如神是非常困难的，甚至很多老手也不见得可以在第一次就做的比新人要好。真正使得作品脱颖而出的，正是后期对作品的反复打磨与改进。

在修订过程中，常常会跟朋友吐槽说写书就像绣花一样，尤其是因为最近几年荒废了写作，竟然遇到了很多语法问题（看英语文档看多了）。

非常感谢 周奕飞 (@austinchou0126) 以及 朱宏旭 (@nixzhu) 对于排版和语法上的修正。

对了——

这本书只是一个开始，和产品一起成长，做出越来越优秀的东西才是最大的快乐。

## 本书的读者

---

这本书主要涵盖了

- 设计原理
- 原型，动效，交互设计
- iOS 开发
- 营销

这四个部分，以 小记 作为例子，完整的还原了一款产品从想法到编码，再到上架，以及营销的全过程。如果你什么都不懂，面对未来正充满困惑，那么这是一本很好的入门书籍，如果你只知其一二，那么这本书也是很好的补全指引。

不过，由于产品的世界不存在只要 A 必然发生 B 的真理，所以请带着怀疑的眼光去审视每条结论，学会如何思考才能淡定得面对这个不测的世界。

## 解疑

---

本书发布后会持续迭代更新，这也是电子书发行的好处。如果你对本书有任何意见，欢迎写 email 给我

[kevinchou.c@gmail.com](mailto:kevinchou.c@gmail.com)

或直接关注我的微博 [@周楷雯Kevin](#) 与我交流。

# Product: 设计驱动开发

---

## 关于设计的第一次

---

第一次总是很特殊的体验，而对于设计来说，第一次的体验可能是很尴尬的。

有可能是在 Photoshop 里拖拽了几个难看的方框，也可能在草稿纸上画了一堆歪扭得不成样子的曲线。反反复复，最后只能感叹一句——“算了吧，我果然没什么天赋！”。

但是，设计并不是看起来那样，坐在桌子前凭空臆想，然后动动鼠标就可以有所产出，也不是一件与在桌子前呆得有多久有关系的活动。设计师，往往要以年为单位，把自己浸泡在优秀的设计里，增强自己的设计嗅觉，才最终能够做出优秀的作品。

设计总是从模仿开始，在我的记忆里，第一次做设计是在 Ubuntu 7.04 上用 GIMP 制作壁纸，模仿 Mac 的霞光壁纸，那还是 2007 年的事。

完成后非常地开心，因为第一次我的桌面铺上了我自己做的壁纸。而且我没有用 PS 这种“恶俗”的软件，没有在 Windows 和 OS X 这种“邪恶”轴心系统上，我内心洋溢着一种人类解放的革命情怀。

## 为什么先做设计

---

很多人以为我是从编程开始的，其实设计是一个走在我做编程之前的事情，但是我真的尝试过先开始编程。

大概是在 2004 年，我小学六年级的时候，由于初生牛犊不怕虎，我进去书店就买了本 C# 的书准备回家编程了。但估计是智商的问题，我完全看不懂那些术语。大概就在第一章，只不过 10 页就开始讲 NameSpace（命名空间）。我当时就蒙了，被这个陌生的名词搞得头脑发热，完全死机，什么是命名空间，完全不能从字面理解这个意思。

我因此觉得自己是个傻X，怎么刚翻开两页就看不懂了？

然后感叹道：“天才如我竟然看不懂编程书！编程真是高深！”。

当时家里没有网络，处于一个完全单机的环境，靠着《少年电脑世界》、《中学生电

脑》、《电脑报》、《电脑爱好者》这些的杂志来积累计算机方面的知识。我家乡的小县城，周围没有一个人从事这个行业，所以我以为每个学编程的人都是直接看这种书并且一次就能看明白而且学会的，因此我遭受了巨大的挫败感。

那本书的前几十页差不多被我翻烂了，用红水笔反复标注了我对每行代码的理解，但最后我还是没有弄懂，只能放弃这个事情。

因此，在尝到了壁纸的甜头后，我开始尝试更多的设计，并且买了原研哉的《设计中的设计》。重视留白且简约的设计对我造成了深刻的影响。

## 设计是什么

---

设计是一种和谐的感觉。

因恰如其分而结合在一起的东西，功能与外观之间达成完美妥协。

我想设计必然是有个人风格的，这种风格会成为产品的灵魂，最终交付到用户手中，让他们去感知。

但是如何让这种个人风格和产品的功能相得益彰，是每个设计师在开始设计之前要最先思考的问题。

我非常喜欢设计驱动开发。先设计，后开发，然后回过头来再设计，再开发。

在这其中，自己和自己能最快速的交流，最快速的反馈，这种方式可以很好的激励你自己，并且让自己很好的把握进度。

但这样做也最容易产生自我妥协，人会犯懒，这是没办法的，因此需要不断激励才能把某些事情做得完美。

在开源社区有两个著名的设计驱动开发的例子，一个是 elementary OS，另外一个是 GNOME。他们是出了名的三年前画好图，三年后只能实现画图的一半。

## 实践

---

不论方法是什么，实践才是能真正提供经验的活动，有经验值才能学会这件事情。

实践同时也意味着另外一件事情 —— 失败。

我们总会逐渐避开不愉快的体验，失败犹甚。无论是学习的失败、交友的失败还是和女孩子交往的失败都可能以一句“我果然还是不适合”为变成逃避这件事情的理由。

在一个人独立完成一款作品的过程中，你也会经历很多失败：第一次设计的丑陋，代码的漏洞百出，放出不成熟的产品导致的恶评等等，它们都会给你残酷的打击。

但是从失败中反思而得到的经验才真正宝贵。更何况，失败得多了，总会成功的。

“毕竟对于生活本身来说，成功和失败并没有什么本质区别，重要的是参与到这个游戏中来。”—— 约瑟夫·休格曼

# Product Design: 设计的感觉 - 从感觉开始

学习一个新东西的方法有很多，一直以来我们被教育的方式都是从基础开始，成年累月的学习每块瓷砖，然后再去组成模块，最终搭建起自己的知识体系。

这种方式最明显的好处是本身基础的夯实，而弊端却是毁灭性的。

枯燥乏味的过程，脑中的知识无处可用的沮丧感，时刻伴随着学习的每一天。而最终，我们失去了兴趣，选择放弃。

对于设计而言，尤其如此。

每一个设计的准则，都源自于对人心理的研究。想要从事设计，你就要懂得人们如何感知、学习、推理、记忆。

但是，为什么我们要花费四五年的时间，从配色，结构，心理学入手？

谢天谢地，我们完全可以反过来，不从理论着手，先看一看有趣的设计是什么样子的，然后再讨论一下，这些设计都遵循了什么样的设计理念，利用这些感觉，我们尝试去做自己的设计。

毕竟，保护好转瞬即逝的兴趣才是最重要的。

但是！别高兴得太早！

不要忽略对基础理论的学习，当你的设计完成之时，你会找到很多设计的缺点，以及自己知识的短板。这时候回去看那些枯燥无味的理论书，你就能体会到如沐甘霖的快感。

在开始之前，和你分享一句被许多有为人士说过的话：

"If you want to please everybody, you please nobody."

如果你想要取悦所有人，那么你不会取悦任何人。

设计师的职业和艺术家有很多不同，注定不能像艺术家那样特立独行。设计并不是把个人品味强加于产品之中，伴随着你对设计理解的深入，终究，“设计为谁而做”这个问题会成为每个设计开始时，你第一个思考的问题。

# Product Design: 设计的感觉 - Clarity

## Clarity

Clarity 是清晰的感觉。

从 iOS 7 开始，Apple 正式把这种感觉引入了系统设计。

- Simplify 简化
- Maximize content 内容为主
- Depth 纵深

## iOS 7 天气 App 的 Clarity



iOS 7 的天气 App 是对这种感觉的最好阐释，雪天的背景占据了整个屏幕，没有了以往常见的界面边界，视界变的极为开阔。

因为去除了边界，内容得以占据更多的空间。但是此时并不是没有了边界，从视觉的角度来讲，此时的天气 App，中间的大字号气温就成了中心，其他区域都是边界。

人眼的边界视力比中心差得多，你能轻易的分辨 29，但第一眼却很难注意到下面的气温都是些什么。

把次级消息放在边界就成了更好的选择。

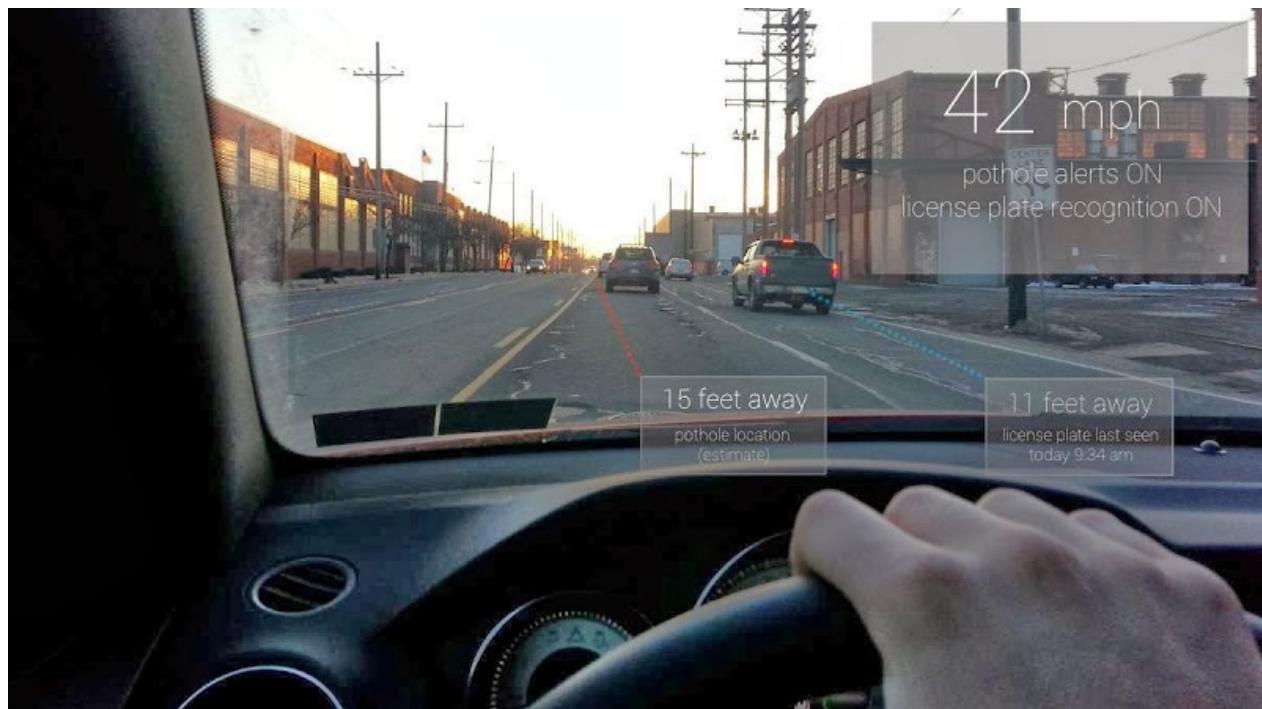
## 《纸牌屋》里的 Clarity

这几天在看《纸牌屋》的时候，我发现纸牌屋 OP 的设计也很有这种 Clarity 的感觉。



文字在屏幕的位置清晰自然，浑然一体。

## Google Glass 里的 Clarity



## 《钢铁侠》里的 Clarity



Clarity 这种感觉，是利用人眼对边界信息感知差的特点，将次级消息放在了边缘。将其应用于对现实场景的信息化中，能很大程度上减少信息和背景之间的干扰。

要达到这种效果，就需要你对信息位置的放置，背景色彩的过渡，以及现实场景有个妥善设计的感觉 - Clarity

的处理。

Google Glass 在文字下方加了个半透明的灰色背景，来让你很好的聚焦边界信息。除了半透明的处理方法，如果信息块不高，区域统一，我们也可以用透明渐变来做这种底衬。这种处理你可以从钢铁侠的图片中感受到。

# Product Design: 设计的感觉 - Clean

## Clean

它延续了 Simplify (简化) , Maximize content (内容为主) 的设计理念，是建立在合理性之上的美学 (Aesthetic) 协调。

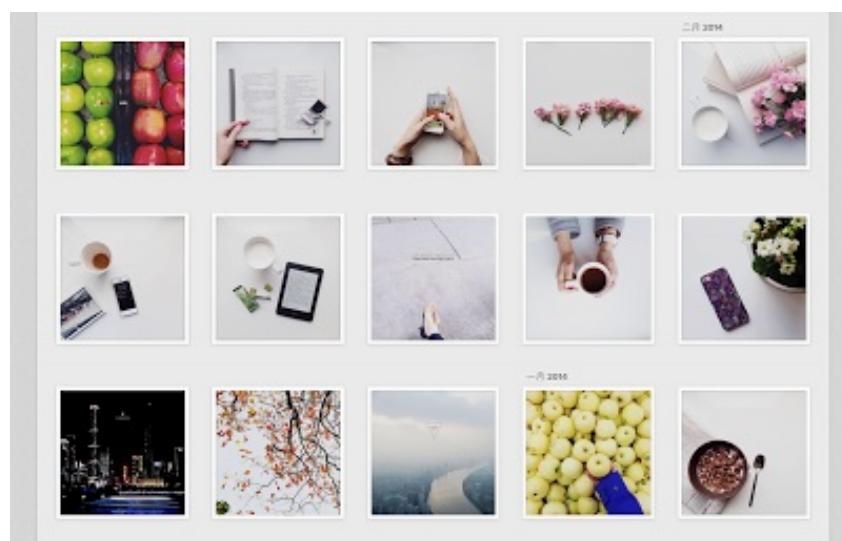
Clean 作为一种极简主义，更能让高端用户喜欢。

Less is more

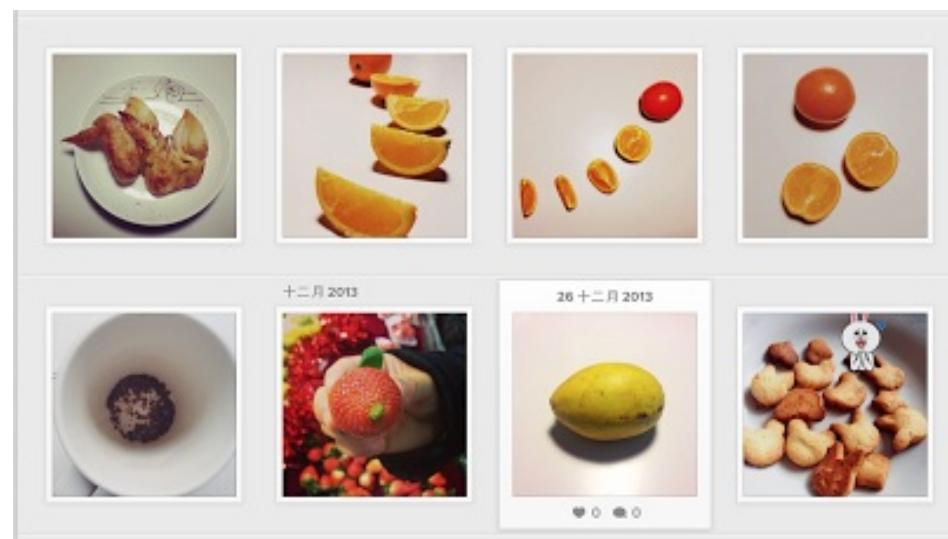
by Ludwig Mies van der Rohe

## 留白

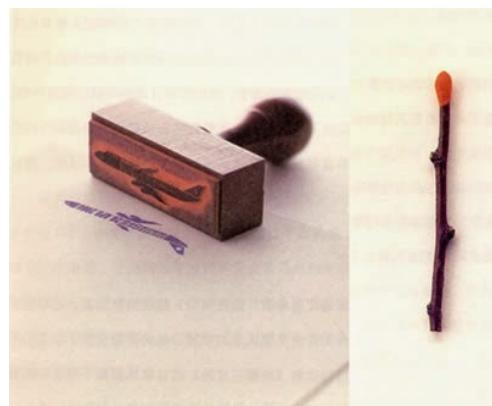
一次偶然的机会，我在 Instagram 上关注了一个非常善于采用过曝做到 Clean 感觉的女神：hx1125。如果她愿意销售其作品的副本的话，我十分愿意买下其中那张她捧着 iPhone 的照片：



后来我学着去捕捉这种感觉，不过很明显结果是我离的不够远。因为是晚上拍摄的，在光线上也有诸多问题。



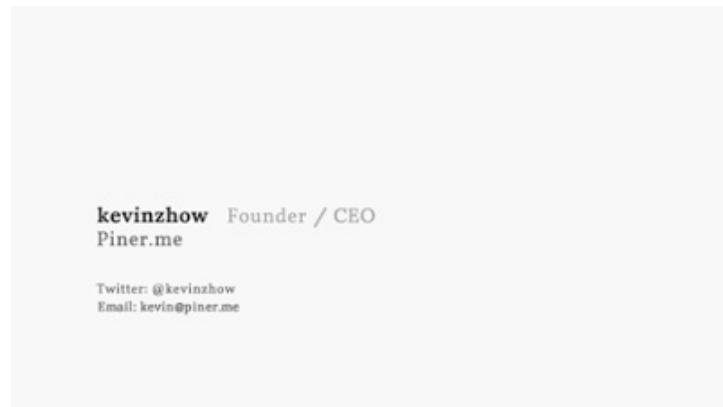
我非常喜欢她那些在白色的背景上只摆着一两件东西的感觉，这种满足感和多年前翻开原研哉先生的《设计中的设计》时何其相似。



如此慷慨的把画面让给了需要我们去注意的信息上。我想 Clean 这种设计语言的灵魂也正是如此 —— 少即是多。

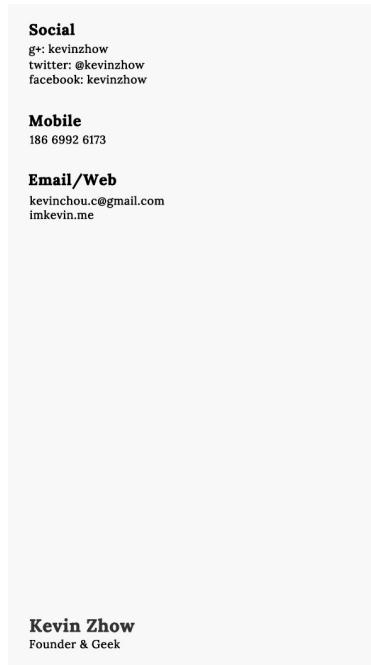
在我刚做设计的时候，潜意识总会驱使我去填满整个画面，那种感觉就是 —— 没塞满就是没设计过。后来，随着我做的设计越来越多，我开始更多的关注画面的比例，协调性，不再去想如何塞满整个画面，而开始去想怎么让画面更少一些元素，最好是选择一款合适的字体，一个合适的字号，一个合适的颜色，只有文字，那真是太美了。

大概在一年前，我设计过一个从未拿来用的名片。没使用的原因是因为这个设计需要非常有质感的纸张，以及非常精巧的印痕才能体现它的美感，但是我不知道去哪里能做到这种效果，只好一直封存在我的 Dropbox 里。



没有手机号，没有二维码，没有地址，只有 Twitter 和 Email，没办法，我觉得这样最纯粹，可能和我那个没有消失殆尽的不群之心有关。

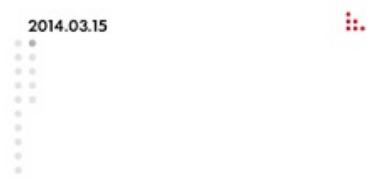
后来我尝试了一个竖版的名片设计



衬线字体的优雅和稳重似乎始终是我在文字排版中的挚爱。

## Clean 的 App 设计

ifanr 有一款非常非常 Clean 的 App —— [数读](#)，我对这款 App 真是一见钟情。



# 2.8亿

我国有 2.5 亿居民的住宅区靠近重点排污企业和交通干道，2.8 亿居民使用不安全饮用水。



环保部昨日发布了首个“中国人群环境暴露行为模式研究”大规模研究结果。

后来，在我的一款 App [Coinsman](#) 中，我也运用了这种设计方式。



这两款 App 都是强信息驱动的设计方式，数读中最重要就是那个点睛的数字，在 [Coinsman](#) 中最重要的就是当前的价格。

把他们用最大字号放在中间无疑是最好的选择，其他一切信息都不要来抢这个关键信息的聚焦区间。

数读的中央数字下面的点睛文字用了远小于主体的字号，而 [Coinsman](#) 在最高和最低这两

个价格上，也是用了小字号，而我觉得还不够，降低了他们的不透明度，最后选定在 60% 左右。

## 元素的主次关系

---

在数读中，下面有一段灰色的文字，是全文的开头，点击这里并往上滑动可以看到全文信息，全文的文字颜色也会增强为黑色。在 Coinsman 中，下面的五分钟交易量柱形图也是明显暗于价格波动五分线的。而所有元素都小于屏幕中间的数字。

这就是元素的主次关系，也是设计非常非常重要的一点 —— 永远不要让不同级的东西看起来一样，不要让同级但不同时的东西看起来一样。

如果发生了这样的“永远不”，那么这个界面看起来就是一场灾难。

## 总结 Clean

---

- 留白来帮助聚焦最重要的信息
- 关键色的谨慎选择
- 拥有唯一的主体

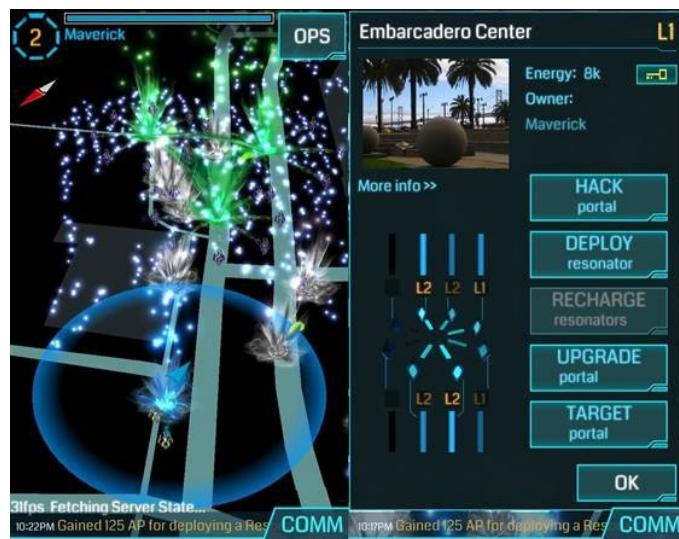
# Product Design: 设计的感觉 - Cool

## Cool

Cool 似乎是找不到形同词时候的唯一选择，Cool 并不是简单的好看有趣，而是一个准确的视觉体验。对于 App 自身而言，Cool 代表一种生命力，而 Depth（纵深）这种理念也始终贯穿着。

## Ingress

[Ingress](#) 是 Google 2012 年上线的一款 LBS 多人游戏，当时我被这个游戏的玩法深深感动了，一向视宅若命的我都不顾夏日的炎热跑出去玩这个游戏。



Ingress 这种虚拟现实的感觉会让人一下子被迷住，地图已经不是我们导航概念里的那种呆板的形象，而变成了一种要去拯救世界的场景。这种风格超过了你对任何一个 LBS 游戏的预期。

而地图中

- 无处不在的粒子特效
- 光晕
- 极强的明暗对比

都让其产生了杀马特般的冲击力。

即使你对出门不感兴趣，但是还是想去探索下这个超酷的软件。那就是 Cool 激发了用户的好奇心。

## Smash Hit

---



前段时间 App Store 推荐的这款 [Smash Hit](#) 真应该算是 Cool 的 Best Practice。它通过

- 飘渺的虚拟感
- 神秘朦胧的场景
- 精致的物理特效

给你打造了一个可以释放压力的空间。虚拟空间更容易让用户产生代入感，按照你的设定，寻着你的逻辑，变成一个中二病。

## 纪念碑谷——场景互动的极致

---



场景互动一直是游戏的命根，从早先《仙剑奇侠传》的迷宫，《轩辕剑》里的机关术，《塞尔达传说》里人物与场景的物理互动，再到底现在的场景破坏。场景的真实反馈给了用户无限的刺激。

纪念碑谷各个关卡的设定都非常有趣，总是用你想象不到的方式将你带入下一个场景，巧妙利用视差、矛盾空间去构建新的路径。

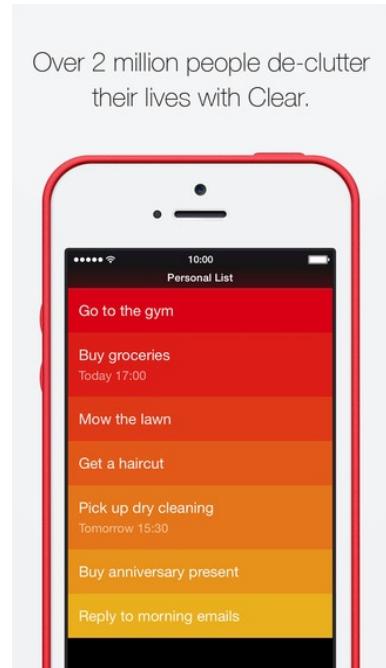
## 总结 Cool

---

1. 强烈的视觉风格
2. 新奇的氛围体验
3. 生动的交互反馈

Cool 的特质如果应用到软件中，那么就是

1. 新奇的交互效果
2. 让人惊喜的视觉反馈
3. 强烈的设计风格



App 的典型代表是 [Clear](#)，适当的加入 Cool 的这些元素构成，必然让我们的 App 富有生机。

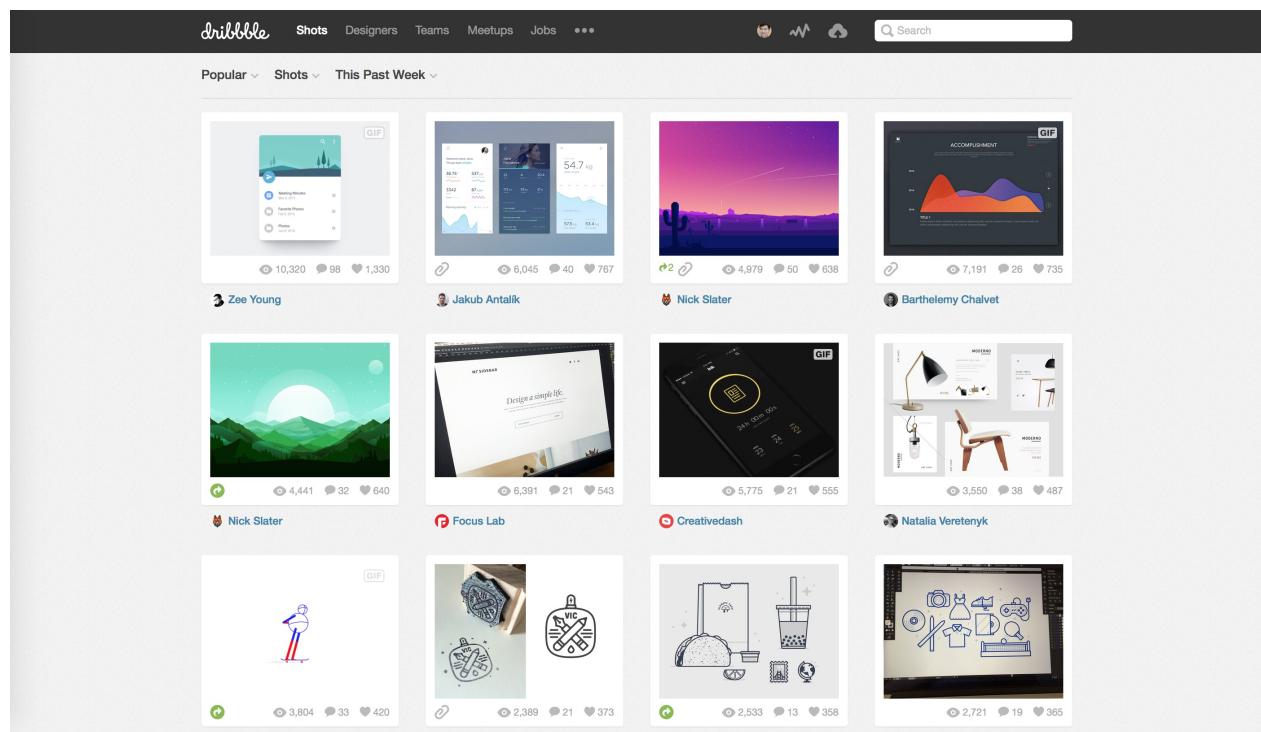
# Product Design: 设计的感觉 - 寻找灵感

对设计的学习来说，除了基本的理论，积累大量的素材更为重要。

“当你手里有个锤子的时候，你看什么都是钉子。”这句话正是在说方法单一的后果。当你积累了大量的设计语言，以及元素的组合方式后，你手里就不再只有一把锤子。融合你生活中对于美好设计的体验，属于你自己的优秀设计也便会自然诞生。

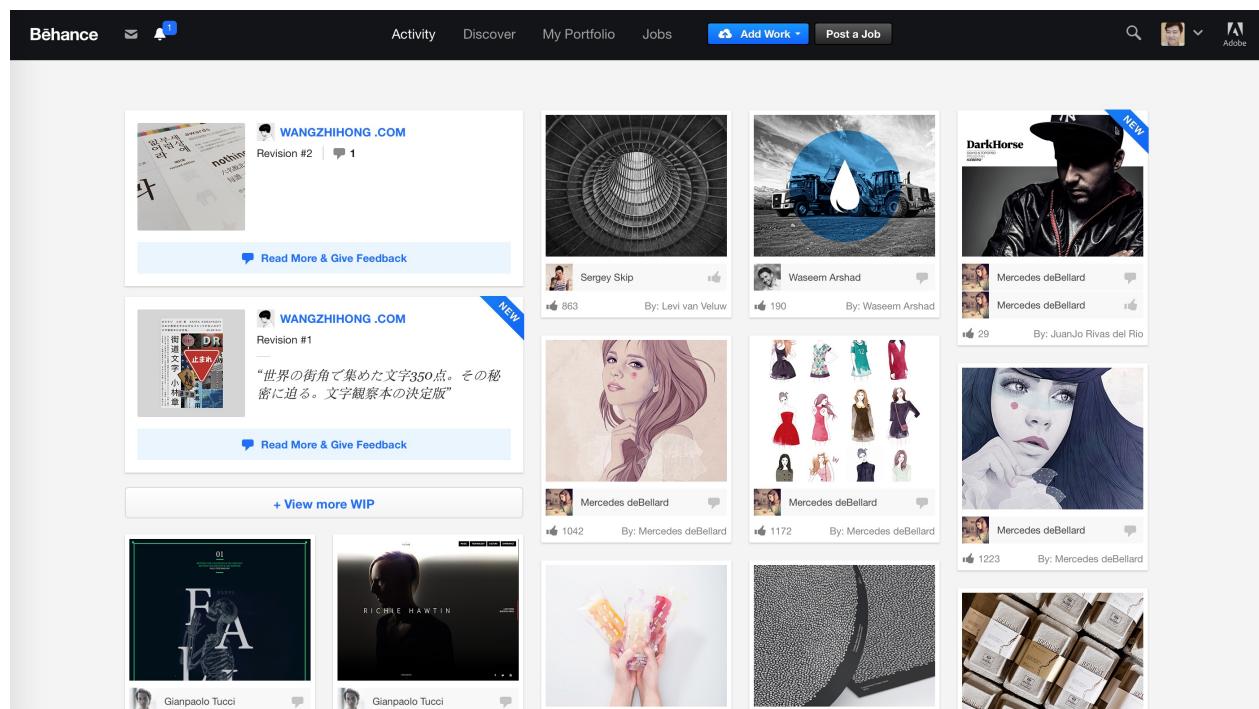
## Dribbble

[Dribbble](#) 是在设计师中非常流行的一个分享设计的社区，每次打开都会有许多让人惊叹的设计，不过惊叹之余，你也可以对这些设计进行更深入的思考——当这些设计应用到真实环境中，是否真的能增加体验。



## Behance

[Behance](#) 这个社区更为专业，已经被 Adobe 收购。它里面的作品完成度都相当高，覆盖的范围也更加广泛。

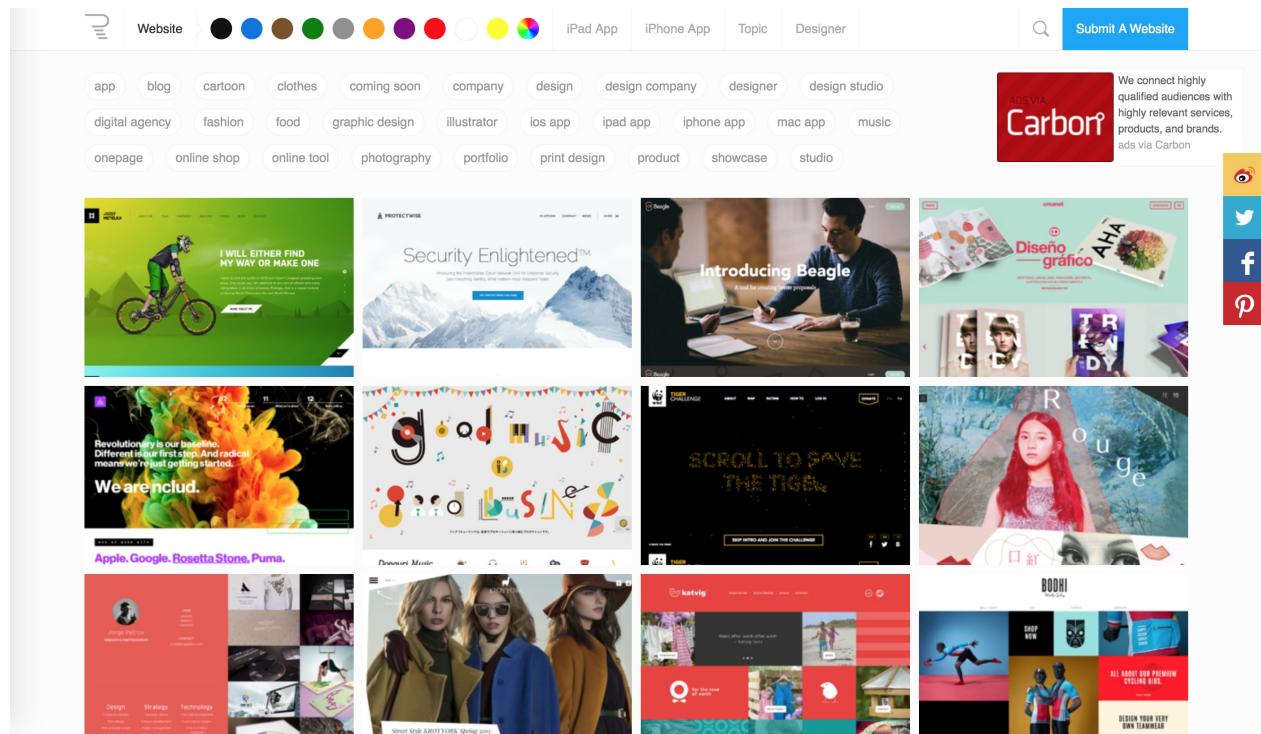


## Siiimple

这是一个专门收集极简主义网站设计的网站，筛选的作品都很独特并且具有启发性。

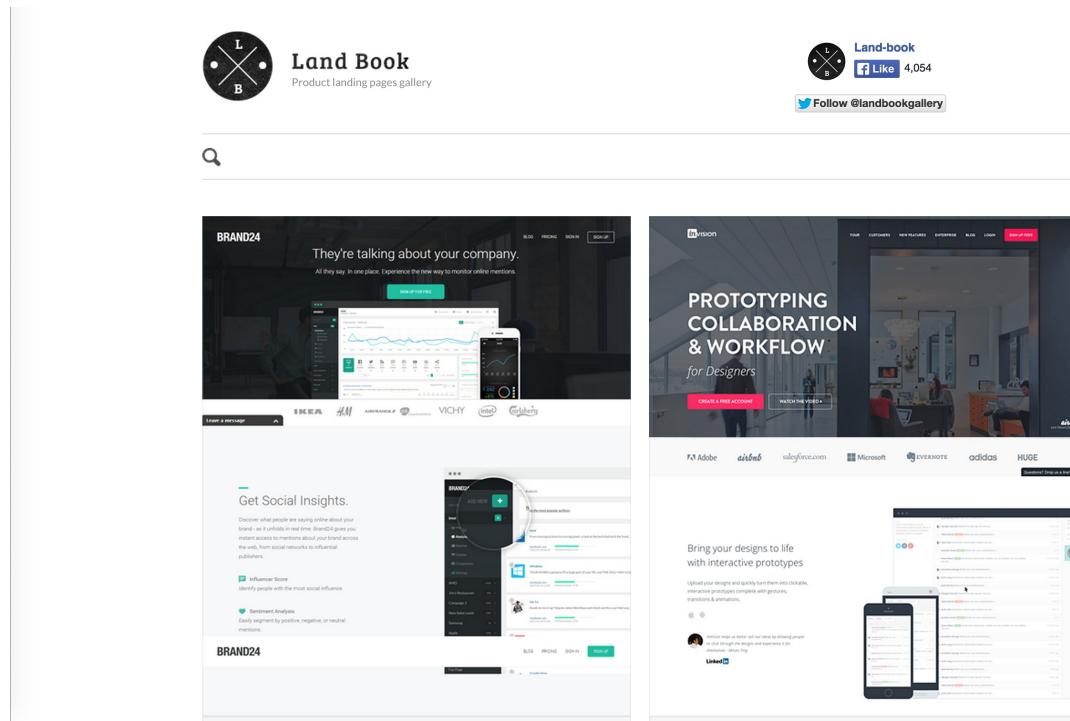
## Reeoo

Reeoo 也是一个专门收集优秀网页设计网站，提供了详细的分类目录，从 App，艺术，卡通到摄影，体育一应俱全，还可以根据颜色的主题筛选，非常好用。



## Land Look

Landlook 专门收集了优秀的 landing page 页面，品类齐全，值得时不时地看一看。



# Call to idea

Call to idea 是一个以类别为线索收集各种设计的网站，专业，精美，是激发灵感的好地方。

COMPONENTS

Logins 32 items	Registers 16 items	PopUp's 27 items	Tabs 54 items	Footers 27 items	Empty states 19 items	Calendars 15 items
Mailings 16 items	Lists 34 items	Galleries 17 items	Forms 16 items	Sliders 41 items	Stats 56 items	Testimonies 18 items
Details 27 items	Pricing 30 items	Players 30 items	Themes 30 items	Menus 44 items	Profile 21 items	Wizards 41 items

[www.calltoidea.com/registers.php](http://www.calltoidea.com/registers.php)

把同类别的优秀设计统观一遍，你就能发现很多共通之处。

The collage shows five different registration forms:

- CALL to IDEA** - A green-themed form with social login options (Facebook, Twitter, LinkedIn) and a "Sign Up!" button.
- APPLY NOW** - A teal-themed form for drivers, featuring a "BECOME A DRIVER" button.
- Formulario de registro** - A Spanish-language registration form with fields for Name, Email, Password, and Birthdate.
- Register!** - A white-themed form with sections for Personal Info (Name, Surname, City, Password) and Additional Information (Hobbies).
- Let's get started.** - A light blue-themed form with fields for First name, Last name, Email address, and a "Create account" button.

# Designer News

[Designer News](#) 是全世界最热闹的设计师的资讯社区，新的设计想法，新的产品，都会在这里进行讨论。

The screenshot shows the Designer News website. At the top, there's a dark header bar with the site's logo on the left, followed by navigation links for 'Stories' and 'Jobs', and buttons for 'Subscribe' and 'Sign in / Register'. Below the header, on the left, is a sidebar with a 'Top stories' section containing links to 'Recent stories', 'Discussions', 'Search', and 'Blog'. The main content area displays a list of eight news items, each with a small circular icon representing its category (e.g., a camera for LayerVault, a smartphone for New Spotify feature). The stories are: 1. 'Congratulations @awilkinson on buying Designer News!' (LayerVault), 2. 'Designer News 2.0 (medium.com)', 3. 'New Spotify Running Feature (youtube.com)', 4. 'The Sliding Scale of Giving a Fuck (blog.capwatkins.com)', 5. 'Show DN: Facebook's Little Red Book (officeofbenbarry.com)', 6. 'Amazing new landing for Pixate (pixate.com)', 7. 'The End of Global CSS (medium.com)', and 8. 'Apple plans to refresh iOS 9, OS X 10.11 using new Apple Watch font (9to5mac.com)'. Each story includes a point count, comment count, and a timestamp.

## 大脑的后台运作

创造性的活动并不是单纯的将时间耗在上面就可以完成出优秀的结果。大脑的运算不同于电脑，潜意识的部分你无法控制，然而它的速度却远远超乎你的想象。

在大量收集，学习了多样的设计之后，不妨离开书桌，出去跑跑步，游个泳，完全忘记工作的事情。这时候你的潜意识会在后台归纳组织你的素材，这些素材可能来自于你半生的积累，数百万条记录在飞快的组织着，等你彻底放松回来后，最终的结果可能就是——“我是怎么想到这个点子的，我真是个天才。”

# Product Design: 产品的起点 - 为什么做 小记

---

每一个产品开始前都要想几个问题，这几个问题很有价值。

## 我为什么要做这款产品

---

这个问题将决定产品的完善度以及整体的表现。

小记 初衷极为简单，想法酝酿于一段极为清闲的时期。

当所有的事情都可以做成社交的时候，给自己的一片方寸之地尤为重要，清静，独处，这一刻方才会感受到世界与自己之间的牵绊。

## 我希望这个产品有什么感觉

---

这个问题将决定这款产品的表现形式，包括气质，设计，交互。

1. 复古，文字不在多，而在于书写时的自我陶醉
2. 独立，一个不受外界干扰的地方
3. 单纯，一个只有文字美的地方
4. 禅，我希望每次打开这个 App 的时候，我都会感到一个更宽广的世界

## 这款产品要给什么人用

---

小记 并不是一个追求每个人都适用的 App，但它确实又拥有每个人都可以使用的场景，如果我大喊“大家快来用这个记事吧！”，那么大家的感觉可能是“我有 Evernote 干嘛要用你的？”。

如果说“为写诗而生的 App”，那么喜欢写诗的朋友可能会十分感兴趣，甚至大众也会对这个小众事项兴致勃勃。

产品有意思的一点是，它不会因为你把目标人群定小就不会成为大众产品，往往小众东西的独特魅力反而会成为一种潮流，在开始把产品定位为 For every one 是一个危险的信号

——这个产品可能毫无吸引力。

## 做多少功能？

---

这个问题更有意思，因为很多产品喜欢走“填鸭”路线，似乎什么都可以做最好，但是事实是越单纯越性感。

你是否要做一款性感的产品？这是个值得思考的问题。

# Product Design: 产品的起点 - 设计工具的选择

---

## Sketch

---

在设计工具上，我选择了 [Sketch](#)，老牌的 Photoshop 其实也不错，但是 Sketch 针对移动端做了大量的优化，有几个我很喜欢的特性，比如

1. 多分辨率的支持
2. 多重格式的支持
3. 在移动端实时预览
4. 原生的效果支持，比如 iOS 的 Blur
5. UI 模版
6. 元素符号，可以让你方便的复用元素
7. Apple 官网的素材支持

当然老牌的 Photoshop 也不是一无是处

1. 专业的颜色管理
2. 丰富的配色方案
3. 强大的处理能力

在不面对印刷的时候，Sketch 是最佳的选择，也无外乎为什么现在几乎所有的移动端设计师都选择了 Sketch，但是就像学习 iOS 要不要学习 Objective-C 一样，永远不要因为工具的进步而放弃对基础知识的了解，会设计和会设计工具终究是两码事。

## Framer

---

对于交互原型的设计，你可以尝试 [Framer](#)，这个不像 Adobe 的 EdgeAnimate 那样纯粹的界面操作，而要做一些编程工作。事实上你会发现，这样做的效率要比使用界面拖拽来得更快捷。

1. 原型自动转换为 HTML 页面
2. 在设备上随改动同步预览
3. 分发给远程人员在任意设备查看

4. 真实交互
5. CoffeeScript 容易上手

可以直接导出成 HTML5 的网页在任意平台测试，好用得让你简直想用 HTML5 来做 App 了。

# Product Design: 产品的起点 - Sketch 基础

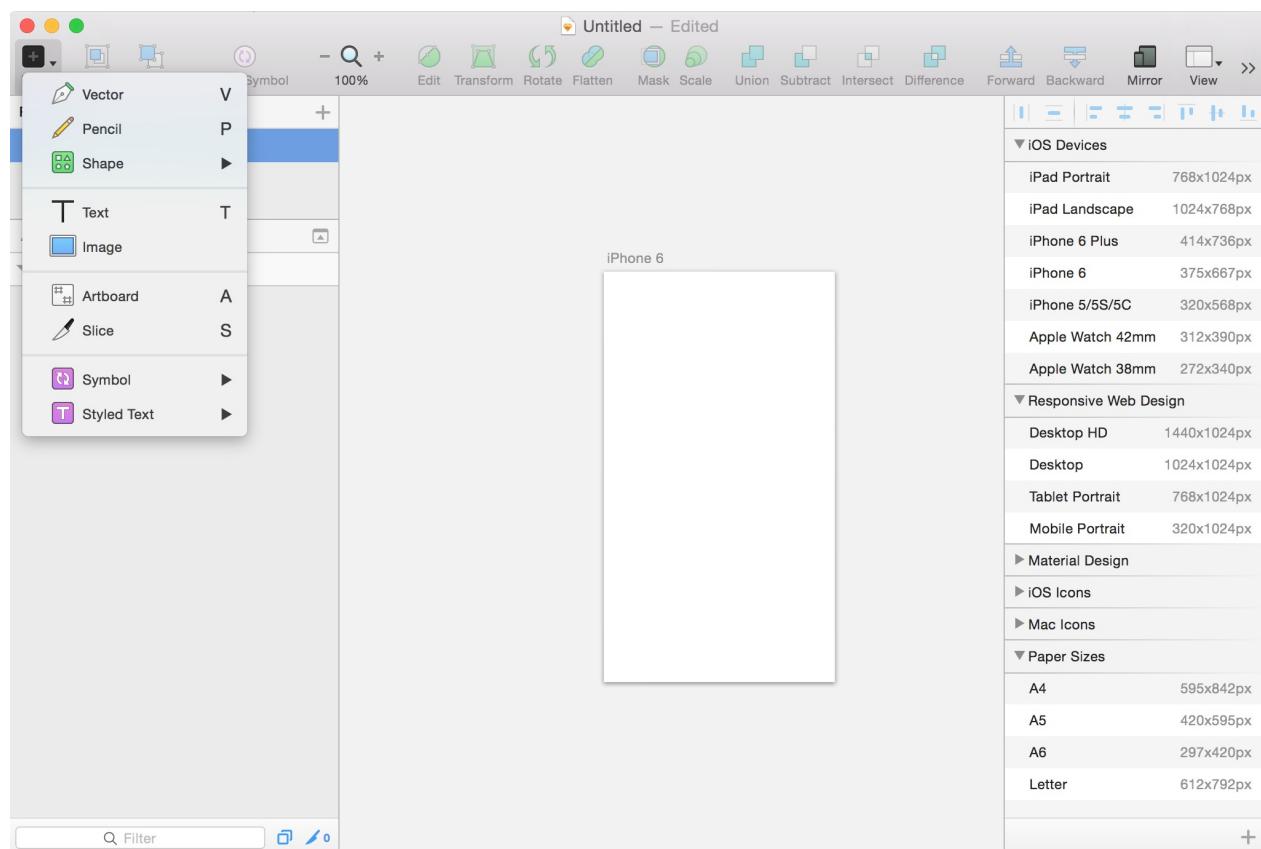
## Artboard

Sketch 项目的结构分为两级

1. Page
2. Artboard

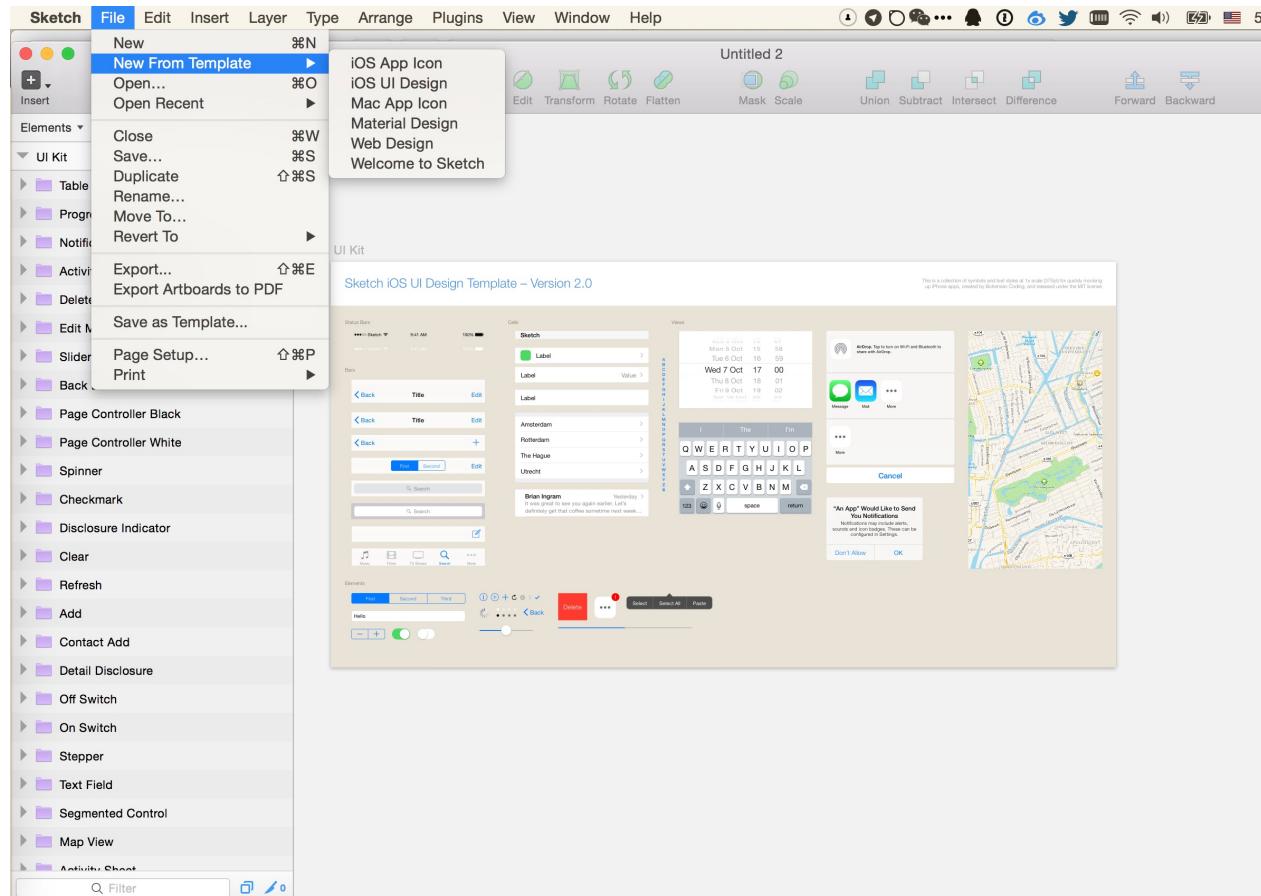
这种针对移动端特别优化的结构非常合理，例如你的 App 有一个添加好友的流程 那么你可以新建个 Page 叫做 Add Contacts，然后每个步骤用一个 Artboard。

点击坐上角的 加号 可以添加 Artboard，也可以按下 快捷键 A 来添加，右边 Sketch 内置了各种设备的尺寸，无数贴心的设计使得 Sketch 无可厚非的成为移动设计的王者。



## Templates

如果 Artboard 已经让你欣喜不已，那么模版功能就会让你立刻爱上这款软件，Sketch 内置了多种操作系统的 UI 模版，你可以快速组装出来自己界面的基础架构。

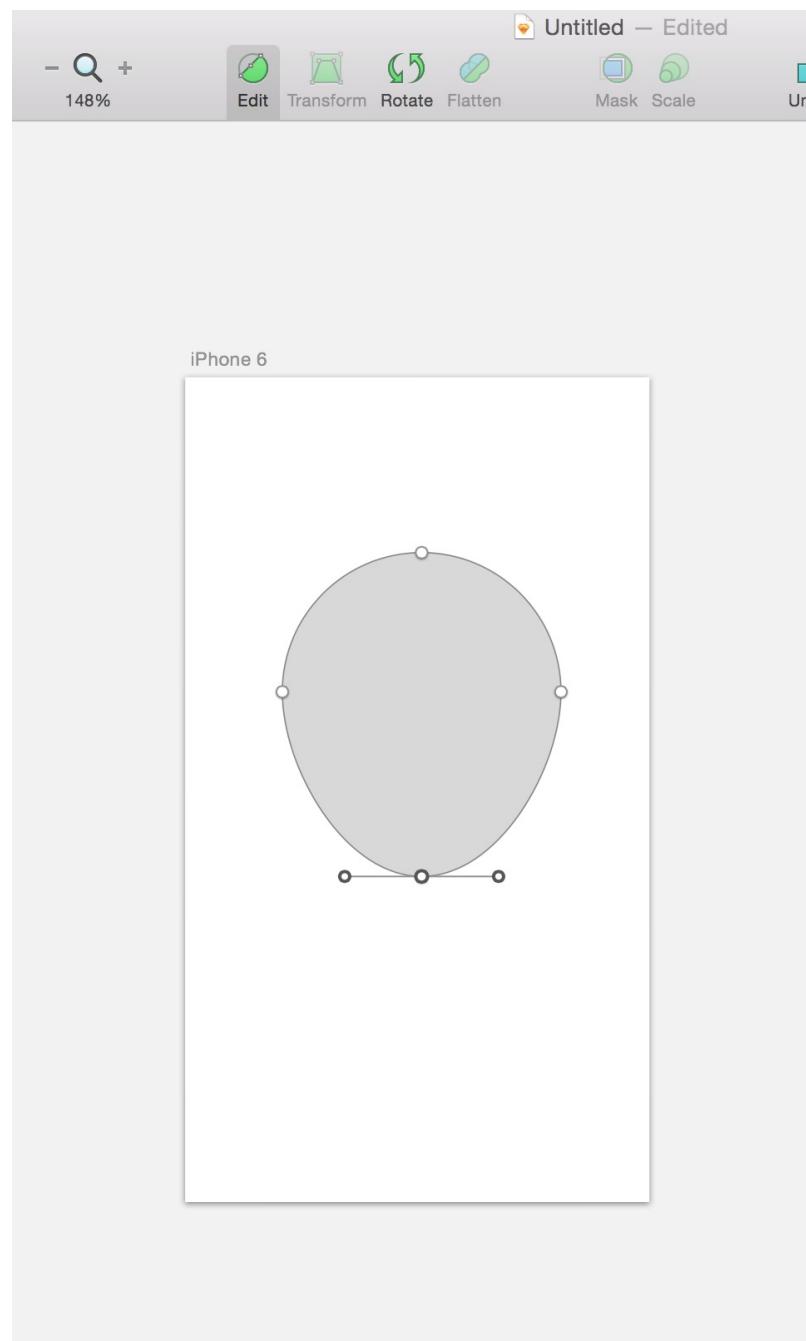


## 元素编辑

Sketch 有 Edit, Transform, Rotate, Faltten 四种基本元素编辑的方式。

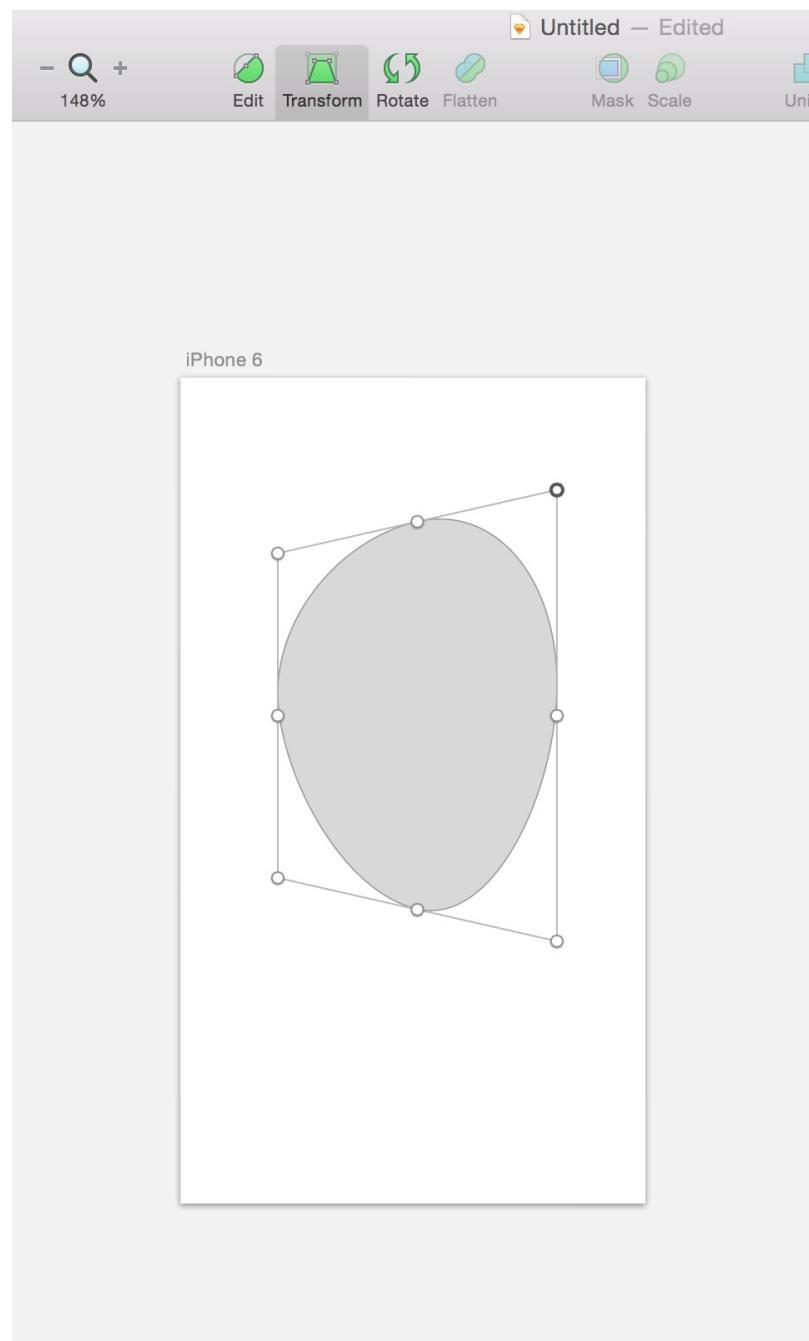
### Edit

选中元素后点击 Edit，元素立刻就会显现出曲线的原型，编辑元素曲线可以创造任意你想要的形状。



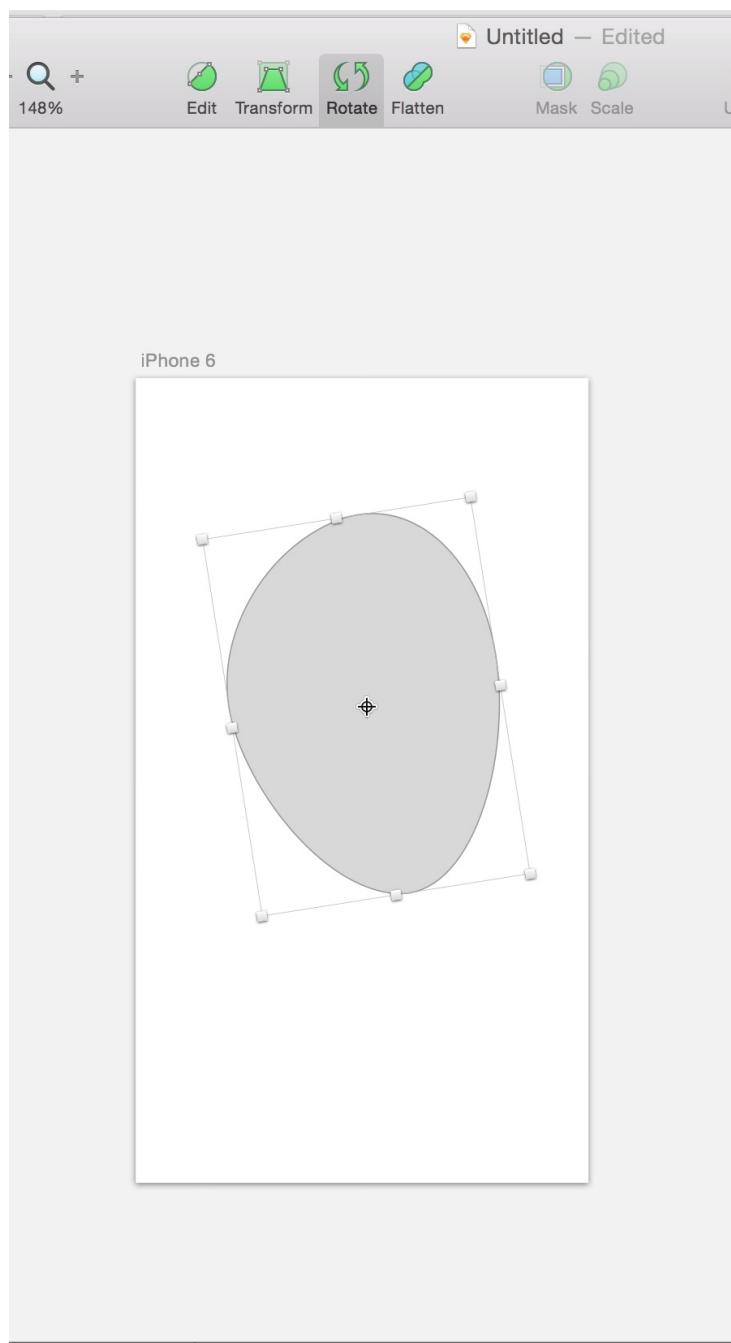
## Transform

Transform 可以对元素的三围角度产生形变。



## Rotate

Rotate 则可以旋转元素。

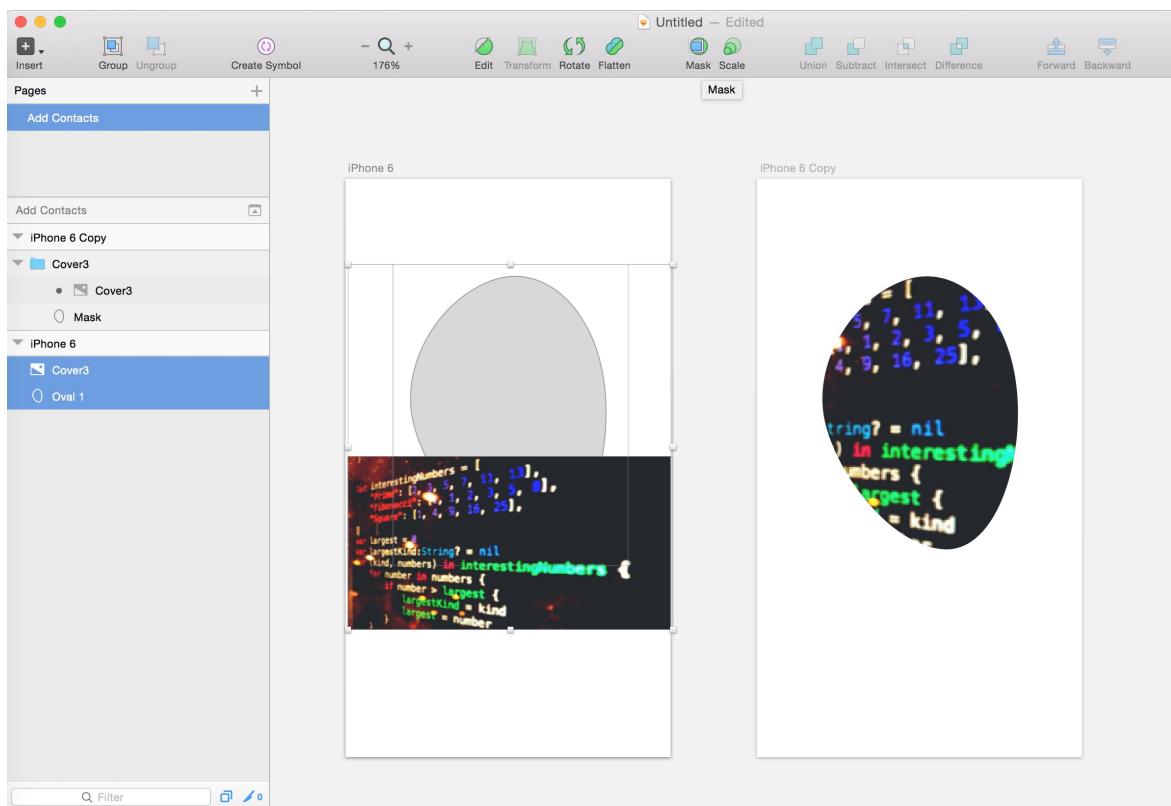


## Flatten

Flatten 则可以基于你改造后的元素重建外框为规则矩形，这个功能可能要你亲自在 Sketch 里尝试下才能深刻理解。

## Mask

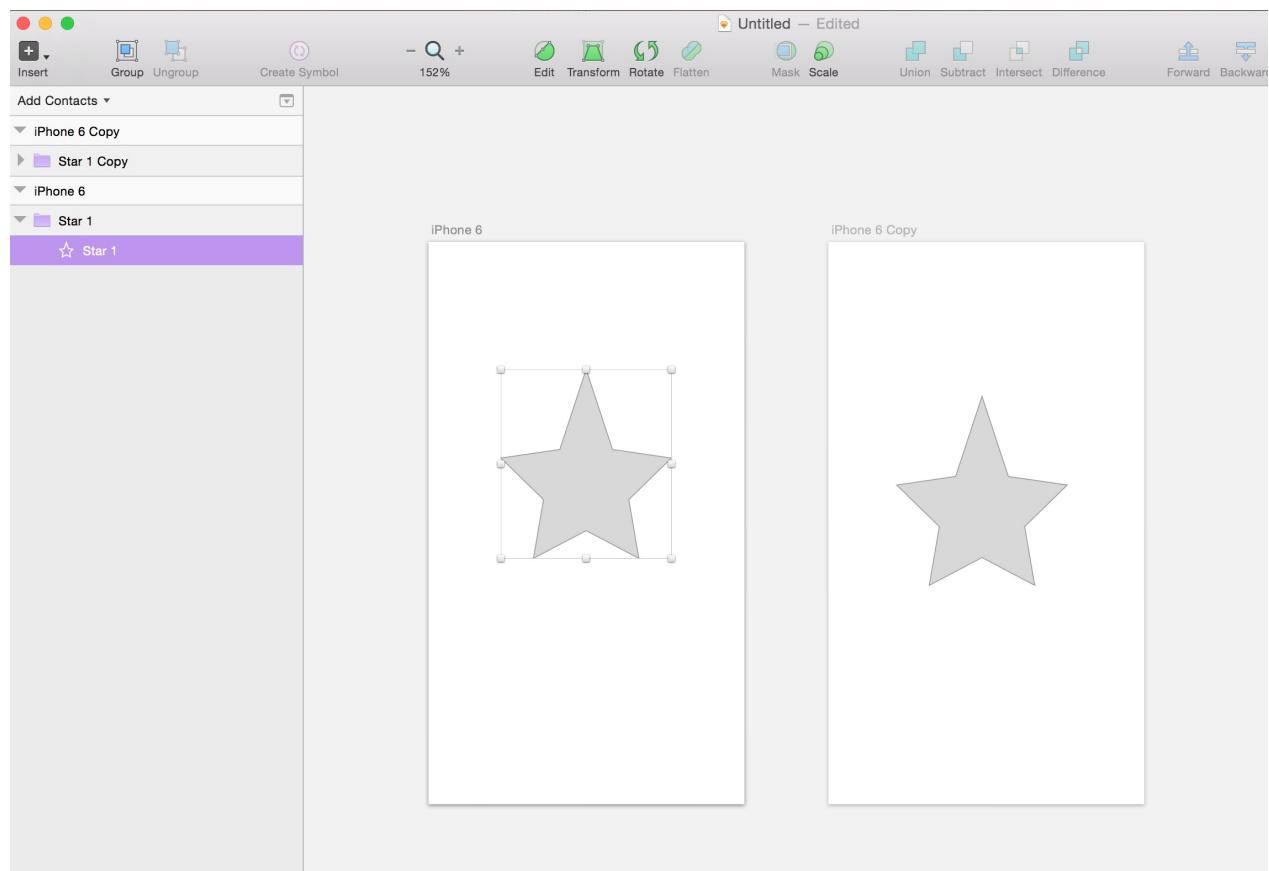
Mask 是一个非常重要的功能，当你想制作圆形头像的时候，很可能是你有一个圆形，一个正方形的照片，那么就要用圆形 Mask 这张照片，就可以看到一张圆形的头像。刚才的图形 Mask 一张照片，就可以产生这样的效果。



## Symbol

一个 App 里面往往有很多元素是通用的，你肯定不希望每次都要逐一修改，这就是 Symbol 产生的原因。

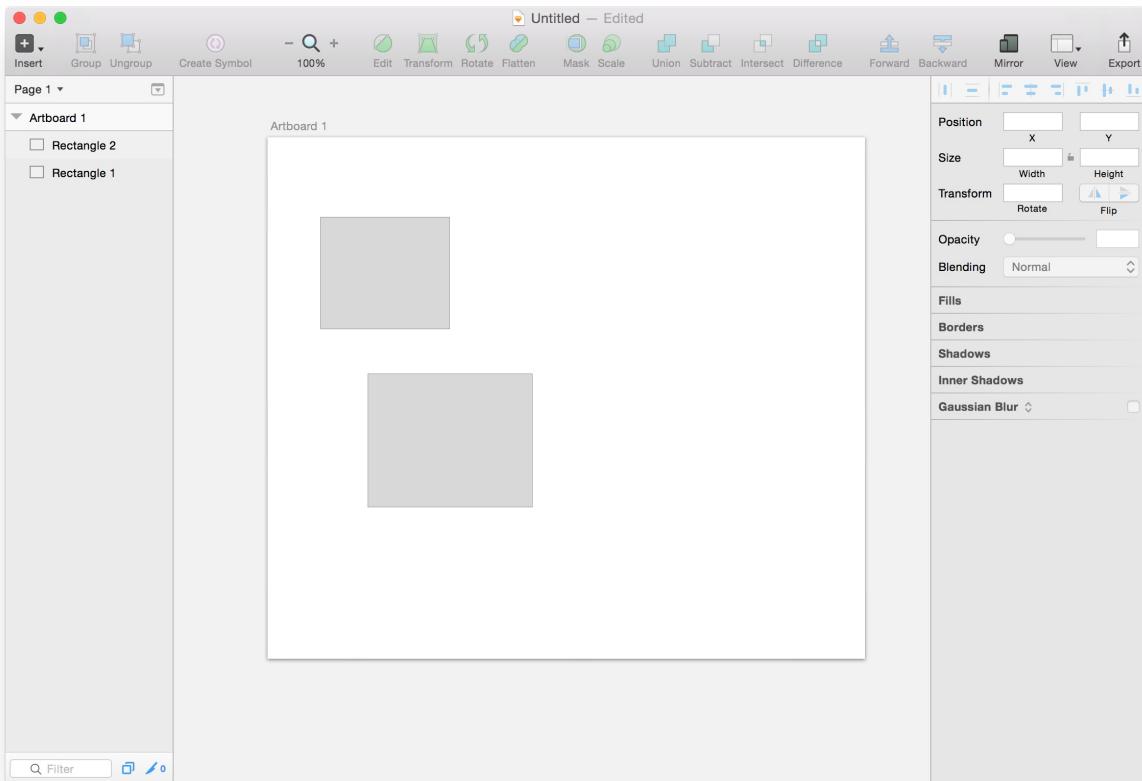
我创建了两个 Artboard，并且给第一个插入了一个星星，然后设置为 Symbol，然后复制这个星星到另外一个 Artboard，当我编辑一个星星的时候，另外一个会产生同样的变化。



这些只是 Sketch 非常基础的使用方法，当你深入学习之后，Sketch 会成你手上爱不释手的神器。

## 对齐

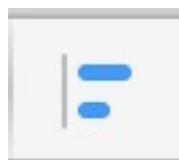
快捷对齐是 Sketch 里非常实用的一个功能，你在界面设计的时候会大量使用这个功能。



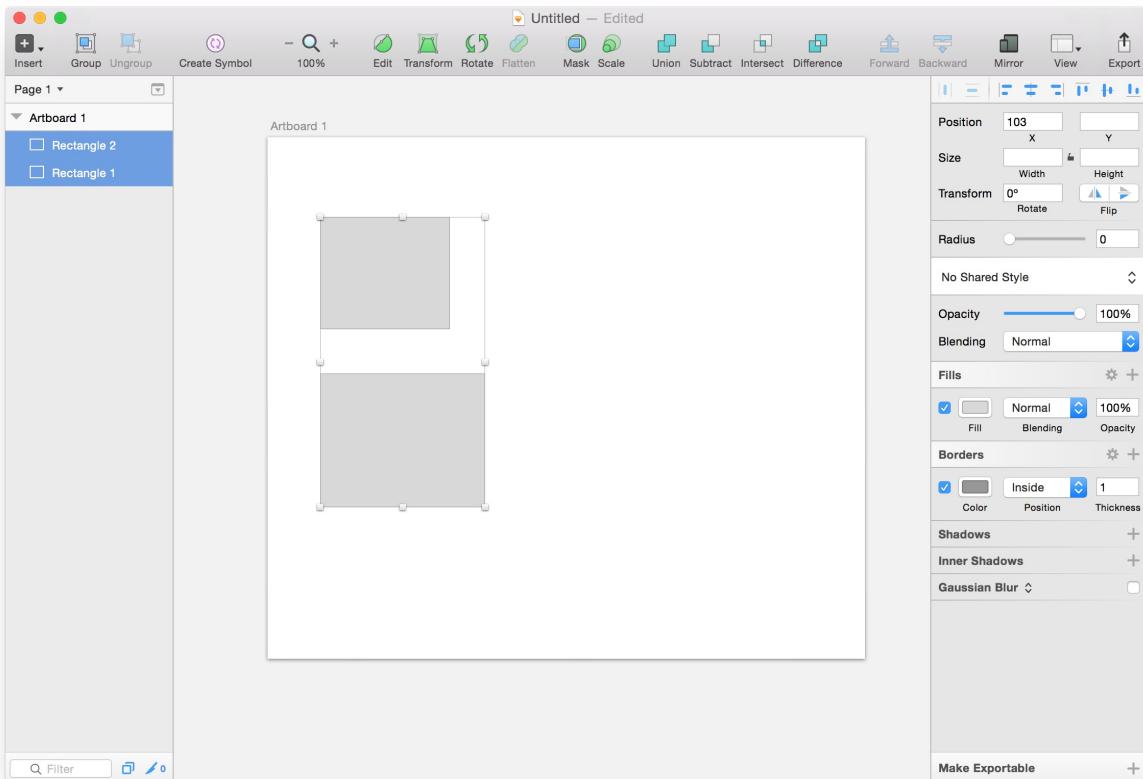
例如上图我们有两个矩形元素，希望他们做对齐，普通的做法可能是拖拽下面的那个然后对齐上面那个。

高端的呢？

同时选中两个矩形，点击右上角的做对齐标志



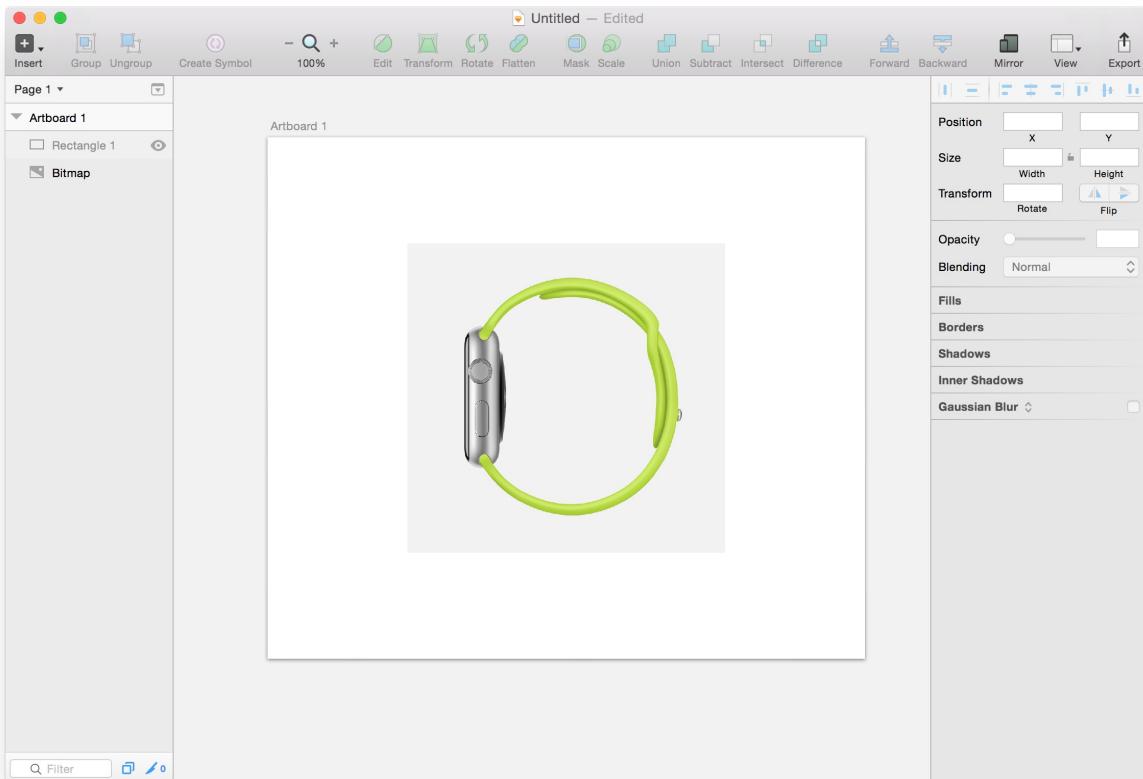
然后我们的奇迹就诞生了



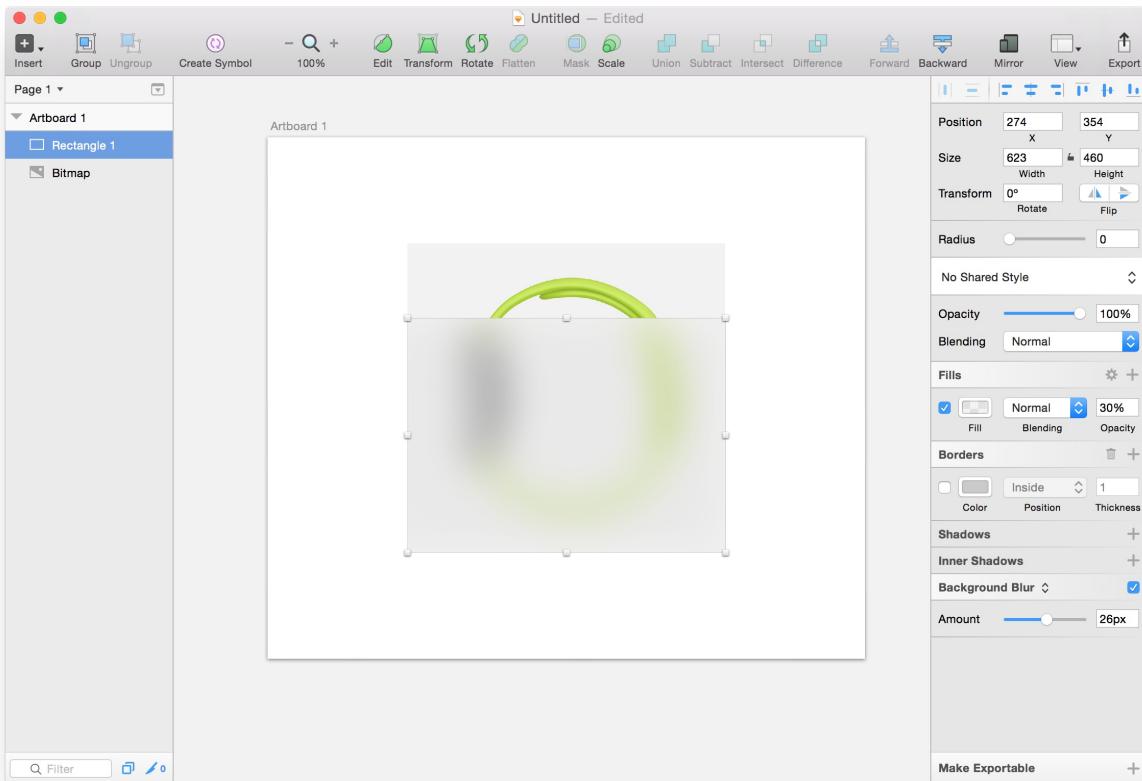
## Blur

iOS 的 Blur 效果在 Sketch 里实现起来非常简单——因为已经内置了

例如我想要挡住下图中的苹果表



只需要新建一个矩形，放置在这个图片上，然后把右侧边栏上的 Gaussian Blur 换成 Background Blur 即可。



别忘了调整 Amount 和 Fills 里的 Opacity 来改善效果。

## 深入学习 Sketch

[Learn Sketch](#)

# Product Design: 产品的起点 - Framer 基础

## CoffeeScript

CoffeeScript 是 JavaScript 的二代语言，或许这两种语言对你来说此刻都很陌生。不过没关系，我们学 CoffeeScript，问题立刻就少了一半。

最快入门一个语言的方法不是从理论开始，而是直接使用这门语言。Swift 是我们的主菜，CoffeeScript 这个小菜可以直接端上来。

## CoffeeScript 基础

你可以进入 [CoffeeScript](#) 的官网，点击 Try CoffeeScript 来学习这门语言。

### 赋值

从现在开始你要习惯计算机世界里各种奇奇怪怪的名字，做的事情都很简单，但是名字着实卖弄。

```
number = 42
alert number
```

把 42 赋值给 number，接下来点击右边的 RUN，你就会看到一个弹窗提示 42。

### 判断语句

```
newNumber = 43
if newNumber > 42
    number = newNumber
else
    number = 42
alert number
```

if..else 是程序世界最常用的条件判断语句，这里的含义就是当 newNumber 大于 42 的时

候，number 赋值为 newNumber 的值，如果小于 42，number 赋值为 42。

点击 RUN，你会看到弹窗提示 43。

## 函数

```
sum = (x,y) ->
  return x + y

number = sum(1,2)

alert number
```

函数是程序世界里另外一个非常常用的功能，这里我们定义了一个函数 sum 它接受两个参数 x 和 y，并把他们的求和返回出来。

RUN 一下可以看到弹窗提示 3。

可能现在你的大脑已经浆糊了，-> 是什么来的？

其实这些东西都不需要深究，每一种编程语言都有自己的约定，而 -> 就是表示函数的特殊符号，你可以当作这是画画。

CoffeeScript 使用行与行之间的缩进关系来表示代码之间的关系，如果你写成这样

```
sum = (x,y) ->
  return x + y

number = sum(1,2)

alert number
```

那么再点击 RUN 的时候，就会出现这个错误

```
SyntaxError: Illegal return statement
```

因为缩进关系一变化，逻辑就也发生了变化。

## 对象

```
me =
  age: 23
  height: 178
  weight: "secret"

alert me.weight
```

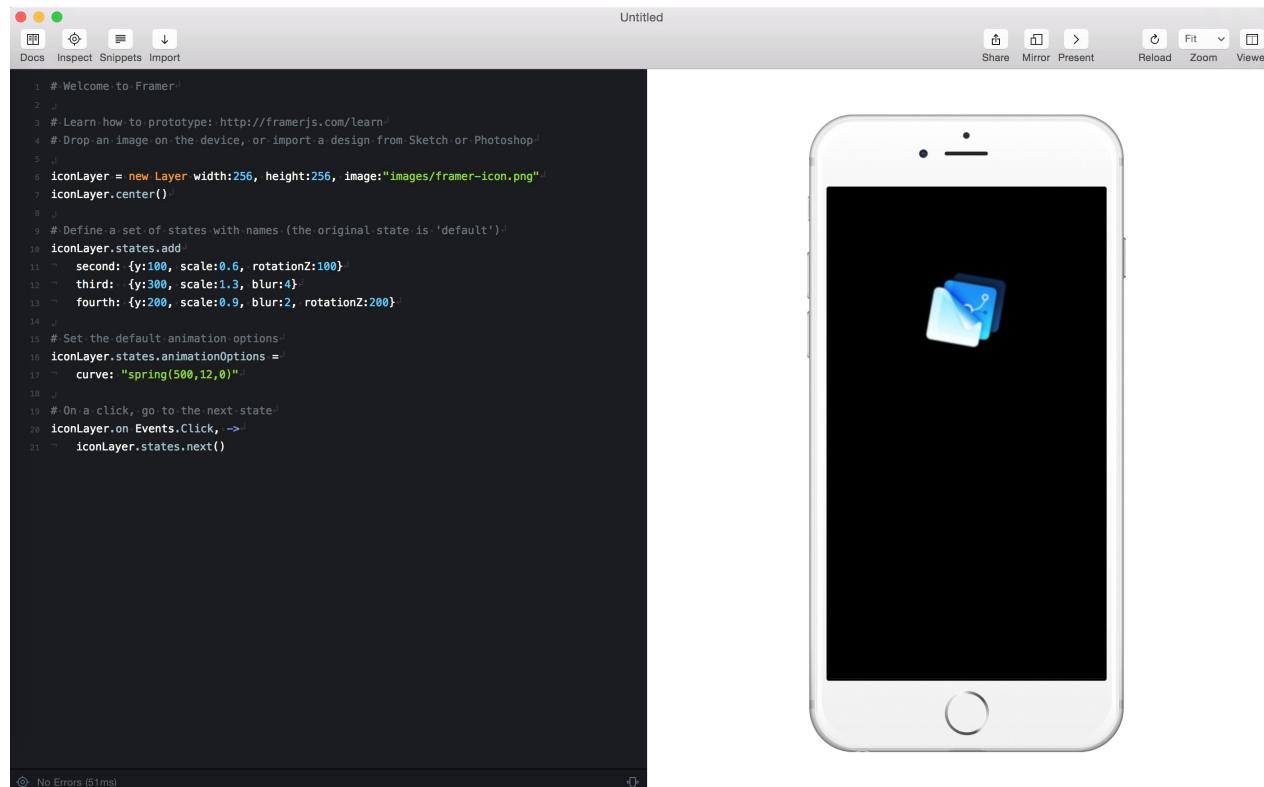
我们可以把一组参数打包进去一个对象里，比如这里我们创建了一个 `me` 对象，里面有年龄，身高，体重。

通过 `me.weight` 这种方式，就能够获得对象属性所对应的值。

当你 RUN 的时候，就会有弹窗提示，`secret`。

## Framer

打开 Framer 之后就可以看到他们已经附带了一个例子



逐行解释下：

```

iconLayer = new Layer width:256, height:256, image:"images/framer-icon.png"

# 创建一个 Layer, 名字是 iconLayer, 宽度是 256, 高度是 256, 显示本程序文件所在的文件夹下 images

iconLayer.center()

# 居中显示这个 Layer

iconLayer.states.add
  second: {y:100, scale:0.6, rotationZ:100}
  third: {y:300, scale:1.3, blur:4}
  fourth: {y:200, scale:0.9, blur:2, rotationZ:200}

#给这个 Layer 定义三种状态, 分别设定了三种状态下 Layer 的 y 坐标, scale 缩放, 以及 blur 模糊程度
#这里有 second, third, fourth, 但是first 是什么? 没有做任何变化之前的默认状态就是 First

iconLayer.states.animationOptions =
  curve: "spring(500,12,0)"

# 设置进行状态变化的时候的动画曲线参数, spring(500,12,0) 是弹簧动画, 里面参数的意义就需要阅读文档

iconLayer.on Events.Click, ->
  iconLayer.states.next()

# 当用户点击这个 Layer 的时候, Layer 的状态自动切换到下一个

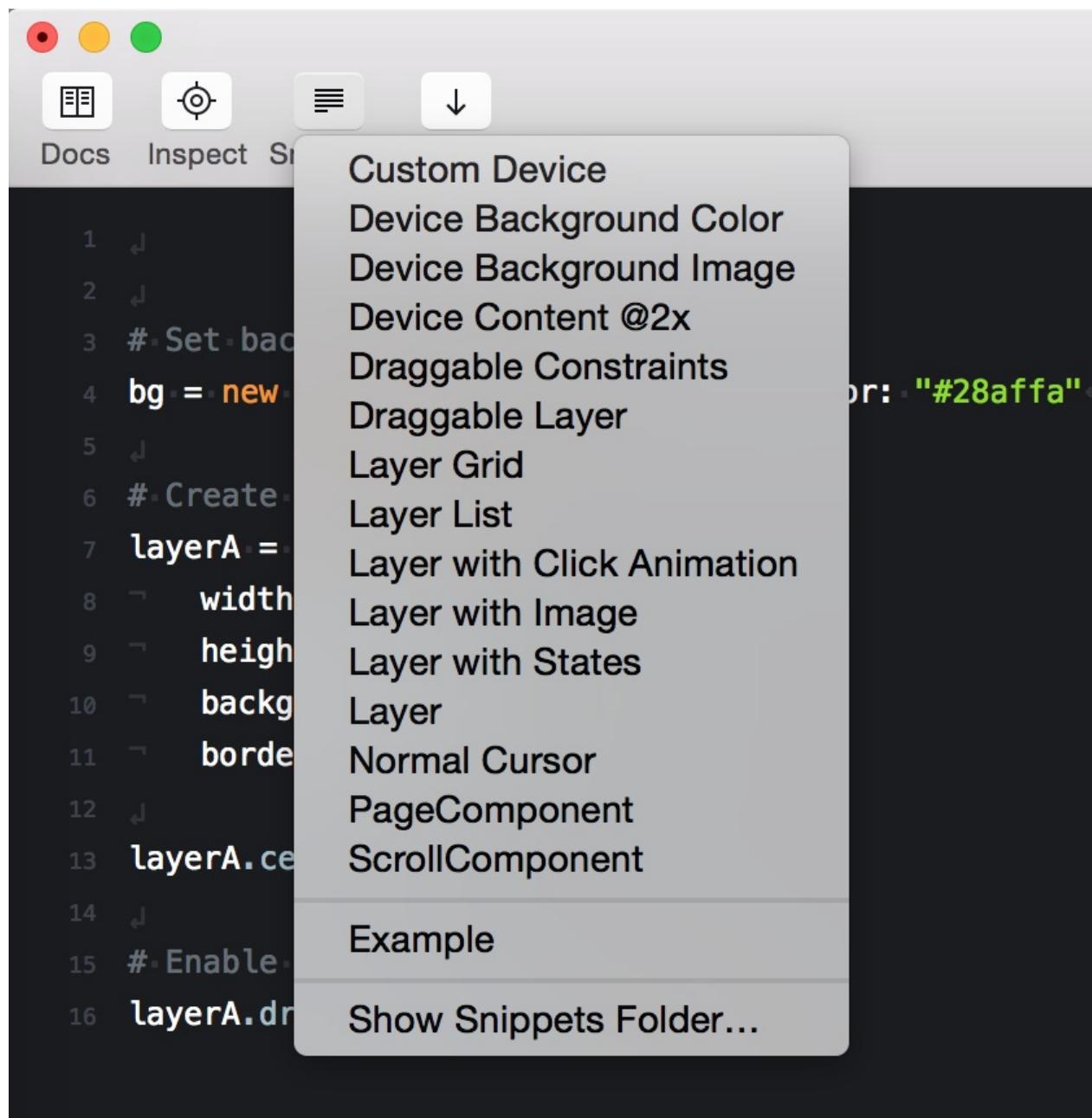
```

对于 `spring` 参数的解释, 你可以[查看这里](#), Framer 有两种 Spring 效果, `spring-rk4` 和 `spring-dho`, 而 `spring` 被映射到了 `spring-rk4`, 所以上面的也可以写作 `spring-rk4(500,12,0)`。

## Drag

---

删掉上面所有的代码, 点击 Snippets 选择 Draggleable Layer



这时候右面 iPhone 就切换了一个场景，你可以拖动中间的白色矩形，来试试交互的感觉。

```

bg = new BackgroundLayer backgroundColor: "#28affa"

# 设置背景

layerA = new Layer
width: 250
height: 250
backgroundColor: "#fff"
borderRadius: 8

# 除了高宽，这个 Layer 还设置了背景色，圆角半径 8

```

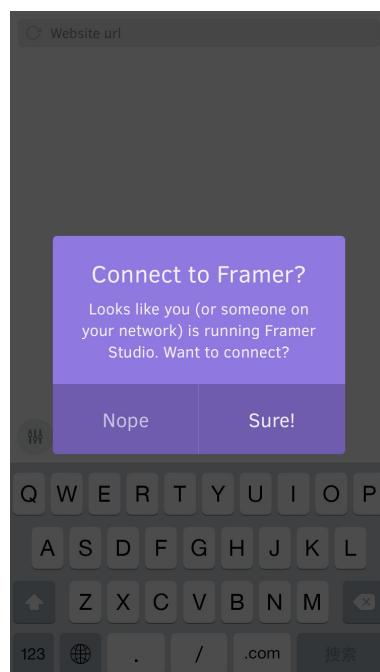
```
layerA.center()  
layerA.draggable.enabled = true  
  
# 使这个 Layer 可以跟随拖动
```

在我刚接触这样代码的时候，我最大的疑问是，为什么 `layerA.draggable.enabled = true` 设置了后，他就可以跟随我的拖动了？背后都发生了什么？

这些疑问会在某一天你来写一个通用模块的时候迎刃而解。

## 实时预览

在 iOS 上下载 Frameless 这个浏览器，你在手机上打开的时候，他就会询问你是不是连接 Framer，点击 Sure，就可以自动同步两边的界面。



在手机上点点试试，对，现在就是这么神奇。

## 游乐园

Framer 带了很多例子，先从这些例子来学习是很好的开始，你可以修改里面参数，反复运行，不管结果是对还是错，当成一个游乐园来玩耍，把不理解的地方都打包起来。

玩累了之后去官网查看对应的文档，当你在文档里看到要找的答案，茅塞顿开的感觉请牢

牢牢记哟。

恭喜你，很快，你就可以入门这个技术了。

## 深入学习 Framer

---

[Learn CoffeeScript](#)

[Learn Framer](#)

# Product Design: 产品的起点 - 设计 小记

小记 我最早的构思是想像 iOS 7 之后的日历一样，以时间为线索，通过纵深层次转换来表现时间的变化，以呈现不同级别的内容。

但是当我开始设计的时候，我突然想为什么要做一个没有特色的东西？为什么日记一定要为大量书写，图文混排而生？

于是在 Sketch 上经过一番尝试，最终，以时间为交互线索，纵书为展现格式的设计诞生了。

为了尽可能实现原汁原味的效果，我做了大量的日本直书设计研究，其中最有价值的是 W3C [关于纵书的规范](#)。

在这之后，首先我设计了主界面——年



## 字体

宋体似乎是每个互联网圈的设计师都极力避免的，但是在这个 App 里，宋体以其严谨，稳重，恰好适用。在界面的设计中，字体的选择起到了决定性的作用，完全可以左右设计

的气质和特点。

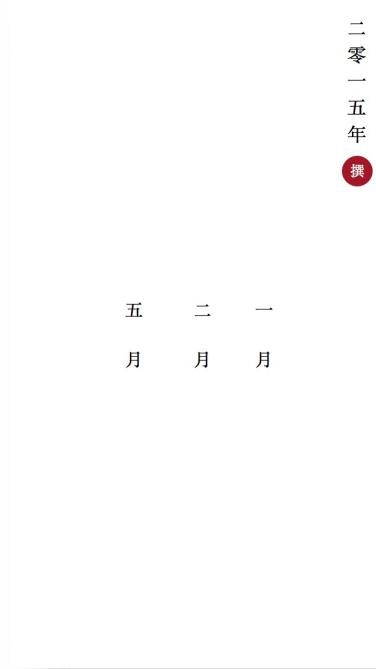
尝试多种字体的效果，选择最贴切的版本，是让产品找到自己灵魂最简单的方法。

## 对齐

对齐，设计里非常关键的要素，我刚开始做设计的时候特别喜欢为了追求不同，而把标题摆在角落里。这样设计虽然看起来够反叛，但是往往是讲不通的，为什么放在右下角，边距是多少，为什么设置这些边距。但是凡事无绝对，在特殊的场合你也可以利用这种无规则来表达一种情绪。

居中是一个很好用的方法，特别是在 小记 的设计里，在打开这个 App 的时候，用户眼睛的视线会自然聚焦到年份这里，足够庄重，足够有仪式感。

接下来月的设计就有了更多的考虑



在前些日子看 [The Architecture Academy](#) 的时候，有一个让我印象深刻的一点——在室内设计中，尤其是极简主义，往往会在场景中布置一个物体来吸引观察者的眼球，以发生联想。

## 色彩

在年的下面，我放了一个红底的“撰”，我想每个用户看到这个“撰”的时候，都会想去点击这里，并且让平稳的平面产生了活力。撰的位置在年视图和月份的视图里都是固定的，随着用户的使用，按钮的位置会成为一种肌肉记忆，从而避免了视觉搜索，凭感觉就可以选中。

在红色的选择上是很谨慎的，这个红色特别从 Photoshop 的 CMYK 色表选择出来，随便在色盘上拽一个颜色是很不负责的，RGB 虽然可以组成你看到的一种颜色，但是会出现这种 RGB (1, 150, 120) 这种情况，事实上并不需要那一点 Red，但是确实加入了这一点 Red，这会导致在诸多情况，特别是印刷的时候的杂色问题。

“二零一五年”的字体大小是 36，距离顶部和右边边缘的距离都是 36，“撰”的圆形顶部距离“年”也是 36，并与“年”的中心对其。这种数字上的精确是美感逻辑上的保证，虽然可能 36 和 35 差不多，但是从不同的人眼里，这个差距会无限放大，特别是从你自己的心里。

月份之间留出了大概一个拇指的宽度，方便交互。月份设置在屏幕的中间，并且只显示三个，多余的通过向右滑动来查看。选择三也是因为它是中国文化里很有禅意的一个数字，放在这里正好能贴合设计上的美感。

在月份里面，这种逻辑被继续贯彻



增加了一个红色的月份，放在右边，与月份顶端对其，与年中心对其——这样月就像这两个区域的中枢，一种元素之间的和谐。

那么接下来，这就是正文的设计

生命 我始终不甚了解  
但是从你那  
我方才开始明白  
每一个关于女人的神经或者纤柔  
礼物不能及时送到  
这天杀的春运  
礼物不能及时送到

二〇一五年二月十四日于广州珠江畔

## 版式

参考 W3C 关于纵书的规范，每行之间的距离大概是字体的大小。字体的间距设置为 3 点，在最终的位置和时间上做了字体的缩小以减少视觉上的吸引。

到此设计就基本完稿，然而在过年的时候，我去了一位从事建筑设计叔叔家住了几天，。在我和他聊这个设计的时候，他又一次给了我很重要的建议：如果你要做这种设计，就把这个味道做足。

于是最终，我使用康熙字典体和文悦古仿宋把这款 App 带入了另一个境界。



这让我回忆起多年前，我刚开始做设计的时候，他告诉了我设计中一个极为重要的要领，就是设计并不是把多个灿烂夺目的东西拼凑到一起，或许一个就足够了。

你可以[在这里](#)获得完整的 Sketch 文件。

## 几个设计里的基本法则

1. 不要仅仅是为了不同而设计
2. 通过点睛的元素赋予设计灵魂
3. 色彩选择上的谨慎
4. 注意元素之间的对齐逻辑
5. 元素边距之间的逻辑，针对人机交互进行优化
6. 元素之间的和谐
7. 参考专业的资料
8. 寻求不同人士的意见

# Product Design: 产品的起点 - 交互设计

人机交互是设计中最复杂的一部分，因为往往需求和现实总是矛盾的。对程序而言，人们总是希望功能强大，但是又要使用简单。或者既要设计简洁，又想在里面体现更多的内容。

对于设计师而言，最大的问题莫过于人们认知的偏差，总是有些非常简单的设计，最终却在用户手里变成了要 Google 一下才能知道的 Trick。

## 人脑的工作模式

人脑主要由三部分组成

- 旧脑 —— 世界分为三类，可以吃的，危险的，性感的。
- 中脑 —— 对事物产生情感
- 新脑 —— 理性思考

三种大脑形成了两种思维模式，由旧脑中脑组成的系统一，以无意识，自动思维为主。

新脑这个系统二，则以逻辑思维为主。

熟能生巧就是指的任务的执行由系统二逐渐过渡到了系统一，而很不幸的是，当你设计交互的时候，你面对的都是用户的无意识，自动思维的大脑。

明白了这个原因后，

“没有用户会这样想。”

“用户仔细看一下就会发现的。”

这些的话就再也不会成为你设计上偷懒的借口了。

## 交互的思考

设计交互的步骤的时候主要考虑下面几个问题

1. 心理感知
2. 用户时间
3. 产品定位
4. 功能数量

## 心理感知

做产品的人总是喜欢说“用户是傻瓜”，其由来就是因为用户总是会把所看到的设计和自己经验里的设计进行相似匹配，来完成功能的猜想。

也正是因为如此，许许多多创新的设计和交互在缺少指引的情况下，用户会胆怯得发呆，然后跑去给你评论“你们这个 App 是什么鬼东西，完全不会用。”。

其实许多交互稍作尝试就可以发现如何使用，但是绝大部分用户都是胆怯型的。

从另外一个角度，使用通用型设计就极大降低了用户的使用门槛，通用型的设计经过了时间的考验，往往是效率最高、感知偏差最小的模型。这一点 Telegram 和 微信之间就是很好的例子，两者都使用了 Tab Bar，框架也基本一致，用户从微信转到 Telegram 使用，并不会有不适。

小记 使用红色的“撰”作为编辑入口，这种色彩和形状上的对比也符合用户对功能心理预期。

## 用户时间

繁杂的交互极为让人诟病，尤其是为了实现一个简单的目的却要做许多复杂的操作。

以前的 PC 时代，你可能冬天大半夜突然爬起来，打开电脑，等上两分钟开机，在冷飕飕的屋里都快冻蒙了只是为了修改 Word 文档里的几个字。

而现在，你只需要躺在被窝里就可以在 iPhone 上轻松完成。

如果用户每次打开你的 App 是为了完成一件事，那么千万不要让他们点来点去，App 开启后直接定位到那里吧 —— 甚至你可以砍掉其他功能。

在 Evernote 的移动端，在你打开的时候 App 就会定位到上次退出时停留的页面。

而 小记 在被打开的时候，也会自动定位到当前月份的最后一天。

## 产品定位

每一种设计都有其不同的受众，小记的极简设计正对应了这个被过度消费的时代，每个人都像拜物狂一样占有了满屋子的物品，拥有三台以上手机的人都不在少数。

App 也像填鸭一样，为了利益和市场格局不停的塞一些可有可无的新功能。

此时一款简单的 App 反而能唤起人们对简单的美好回忆。

另外一款很特别的软件 —— Clear 也是如此，它的交互非常酷，但是特殊的交互和配色并不适合所有人使用，iOS 自带的事项提醒交互直白，给人一种稳重的感觉。

对新鲜事物触觉敏锐的人会更喜欢 Clear，而历经人世沧桑的人会喜欢后者。

## 功能数量

功能一多，软件就可能会变得难以设计，删繁就简，采用折叠的方式把功能先归类隐藏成了更好的选择。

但是，此时会回到上一个问题，产品定位。

移动端普通用户往往只需要一个软件完成一两个事情，当功能增多进行折叠之后，用户因为时间成本的增加，会宁愿装几个简单的 App 来完成其他功能。

保持简约是移动时代的制胜法宝。

## 交互的动画

当我刚开始会用动画的时候，我希望每个界面的和元素的每个交互，都有动画效果，否则我岂不是白学了动画？

但是很快我意识到，动画并没有真的改进体验，第一印象会很酷炫，但是软件因其功能的不同，对交互动画的要求也是截然不同的。

设计交互动画的时候，**时间成本** 是最大的问题。

如果你是一款订票软件，那么用户不需要切换页面到时候任何酷炫的效果。

超过 0.5 秒的动画对他们来说都是时间上的浪费。

但是如果这个动画放在订票成功之后，做个烟花的动画也不会惹人厌烦。

动画的本质是让程序的等待变得自然甚至不可感知，后来衍生出了增加人机情感的作用。如果不必要的动画延长了等待，那反而伤害了动画的初衷。

我们将在后面讨论关于交互动画的实现。

## 深入理解交互设计

---

[《简约至上》](#)

[《认知与设计》](#)

# Product Design: 产品的起点 - 小记 的交互设计

小记 的交互非常简单：

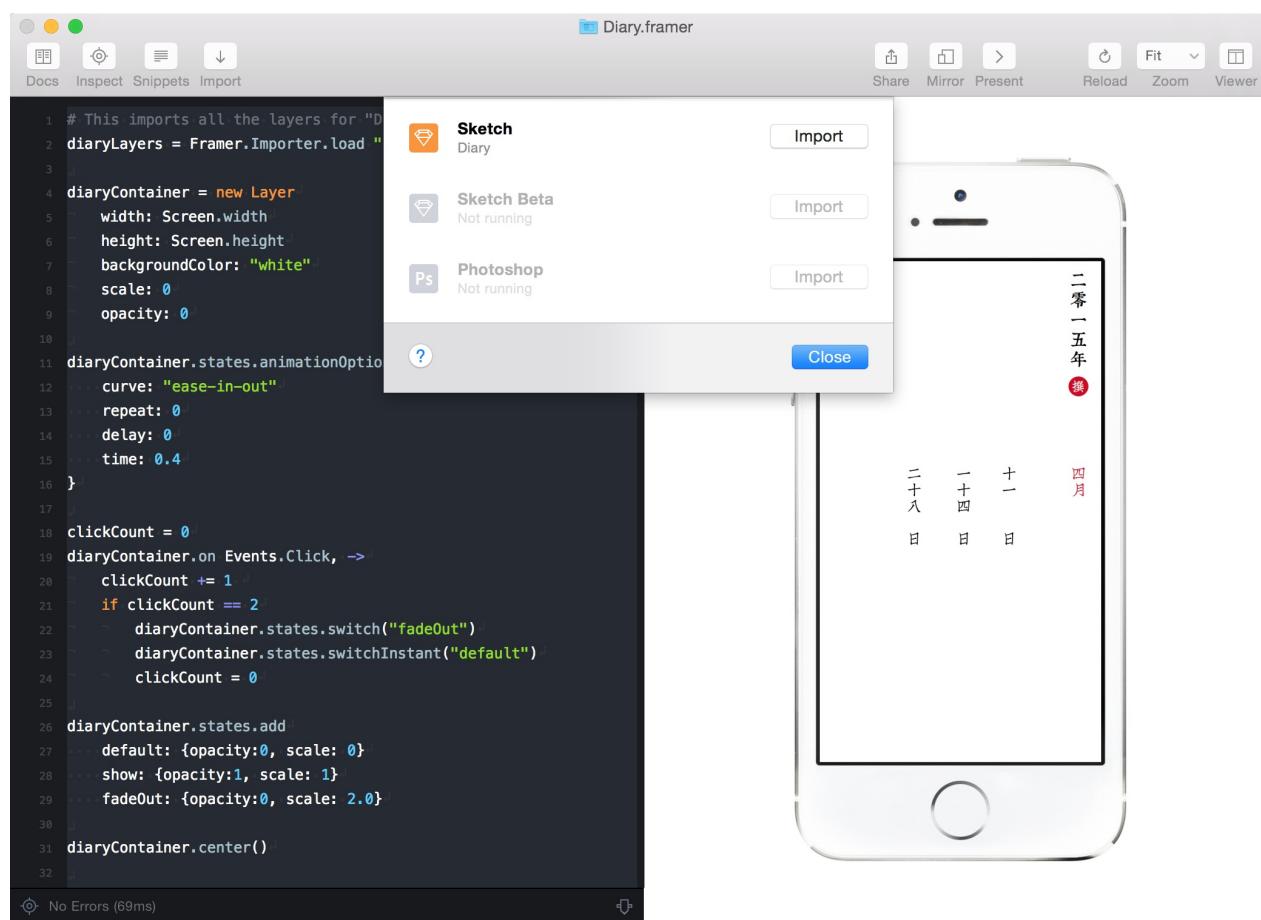
- 单击进入
- 双击返回

返回的时候采用放大，渐变的动画。

Sketch 里已经完成了界面的设计，剩下的就是导入到 Framer 里加工一下交互。

## 导入 Sketch 文件

打开你的 Sketch 文件，然后切换到 FramerStudio 里，点击 File -> Import



这时候 Framer 自动根据我们的 Sketch 文件里对每个 Artboard 的命名生成了图层。

先把主页面显示出来

```
# This imports all the layers for "Diary" into diaryLayers1
diaryLayers = Framer.Importer.load "imported/Diary"

# Create layer
container = new Layer
  width: Screen.width
  height: Screen.height
  backgroundColor: "#fff"

container.center()

diaryLayers.Month.visible = true
diaryLayers.Month.superLayer = container
```

点击中间的日期后切换到日记页面

```
# This imports all the layers for "Diary" into diaryLayers1
diaryLayers = Framer.Importer.load "imported/Diary"

diaryContainer = new Layer
  width: Screen.width
  height: Screen.height
  backgroundColor: "white"
  scale: 0
  opacity: 0

diaryContainer.states.animationOptions = {
  curve: "ease-in-out"
  repeat: 0
  delay: 0
  time: 0.4
}

diaryContainer.states.add
  default: {opacity:0, scale: 0}
  show: {opacity:1, scale: 1}
  fadeOut: {opacity:0, scale: 2.0}

diaryContainer.center()

diaryLayers.View.visible = true
diaryLayers.View.superLayer = diaryContainer

# Create layer
container = new Layer
  width: Screen.width
```

```
height: Screen.height
backgroundColor: "#fff"

container.center()

diaryLayers.Month.visible = true
diaryLayers.Month.superLayer = container

diaryLayers.Month.on Events.Click, ->
  diaryContainer.bringToFront()
  diaryContainer.states.switch("show")
```

双击日记页面后页面消失

```
clickCount = 0
diaryContainer.on Events.Click, ->
  clickCount += 1
  if clickCount == 2
    diaryContainer.states.switch("fadeOut")
    clickCount = 0
```

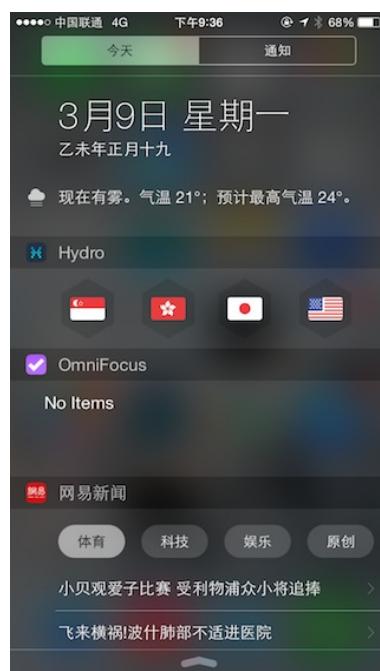
短短 50 行代码，Framer 和 Sketch 就可以迅速完成交互的原型。

这个双击返回的交互贯彻在所有界面。

# Product Design: 产品的起点 - Hydro 的交互与体验设计

去年感恩节的时候，我和 Ray 突然谈起能不能在 Today Widget 里面快速开关 VPN 的问题，我们希望做一款极致体验的 VPN 产品，送给朋友和家人以表达感谢（不要以为我们从去年一直搞到现在，其实两天开发 + 四天打磨，两个月后才提交，今天才过审核而已……）。

之前 iOS 8 的 Personal VPN 相关技术已经被我们完全攻克，所以整体来说，这个 App 剩下的似乎只是设计和体验的问题。



## Hydro 名字的由来

Hydro 是从 Hydrogen 里来的，氢元素是元素周期表第一个元素，我们希望这个只有一个质子的产品，可以像中国第一次接入互联网的时候说的那句话一样，穿越长城，走向世界。

当你收到邀请函的时候，邀请函的发件人就是那个质子，Proton。



## 界面设计思路

在这款产品里，交互和视觉无论如何炫酷，都不应该影响 VPN 最根本的一个体验，就是快点翻越。所以我们没有做任何的页面切换的部分，而是所有交互层都叠加在了一张世界地图里。

0. 利用 Today Widget 的特性来提供快速拨号功能。
1. 灰色世界地图上会有一个 Station 基站标点标注服务器位置，标注点有一个波纹动画。
2. 运用视差效果增加 App 的动感。
3. 链接成功后地图变成蓝色，意为与世界成功联通的情感。
4. 每个可以点按的地方都做了缩放交互动画来体现 App 的层次感。
5. Station 基站使用红，绿，黄这三种通用颜色来表示线路质量。

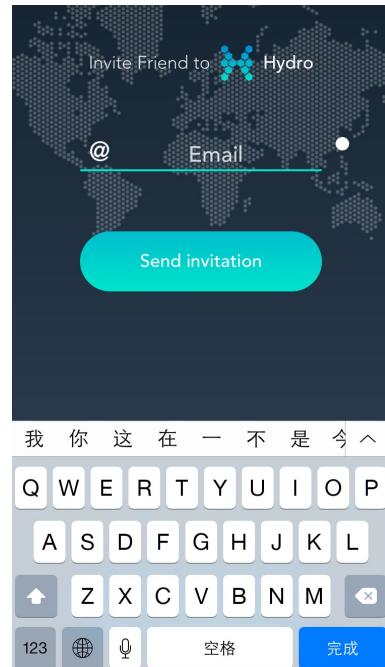
你可以[下载视频](#)来一探究竟。

## Today Widget 与 邀请

通知中心以及邀请界面由 Nix 开发完成，这算是 Hydro 里第一次应用了 AutoLayout 的地方。我属于纯代码流，加上比较懒，Nix 成了 AutoLayout 的布道者。

在 Today Widget 里也遇到了一些小坑，最坑的是开发过程中经常不响应部署。

邀请采用了一个比较取巧的方法，在你有资格的时候，上滑主界面就会进入邀请界面，这时候后面的地图会放大，产生一种空间的深度感。



## 机制的设计

VPN 服务最大的问题就是稳定性，为了最小化这种影响，我们只做移动端，并且实行邀请制。

1. 每个用户只能通过邀请注册
2. 用户五天后会产生唯一一个邀请码，用来邀请他最想带入的朋友或者家人

我们希望以此提供稳定的服务以及线路的通畅。

但是国际出口的情况很复杂，即使到今年，我所了解的能够影响国际出口的因素依旧在增加。

你可以[在这里](#)找到 Hydro 的源码。

# Product Code 产品的实现 - 成为开发者

---

在开始 iOS 开发前，很重要的一件事是成为 iOS 开发者，这一步着实很难迈出，首先你需要一台 Mac，然后你需要一台 iOS 设备（最好是 iPhone），最后你还需要花费 99 美刀注册成为 iOS 开发者。

## 硬件的问题

---

在硬件这个门槛上，Apple 最终会教会你几件事情 ——

- 能用钱解决的问题不要浪费时间；
- 对自己投资是一件最超值的事情；
- 免费的才是最贵的。

可能你最先想到的是，我虚拟机（可能是盗版虚拟机）弄个 Mac 系统（黑 Mac）吧，另外一些人可能想——我买台 Mac Mini 吧（大概 4000 元），还有些奇葩可能跑到社交网络说“我真心想学 iOS 开发，谁能送我台 Mac！”

这些都会发生，不过我给你的建议是——买台 Macbook，买台 iPhone（预算不够可以买官方翻新版）。

## 注册开发者

---

首先，你得有个信用卡，不一定非是你自己的 ;)

然后你就可以前往 [Apple 开发者中心](#) 注册你的 Apple ID 并开始申请 Developer 了。

## Are you enrolling as an individual or organization?

### Individual

Select this option if you are an individual or sole proprietor/single person company.



#### Seller Name

Your legal name must correspond to your tax ID and will be listed as the seller of your apps on the App Store.

Example:  
Seller: John Smith

#### Individual Development Only

You are the only one allowed access to program resources.

#### You will need:

- A valid credit card for purchase.  
We may also require additional personal documentation to verify your identity.

### Company/Organization

Select this option if you are a company, non-profit organization, joint venture, partnership, or government organization.



#### Seller Name

Your organization's legal entity name will be listed as the seller of your apps on the App Store. This name must correspond with the tax ID you plan to use.

Example:  
Seller: ABC Company, Inc.

#### Development Team

You can add additional developers to your team who can access program resources. Companies who have hired a contractor to create apps for distribution on the App Store should enroll with their company name and add the contractors to their team.

#### You will need:

- The legal authority to bind your company/organization to Apple Developer Program legal agreements.

选择 Individual

然后确认你的资料，在项目确认对话界面，选择 iOS Developer Program

Program	Price	Status
iOS Developer Program	RMB 688 /year	Your membership has expired. Renew your membership now.
Mac Developer Program	RMB 688 /year	Get everything you need to develop, sign, and distribute your apps.
Safari Developer Program	Free	Create a signing certificate for your Safari extensions. Your membership is set to expire on Dec 31, 2015.

最终，Apple 会把你带向在线商店，付款之后，你就拥有了你的 iOS Developer 资格了。

The screenshot shows the Apple Store checkout interface. At the top, there's a navigation bar with links for Online Store, Mac, iPhone, Watch, iPad, iPod, iTunes, Technical Support, and a search icon. Below the navigation bar, it says "Apple Store". On the right side of the header, there are links for "查找零售店" (Find Retail Store), "学习" (Learn), "企业" (Enterprise), "获得帮助" (Get Help), a user profile for "GengQi", and a shopping cart icon.

The main content area is titled "1 交付选项" (Delivery Options). It shows a product card for the "iOS Developer Program – Membership for one year" at "RMB 688". The card includes a small image of an iPhone and the text "会籍产品". There's also a "编辑" (Edit) button in the top right corner of the card.

Below this, the next step is "2 付款" (Payment). It asks for payment type, showing options for VISA and MasterCard. A checked checkbox says "存储为我的默认付款信息" (Save as my default payment method). A green "继续" (Continue) button is located to the right.

Further down, steps 3 (发票) and 4 (条款与条件) are listed. At the bottom right, it says "订单合计 RMB 688".

比起以前要发电邮注册的时代，现在真是太幸福。

# Product Code 4.1 产品的实现 - Swift

在现代编程，有个概念需要理解清楚——类。这种概念用比较酷的讲法就是——面向对象编程，这些专业名称都非常扯淡，只需要知道是如此约定的读法即可。

## 类——零件

类也不是什么通俗易懂的字眼，如果把 App 看作一个机器人，那么类就像他的零件。不如一起写一个转换阿拉伯数字为中文的类来了解下到底什么是类。

```
class NumberParser {  
}
```

在 Swift 里，一个类就这么简单就完成，并且给他起了个名字叫 NumberParser，但是这东西除了占点空间啥都还做不了，所以我们要教会它第一件事情就是把单个数字字符 0 转换成中文字符 零。

```
class NumberParser {  
  
    func singleNumberToChinese(number:Character) -> String {  
        switch number {  
        case "0":  
            return "零"  
        case "1":  
            return "一"  
        case "2":  
            return "二"  
        case "3":  
            return "三"  
        case "4":  
            return "四"  
        case "5":  
            return "五"  
        case "6":  
            return "六"  
        case "7":  
            return "七"  
        case "8":  
            return "八"  
        case "9":  
            return "九"  
        }  
    }  
}
```

```

    default:
        return ""
    }
}

}

```

func 是 function 的缩写，在这里就是指有件事情叫 singleNumberToChinese，我会给你一个名为 number 的字符（Character），你返回一个字符串（String）给我，如果对于一个字符不知道该怎么处理的情况（比如我给你了一个“一”），你可以直接返回一个空文字给我。

落实到具体做事的方法呢，就是，通过 switch 来对不同的 case 进行处理，number 是 "0"，那么返回（return）“零”，如果不是我们想要的哪几种情况，则返回空字符串。

嗯，这个零件就完成了。

## 实例化

如果你要使用这个零件，那么要做一件事情就是实例化，其实上面写的那些代码其实是一个零件的图纸，要通过实例化变成一个真正的零件。

```

let convertor = NumberParser()
let zeroString = convertor.singleNumberToChinese("0")
println(zeroString)

```

Swift 中，let 用来创建常量，var 用来创建变量。在这里 convertor 以 let 创建，所以它只能是 NumberParser 的类型而不能再改成其他的类型了。Swift 中对于方法的调用非常的简单，即在对象名字后面加上 . 和想要调用的方法名。举例，如果想要使用 convertor 对象的 singleNumberToChinese 方法，可以用 convertor.singleNumberToChinese() 这种方式，在括号里放进去我们要转换的数字，返回的数字就存到 zeroString 里。

最终我们使用 println() 这个 App 本身能理解的函数打印 zeroString 到屏幕上。

明白了这两个基础的概念，我们就可以开始“搬砖”了。

## 类的继承

类的继承就是继承上一代的功能，改造或衍生出新功能。

假如我们需要一个 NumbersParser 那么可以这么写

```
class NumbersParser: NumberParser {

    func numberToChinese(number:Int) -> String {
        var numbers = Array(String(number))
        var finalString = ""
        for singleNumber in numbers {
            var string = singleNumberToChinese(singleNumber)
            finalString = "\(finalString)\(string)"
        }
        return finalString
    }

}
```

你这下是不是看晕了？ NumbersParser 和 NumberParser 没区别啊？ 仔细看看，中间有个 s 的区别。

这是非常好的一个命名问题的例子，千万不要采用这种非常像的名字来命名不同的类，你会把自己搞晕的。

从另外一个角度，再遇到不明 Bug 的时候，一定要仔细看你是不是打错了名字了。

此时这个 NumbersParser 就可以利用 NumberParser 的方法来实现 numberToChinese。

for singleNumber in numbers 这种，for x in y 的形式的方法是逐个取出 y 中的元素，命名为 x。

Swift 使用 "\(finalString)\(string)" 这种方式来组成字符串。

例如

```
var name = "牛奶"
println("今天早上我有喝 \(name)")
```

就可以得到结果 —— “今天早上我有喝牛奶”

```
let convertor = NumbersParser()
```

```
let yearString = convertor.numberToChinese(2015)
println(yearString)
```

此时就可以得到我们想要的结果 —— 二零一五

## 单例

像 NumbersParser 这种类你可能会在程序里多处使用，如果每次使用的时候，都 let convertor = NumbersParser() 一下就显得多余。就像现实场景一样，你要在每个房间里都可以听歌不见得需要每个房间都放一台音响，你可以搬过去。

单例就是为此而生。

给类增加一个这样的方法

```
class NumbersParser: NumberParser {
    static let sharedInstance = NumbersParser()
    ...
}
```

这样你每次使用的时候，就可以简化几步

```
let numbersString = NumbersParser.sharedInstance.numberToChinese(2015)
println(numbersString)
```

单例有诸多便利之处，像这种负责单纯的单位转换场景的类非常适合用单例。不过也有其滥用的危害，在被坑到之前，你并不需要过多在意。

## Override

这也是一个极为有趣的功能，例如你想利用上面的方法实现一个 YearParser，输入 2015 可以输出 二零一五年，那么可以这么做

```
class YearParser: NumbersParser {
    static let sharedInstance = YearParser()

    override func numberToChinese(number:Int) -> String {
        var numbersString = super.numberToChinese(number)
        return "\u{20e3}(numbersString)年"
    }
}
```

```
    }  
}
```

`override` 的意思是说明要对某个方法进行改造。`super.numberToChinese(number)` 可以使用改造前的方法，得到旧的结果，随后用 `"\($numbersString)年"` 格式化了结果

因此

```
let yearString = YearParser.sharedInstance.numberToChinese(2015)  
println(yearString)
```

得到的 `yearString` 结果就是 —— “二零一五年”。

`YearParser` 里对于 `NumbersParser` 的改造并不会影响 `NumbersParser` 的结果，所以

```
let numbersString = NumbersParser.sharedInstance.numberToChinese(2015)  
println(numbersString)
```

得到的结果依旧是 —— “二零一五”

如果你希望全盘改造，就不要使用 `super.numberToChinese(number)` 方法，完全重写一个即可。

## 深入学习 Swift

[The Swift Programming Language](#)

[Swifter - 100 个 Swift 必备 tips](#)

# Product Code: 产品的实现 - iOS App 是什么

---

我想这可能是一切 Magic 和困惑的开始，iOS App 从哪里开始？传统的 Objective-C 项目要更低层一些，但是从 Swift 开始，已经隐藏了部分底层细节。既然我们用 Swift 开发生涯，那此刻对我们来说，了解那么多徒增困扰，不如先把鸭子煮熟吧。

## 创建第一个 App

---

创建第一个 App，这是一个又兴奋又激动的过程，虽然过程中会有很多你不明白的地方，没关系，后面我们会逐一了解。

1. 打开 Xcode，点击 Create a new project

X



# Welcome to Xcode

Version 6.3 (6D554n)



## Get started with a playground

Explore new ideas quickly and easily.



## Create a new Xcode project

Start building a new iPhone, iPad or Mac application.



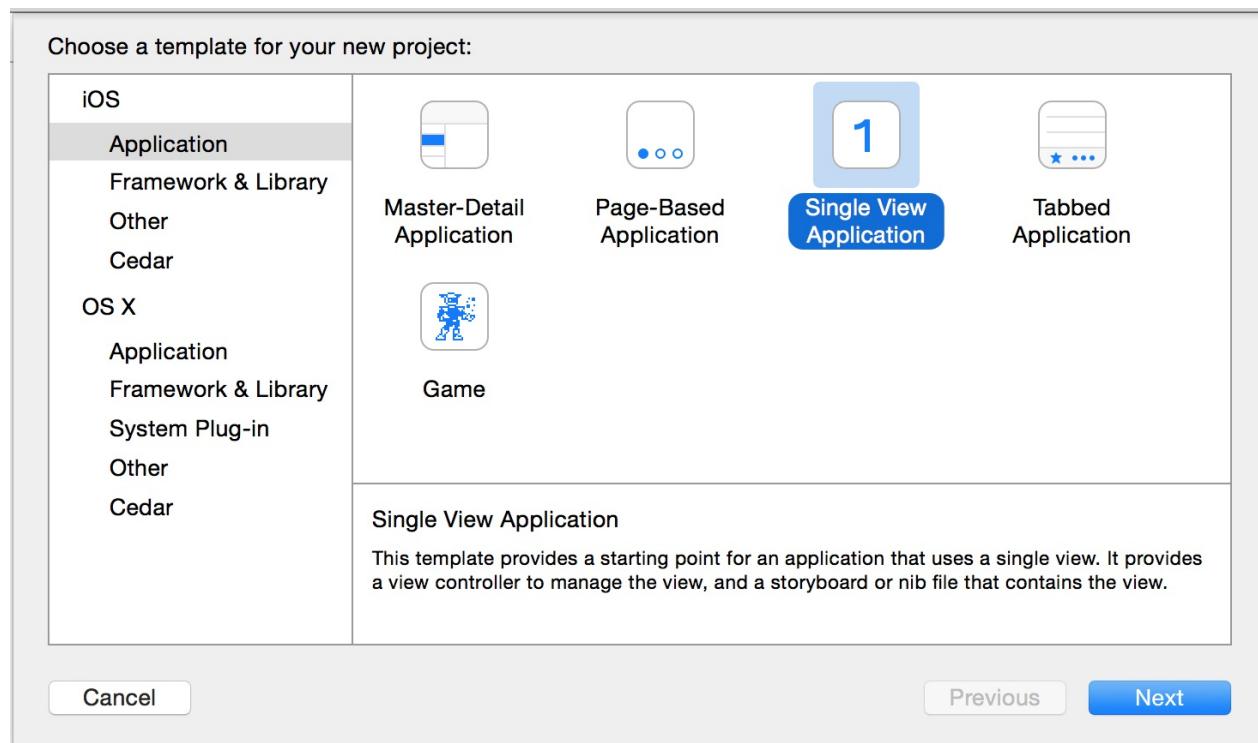
## Check out an existing project

Start working on something from an SCM repository.

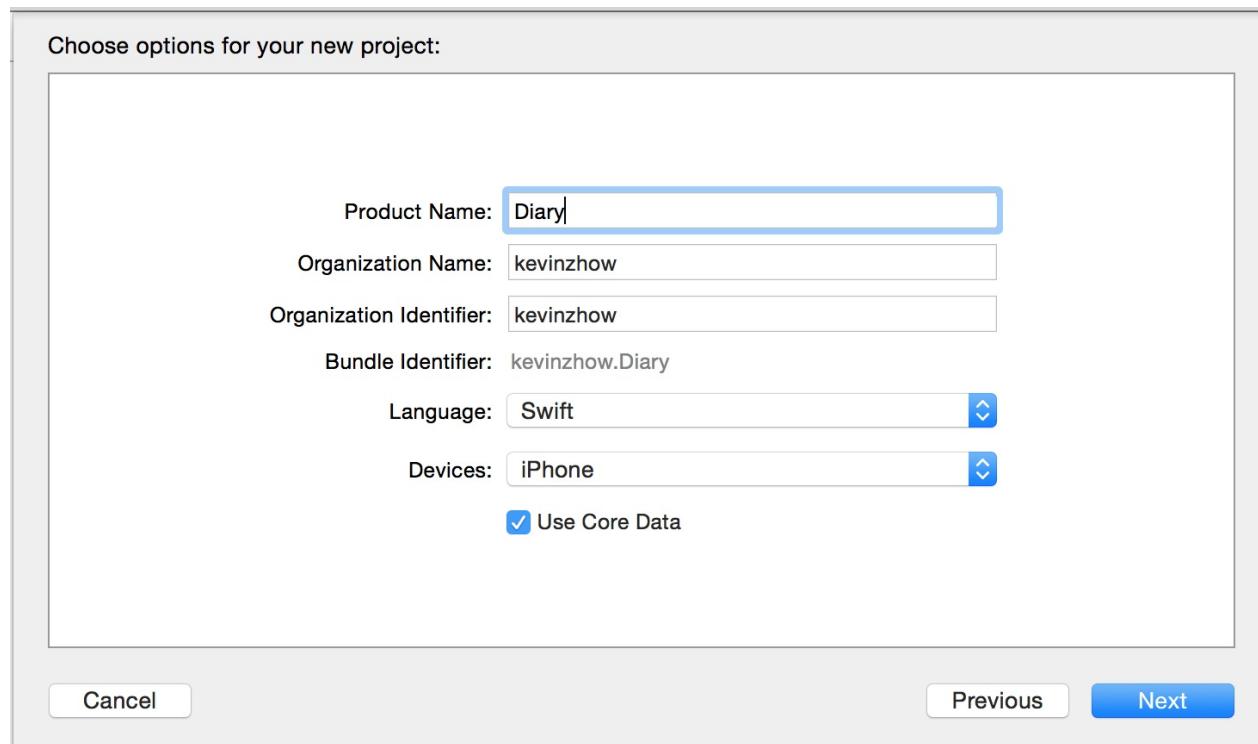


Show this window when Xcode launches

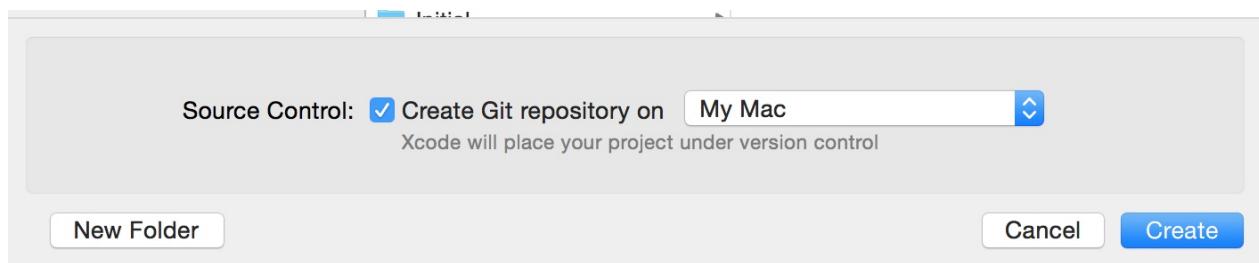
2.选择 Single View Application 创建一个空的 App



### 3. 设置 App 的基本信息，选择 Swift 以及勾选 Core Data



### 4. 启用 Git 来管理代码



## AppDelegate

创建完你的第一个 Swift App 之后，当你按下 Command ⌘ + R，你的 App 就会从这个文件这个函数开始。

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:  
    // Override point for customization after application launch.  
    return true  
}
```

可能这已经让人足够困惑了，它的参数都表示什么？先不管它们的形式，这句话翻译成汉语就是你的程序在问你“程序完成启动了吗？”。

请带着你对这几行英文的思考，先听我讲个故事。

Once upon a time，在程序世界最开始的时候，那个世界是很单纯的，你告诉程序你想知道  $10000 + 2399$  是多少，他计算一番后告诉你它是  $12399$ ，后来他们记忆能力有所发展，你可以告诉他， $a$  是  $10000$ ,  $b$  是  $2399$ ，当你问他  $a + b$  是多少的时候，他能知道你在问  $10000 + 2399$ ，于是也可以告诉你  $12399$ 。

再到后来，他们变的更聪明了，你可以告诉计算机，把两个数字算到一起这个事情叫做加法，如果我告诉你  $a$  和  $b$ ，让你用加法处理一下，你就把它们加起来。

他记住了，为了让这件事看起来很酷，做一件事情被命名为函数。 $a$  和  $b$  这种某个数值的代号被命名为变量。

单纯的世界开始变得复杂，因为一切都要看起来听起来很酷。

当你进入这个世界的时候，这个世界已经发展出了一种东西叫做约定。

每一个 App 都会和你签订下契约。

你不再需要从零开始教计算机做每一件事情，绝大部分事情都已经随着操作系统这个大脑记录进去，而手机 App 更像这个操作系统的房子，你的角色更像回答 App 一切的问题的老师。

1. 当我启动的时候，我该做什么？
2. 我启动完毕了，我该显示什么？
3. 用户用手指戳了我这里，我是该做什么？
4. 我做完了，然后呢？

你需要做的就是如何严谨完美地回答这些问题，并提供一个清晰的思路给它。

`AppDelegate` 的意思，就是你要在这个文件里解答 App 一些最基础的问题，App 会在这里问：“我该不该启动？”、“我进入后台了该不该干什么？”、“我要被系统大人关闭了，我该怎么办？”等等之类的问题。

所以你应该明白刚才那个问题了。

你第一个签定了契约的 App 问你，我是不是可以完成启动啦？

如果你简单地回答：可以，那么它就会跑到 StoryBoard 里面寻找故事的开始，进入下一个充满有趣问题的世界。

为什么呢？因为这都是约定。

## View Controller

每一个界面一般来说都会对应一个 View Controller。



Main.Storyboard 里的 View Controller 界面通过右上角的 Custom Class 指定了使用 ViewController 作为大脑。他会寻找 ViewController 这个类。而 ViewController 就定义在左边的 ViewController.swift 里面。

```
import UIKit
```

```

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}

```

你可以按着键盘的 option 点击 viewDidLoad，来看看这个莫名其妙的东西是什么。

Called after the controller's view is loaded into memory.

我想你可能更好奇了，这到底是做什么用的，让我们来做一个小实验。

在 super.viewDidLoad() 的后面插入一句

```
self.view.backgroundColor = UIColor.blackColor()
```

这时你再次运行 App，App 的界面背景就变成了黑色。

`viewDidLoad` 在界面准备完毕后执行，在这里你可以做一些界面 UI 的准备，比如创建按钮啦，添加图像啦。相对应的，还有 `viewWillAppear` `viewDidAppear`。

举个例子，微信的聊天对话列表界面，首先会执行 `viewDidLoad` 准备这个界面，当界面准备就绪可以显示的时候，`viewWillAppear` 会执行，界面显示出来后，会执行 `viewDidAppear`，这时候你点击了其中一个对话，然后再返回到对话列表，这时候 `viewWillAppear` 和 `viewDidAppear` 会再次执行，但是 `viewDidLoad` 不会再次执行。

## 深入了解 iOS App

[About iOS App Architecture](#)

## About the iOS Technologies

# Product Code: 产品的实现 - 在年之外

## UILabel

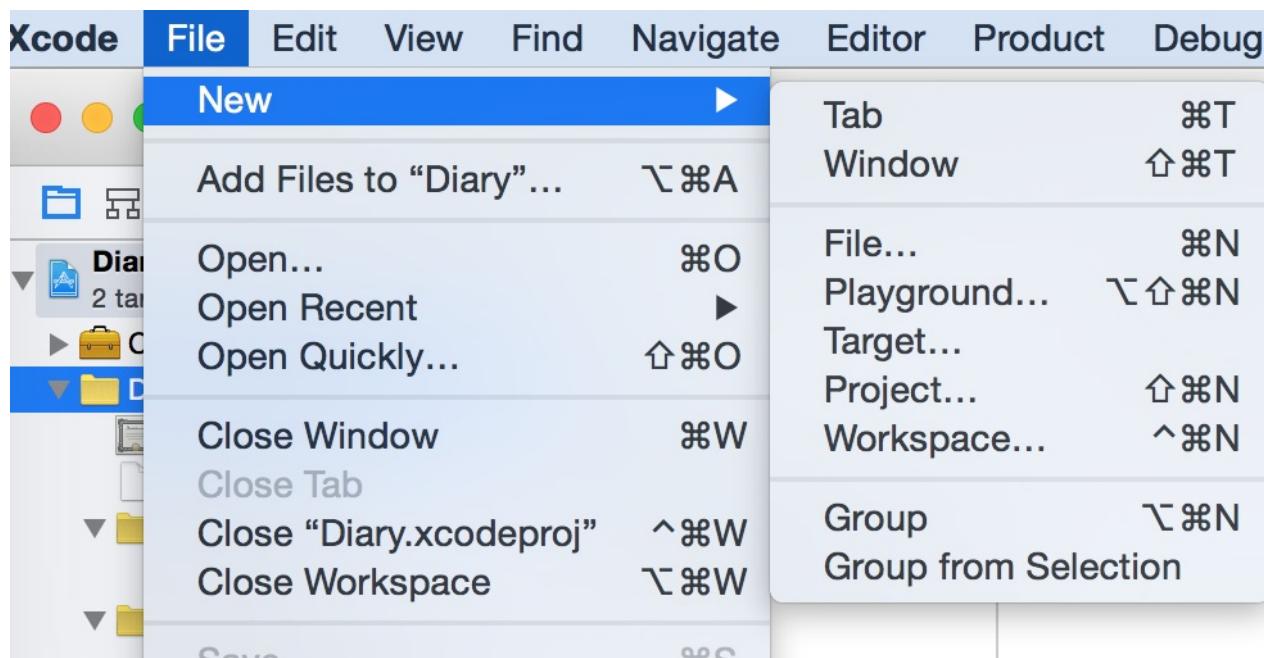
竖排是我们第一个要攻克的问题，这个问题可大可小，往大了讲就是实现一个可以竖着显示，竖着编辑，竖着选择的体验和横着完全一样的控件，小了说就是一个竖着显示文字的控件。

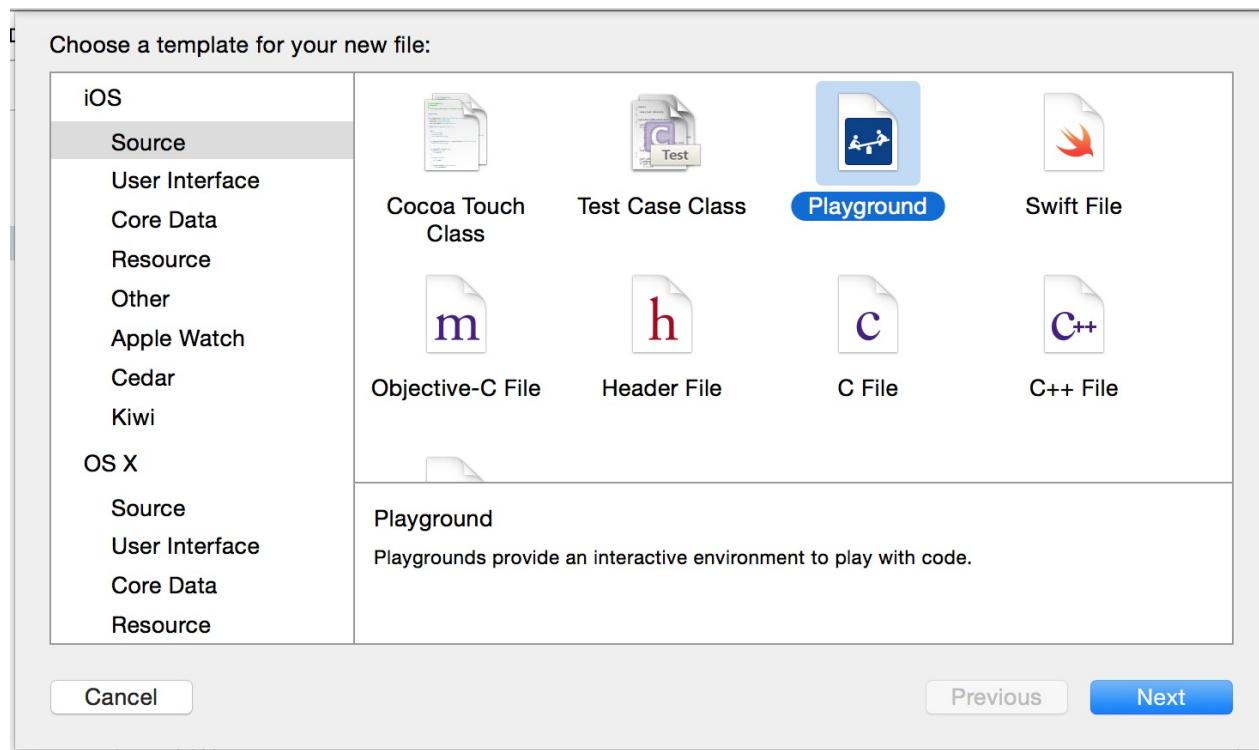
那么，在小记里，大的地方就是查看的界面，小的地方就是显示年份的这种小 Label。

在 iOS 里，UI 方面的控件都在一个叫 UIKit 的 Framework 里面，UIButton，UITextField，UITextView 等等，这些控件各有特性，但是都基于 UIView 衍生出来，需要你慢慢地挨个熟悉。

我们先来搞定 UILabel。

新建一个 Playground，这是 Swift 时代新引入的一个东西，可以实时查看代码的效果。





然后在里面输入如下代码

```
import UIKit

// 引用 UIKit 这样才能找得到 UILabel

var newLabel = UILabel(frame: CGRectMake(0, 0, 300, 100))

// 创建新的 UILabel，并且设置长宽为 100 和 300

newLabel.text = "HeyLabel"

// 设置 Label 的文字

newLabel.sizeToFit()

// 使 Label 的大小自动适应文字的大小

newLabel
```

这时候你就可以在右边点小眼睛查看到这个控件的外观



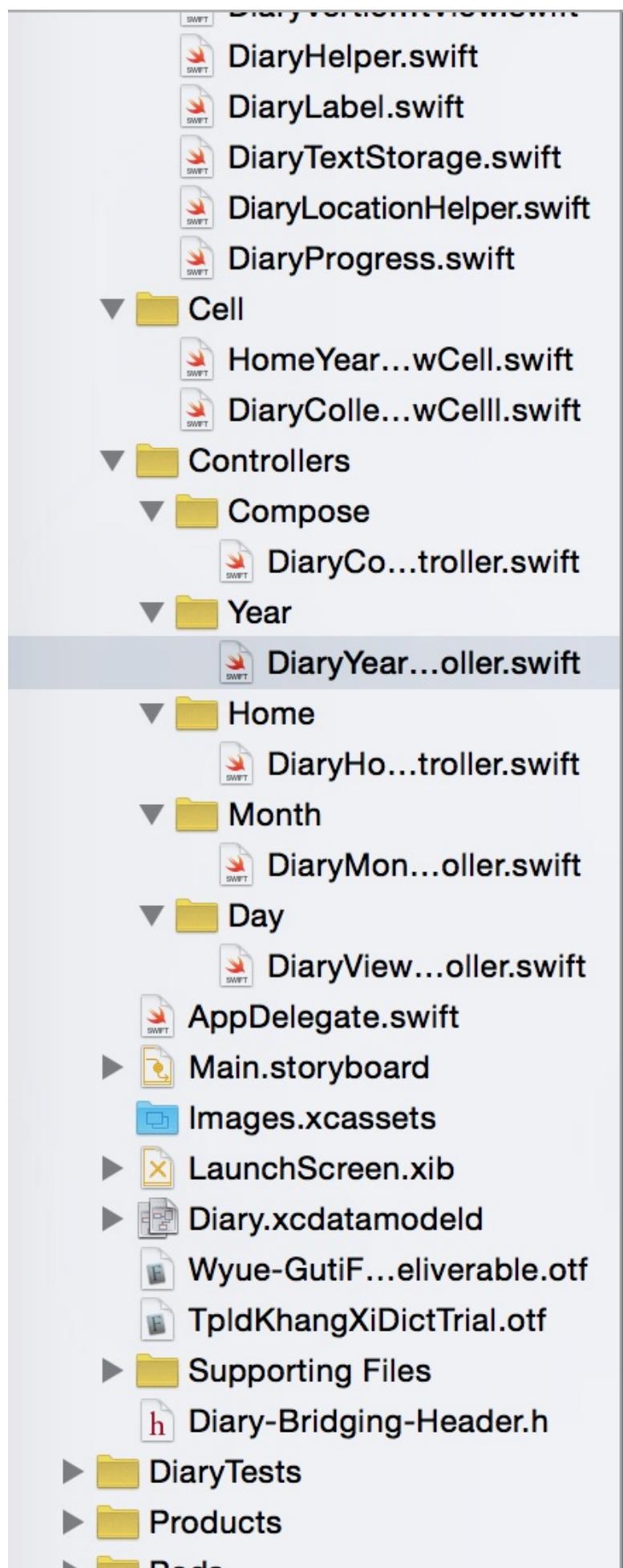
任重而道远，开心一下继续下面的任务，就是让它竖着显示。这里我们采用的方法比较取巧

1. 设置 Label 的宽度为文字单个字符的宽度
2. 高度为全部文字叠加起来的高度
3. 自动换行

这样文字就会竖着显示了。

首先我们要添加两个字体，[文悦字型](#) 提供了我们需要的康熙字典体和文悦聚珍仿宋。

下载后把字体拖到 Xcode 的项目里。



然后修改项目的配置文件 Supporting Files/Info.plist，右键在 Fonts provided by application 增加两行 String，分别为字体的文件名。



最后你需要知道这里两个字体在应用里的正式名字是什么，方法很简单，在 Finder 里面双击安装这个字体，然后你就能在 PostScript 看到它的名字。

### TypeLand KhangXi Dict Trial Regular

TypeLand KhangXi Dict Trial Regular

PostScript 名称 TpIdKhangXiDictTrial  
 全名 TypeLand 康熙字典體試用版  
 系列 TypeLand 康熙字典體試用版  
 样式 Regular  
 种类 OpenType PostScript

有了字体，就可以正式开始竖排 Label 的创建啦。

```
let font = UIFont(name: "TpIdKhangXiDictTrial", size: 20) as UIFont!
// 创建 康熙字典体

let textAttributes: [NSObject : AnyObject] = [NSFontAttributeName: font]
// 定义一个字体样式 [key: value] 的写法是一种映射，意思是 key 的值就是后面的 value，在这里的意思

var newLabel = UILabel()

newLabel.attributedText = NSAttributedString(string:
    labelText, attributes: textAttributes)
// 设置带有字体样式的文字

newLabel.lineBreakMode = NSLineBreakMode.ByCharWrapping
// 以字符为换行标准

newLabel.numberOfLines = 0
// 允许多行
```

为了实现这一点，我们要写一个函数，来计算竖着显示这些文字的大小是多少

```
func sizeHeightWithText(labelText: NSString,
    fontSize: CGFloat,
    textAttributes: [NSObject : AnyObject]) -> CGRect {
    return labelText.boundingRectWithSize(CGSizeMake(fontSize, 480),
        options: NSStringDrawingOptions.UsesLineFragmentOrigin,
        attributes: textAttributes, context: nil)
}
```

`NSString` 这个类有一个自带的 `boundingRectWithSize` 方法，这个方法可以计算在给定的大小里填充进文字后的实际大小是多少，`CGSizeMake(fontSize, 480)` 这个参数给了一个宽度是文字的宽度，高度是 480 的长方形大小。

`NSStringDrawingOptions.UsesLineFragmentOrigin` 设定了文字的排列对齐的方式，`textAttributes` 里存放了文字的字体以及行高宽之类的信息。

那么我们修改下 `Label` 的面积大小。

```
let font = UIFont(name:
    "TpldKhangXiDictTrial", size: 20) as UIFont!

let textAttributes: [NSObject : AnyObject] =
    [NSFontAttributeName: font]

var labelSize = sizeHeightWithText(labelText,
    fontSize ,textAttributes)
// 获取文字竖排的大小

var newLabel = UILabel(frame: labelSize)
// 以 labelSize 的大小创建 Label

newLabel.attributedText = NSAttributedString(string: labelText,
    attributes: textAttributes)

newLabel.lineBreakMode = NSLineBreakMode.ByCharWrapping
newLabel.numberOfLines = 0
```

这时，你的 `newLabel` 就完成了竖排的任务



接下来，独立成一个单独的类方便重复使用，简称复用。

```
class DiaryLabel: UILabel {

    var textAttributes: [NSObject : AnyObject]!

    convenience init(fontname:String,
                     labelText:String,
                     fontSize : CGFloat,
                     lineHeight: CGFloat){

        self.init(frame: CGRectZero)

        let font = UIFont(name: fontname,
                          size: fontSize) as UIFont!

        var paragraphStyle = NSMutableParagraphStyle()
        paragraphStyle.lineSpacing = lineHeight

        textAttributes = [NSFontAttributeName: font,
                         NSParagraphStyleAttributeName: paragraphStyle]

        var labelSize = sizeHeightWithText(labelText,
                                           fontSize ,textAttributes)

        self.frame = CGRectMake(0, 0, labelSize.width,
                               labelSize.height)

        self.attributedText = NSAttributedString(string: labelText,
                                                attributes: textAttributes)
        self.lineBreakMode = NSLineBreakMode.ByCharWrapping
        self.numberOfLines = 0
    }

    func resizeLabelWithFontName(fontname:String,
                                labelText:String,
                                fontSize : CGFloat,
                                lineHeight: CGFloat ){
        let font = UIFont(name: fontname, size: fontSize) as UIFont!

        var paragraphStyle = NSMutableParagraphStyle()
        paragraphStyle.lineSpacing = lineHeight

        textAttributes = [NSFontAttributeName: font,
                         NSForegroundColorAttributeName: UIColor.blackColor(),
                         NSParagraphStyleAttributeName: paragraphStyle]
    }
}
```

```

var labelSize = sizeHeightWithText(labelText, fontSize, textAttributes)

self.frame = CGRectMake(0, 0, labelSize.width, labelSize.height)

self.attributedText = NSAttributedString(string: labelText,
attributes: textAttributes)

self.lineBreakMode = NSLineBreakMode.ByCharWrapping
self.numberOfLines = 0
}

func updateText(labelText: String) {

var labelSize = sizeHeightWithText(labelText,
self.font.pointSize,
textAttributes)

self.frame = CGRectMake(0, 0, labelSize.width, labelSize.height)

self.attributedText = NSAttributedString(string: labelText,
attributes: textAttributes)
}

func updateLabelColor(color: UIColor) {

textAttributes[NSForegroundColorAttributeName] = color

self.attributedText = NSAttributedString(
string: self.attributedText.string,
attributes: textAttributes)
}

}

```

这里我们拆分出来几个方法，`resizeLabelWithFontName` `updateText` `updateLabelColor` 方便我们以后灵活地更新这个 Label 的内容。

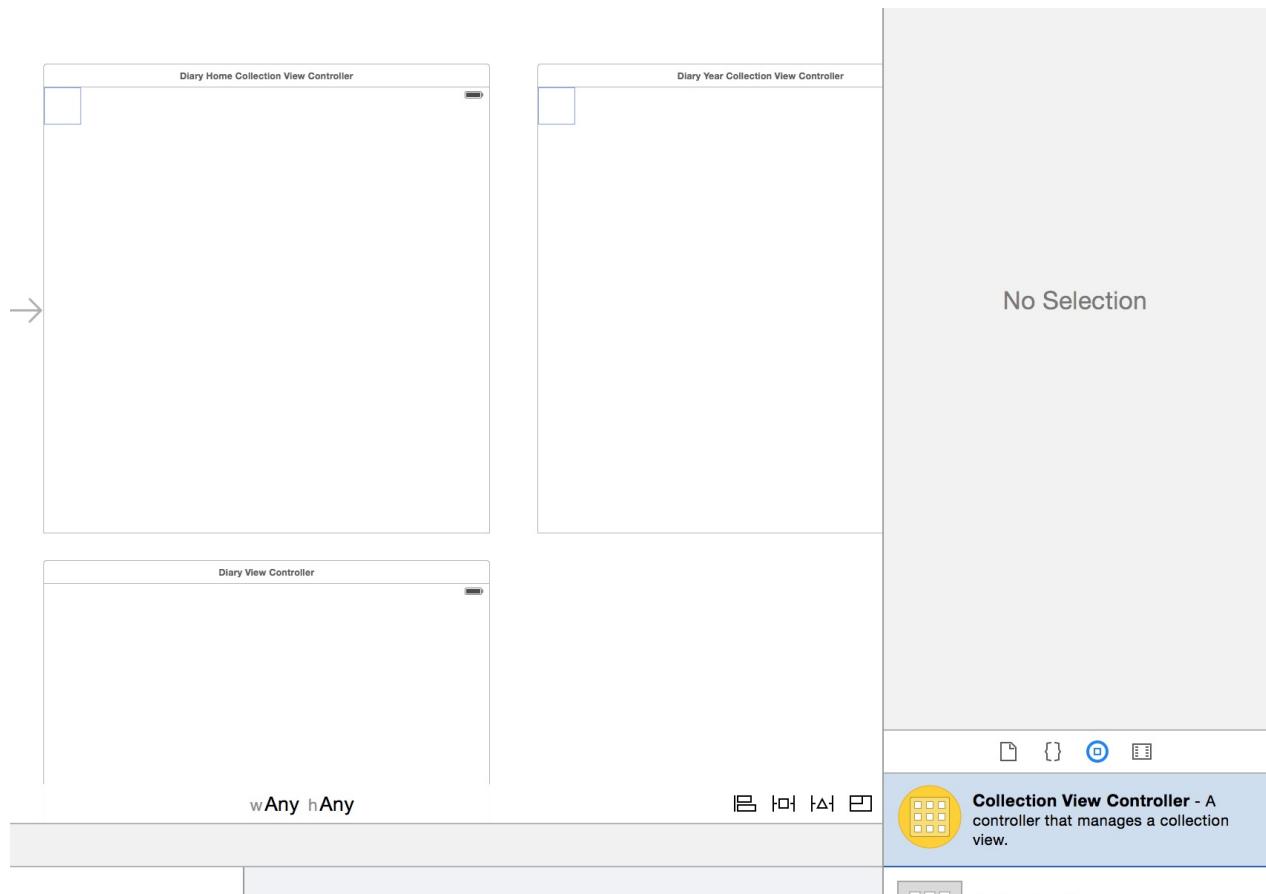
这种行为就叫做重构，是用来消磨时间，增加代码的可维护性，提升满足感的好活动。

## UICollectionView

回顾年的设计，中间的年份并不是一成不变的，可能出现二零一六年，二零一七年。所以我使用 UICollectionView 在这里实现动态的显示。

UICollectionView 预设了一套非常先进的显示逻辑，这点你在 iOS 的相册里已经体验到了，当你拍摄了更多的相片，相片会按照日期排列好，你可以通过滑动查看不同的相片，这都是 UICollectionView 的功劳。

打开 Xcode 里的 Main.StoryBoard，选中默认 View Controller 按 Delete 删掉，然后在右下角拖进去 Collection View Controller



这个 Collection View Controller 默认并没有关联任何大脑。我们需要新建一个 HomeCollectionViewController 作为大脑。

Choose a template for your new file:

iOS	 Cocoa Touch Class	 Test Case Class	 Playground	 Swift File
	 Objective-C File	 Header File	 C File	 C++ File
OS X	 Cocoa Touch Class A Cocoa Touch class.			

Cancel Previous Next

Choose options for your new file:  
The implementation language

Class:

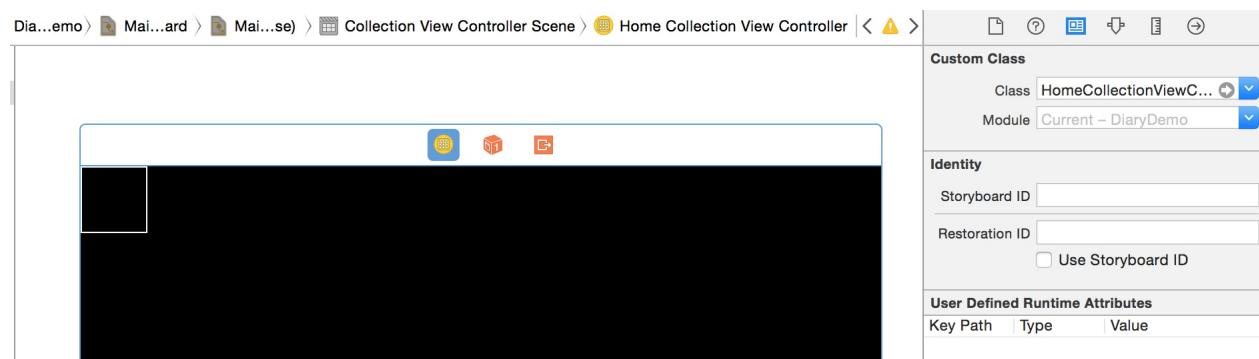
Subclass of:

Also create XIB file

Language:

Cancel Previous Next

UICollectionViewController 是苹果为我们预先准备的大脑，它可以自动关联 UICollectionView，在 StoryBoard 里修改 Custom Class 为 HomeCollectionViewController 就可以使两者关联。



UICollectionView 有两个极为重要的概念，一个是 `DataSource`，它为 `UICollectionView` 提供数据。另一个是 `Delegate`，它为 `UICollectionView` 处理交互逻辑。

在这里，`DataSource` 和 `Delegate` 会被自动设置为 `HomeCollectionViewController`，因为 `HomeCollectionViewController` 继承自 `UICollectionViewController`，这是 App 和我们的一个小约定。

## UICollectionView DataSource

`UICollectionView` 会问 `DataSource` 几个问题。

1. `numberOfSectionsInCollectionView` —— 显示几个栏目？
2. `numberOfItemsInSection` —— 每个栏目里有几个项目？
3. `cellForItemAtIndexPath` —— 项目需要长成什么样子？

对于第一个问题，我们只显示年，所以在应答里返回 1 即可

```
override func numberOfSectionsInCollectionView
(collectionView: UICollectionView) -> Int {
    return 1
}
```

对于第二个问题，我们此时只有 2015 年，所以可以如下作答

```
override func collectionView
(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    return 1
}
```

而第三个问题就略微复杂一些，我们的回答模式如下，但是并不完整

```

override func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) -> HomeYearCollectionViewCell {
    let cell = collectionView.dequeueReusableCellWithReuseIdentifier("HomeYearCollectionCell", forIndexPath: indexPath) as! HomeYearCollectionViewCell

    cell.textInt = 2015
    cell.labelText = "二零一五 年"

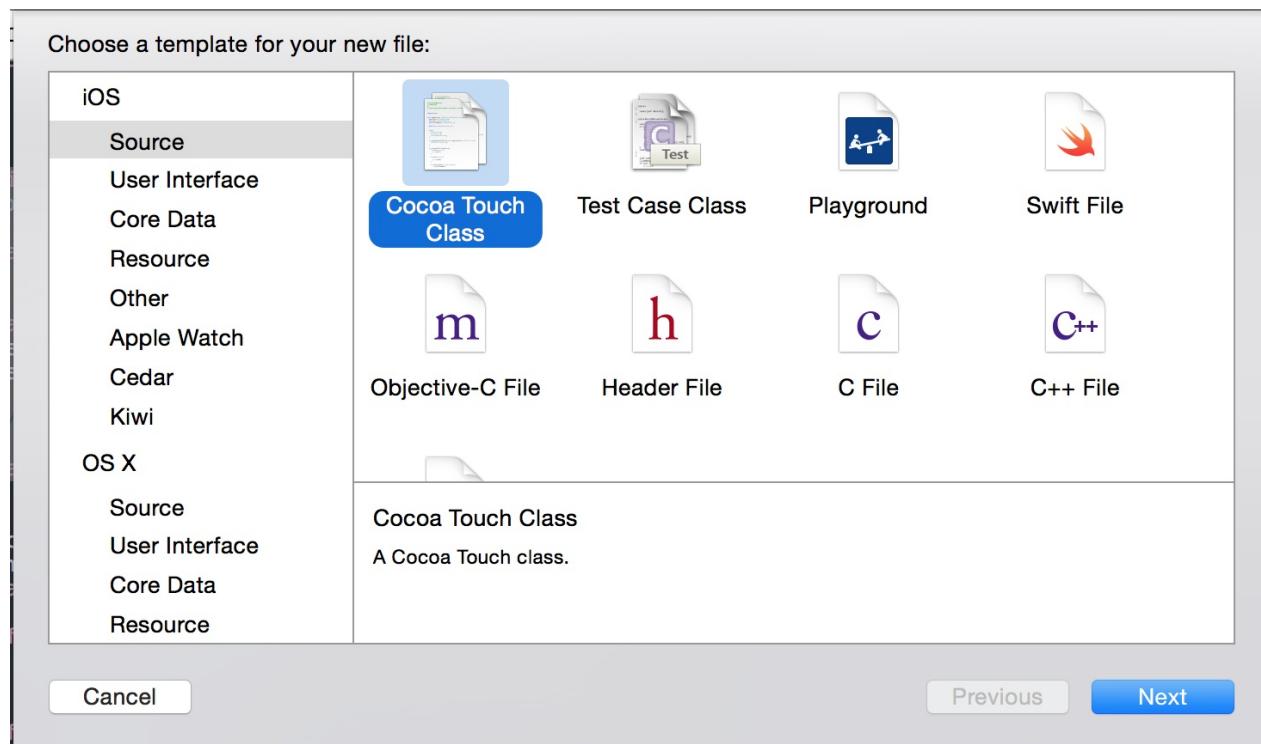
    return cell
}

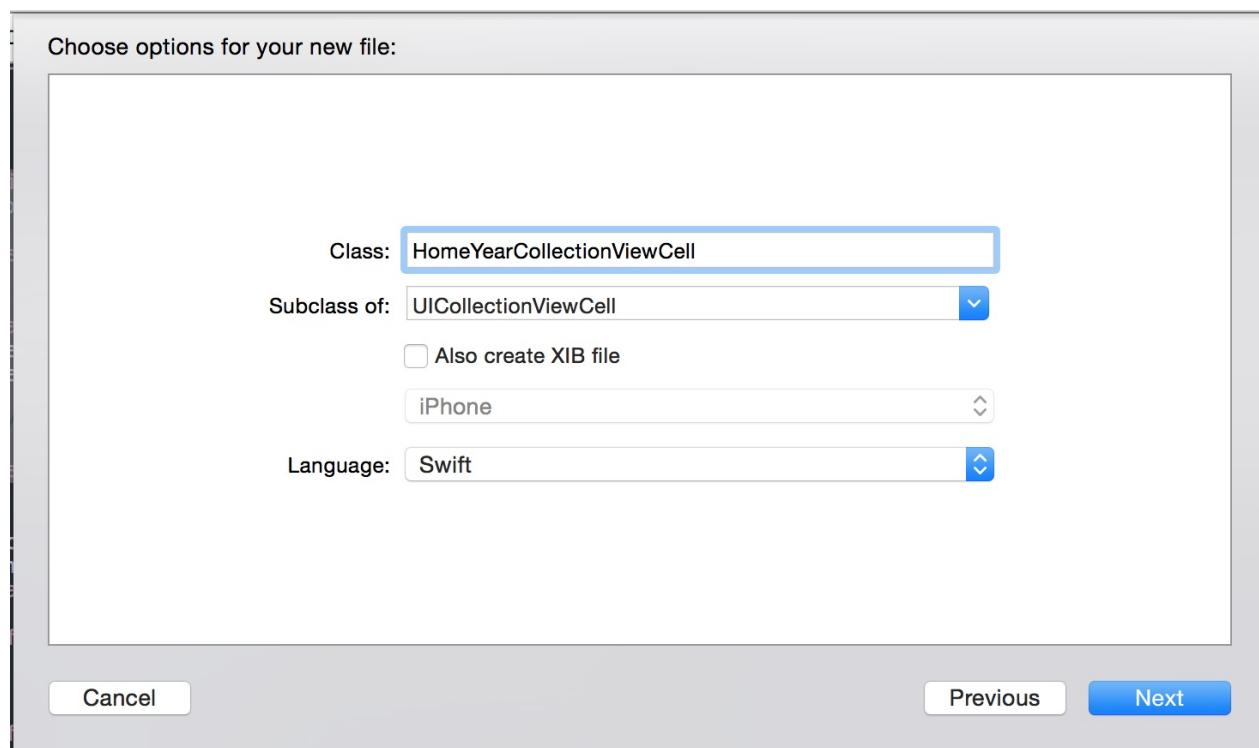
```

因为我们还没有创建 HomeYearCollectionViewCell，Cell 是一种针对数据的模版。在 App 里有很多样式相同，但是数据不同的界面，例如聊天界面的气泡，气泡的样式都是一样的，但是文字内容不同。

而气泡就是一个 Cell，我们需要做的就是把数据填充进去。

先创建一个 HomeYearCollectionViewCell





在 HomeYearCollectionViewCell 里，我们如下编写

```

var.textLabel: DiaryLabel!
var textInt: Int = 0
var labelText: String = "" {
    didSet {
        self.textLabel.updateText(labelText)
    }
}

override func awakeFromNib() {
    self.textLabel = DiaryLabel(fontname: "TpIdKhangXiDictTrial",
        labelText: labelText,
        fontSize: 16.0,
        lineHeight: 5.0)

    self.addSubview(textLabel)
}

override func layoutSubviews() {
    super.layoutSubviews()

    self.textLabel.center = CGPointMake(itemWidth/2.0, 150.0/2.0)
}

```

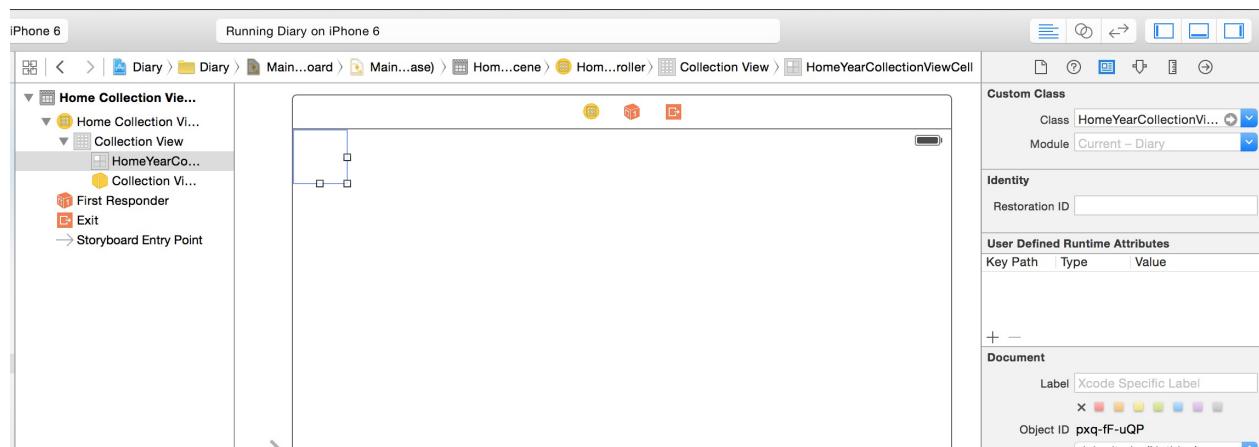
流程很简单，awakeFromNib 的时候，添加了一个 `TextLabel`，当更新 `labelText` 的时候，通过 `didSet` 事件来用 `updateText` 方法更新文字以重新计算 Label 的大小。在最终显示前，通过 `layoutSubviews` 来重新计算 `TextLabel` 的中心。

`awakeFromNib` `layoutSubviews` 这些是哪里来的以及我是怎么知道它们的用法的呢？

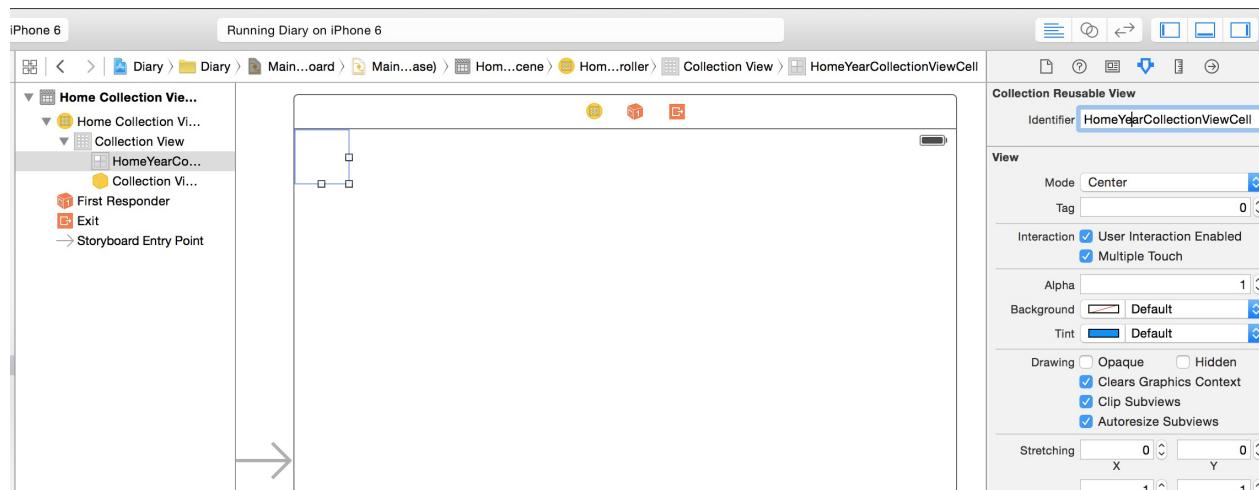
这些当然都是要你从不停地看各种枯燥无味的基本文档中练出来。

`textInt` 存储了年份的整数形式。

完成了我们的 Cell 后，回到我们的 Storyboard 里找到刚才的 UICollectionView，点击里面的小正方形，把 Custom Class 设置成 HomeYearCollectionViewCell



同时，我们也需要把 Identifier 也设置成 HomeYearCollectionViewCell



## 全局常量

在 App 里有些参数是需要反复使用的，例如上面的 Cell 在计算的时候，写了很多临时的数字，那么明显存在某个地方可以方便统一管理，在 AppDelegate 的最顶端，我放置了这些常量

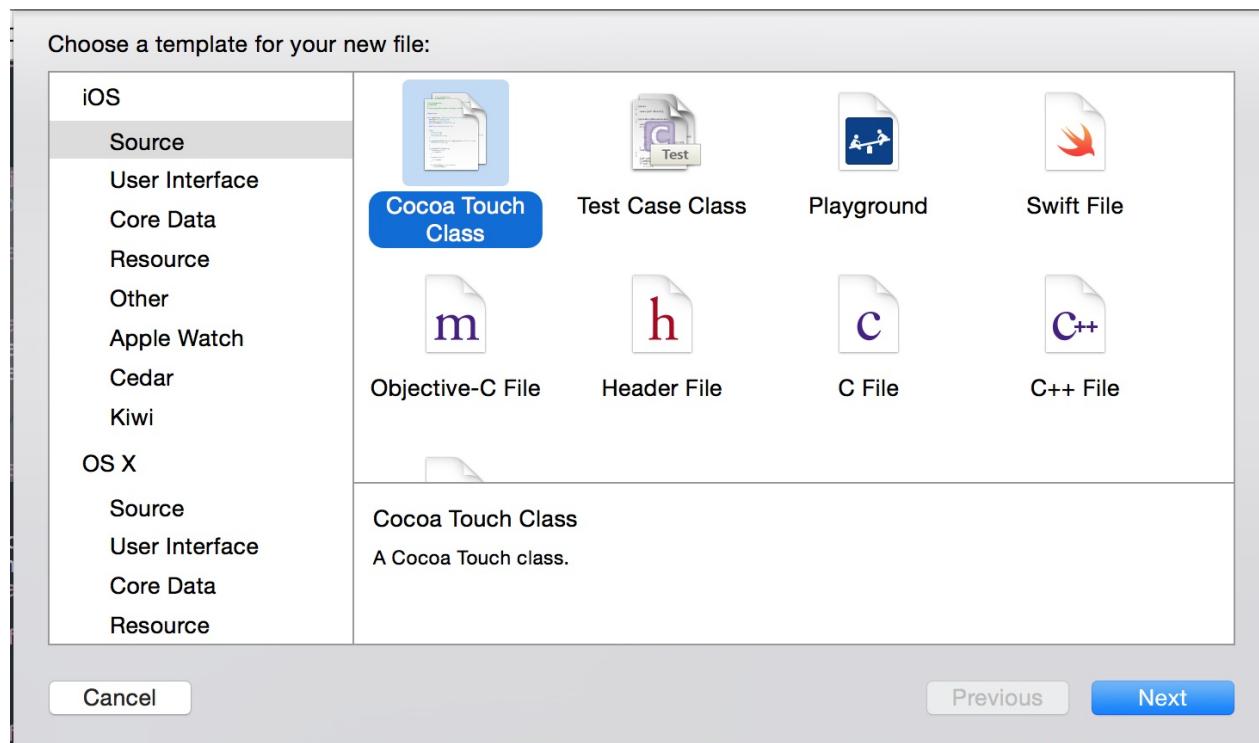
```
let itemHeight: CGFloat = 150.0  
  
// Cell 的高度  
  
let itemWidth: CGFloat = 60  
  
// Cell 的宽度  
  
let collectionViewWidth = itemWidth * 3  
  
//同时显示三个 Cell 时候
```

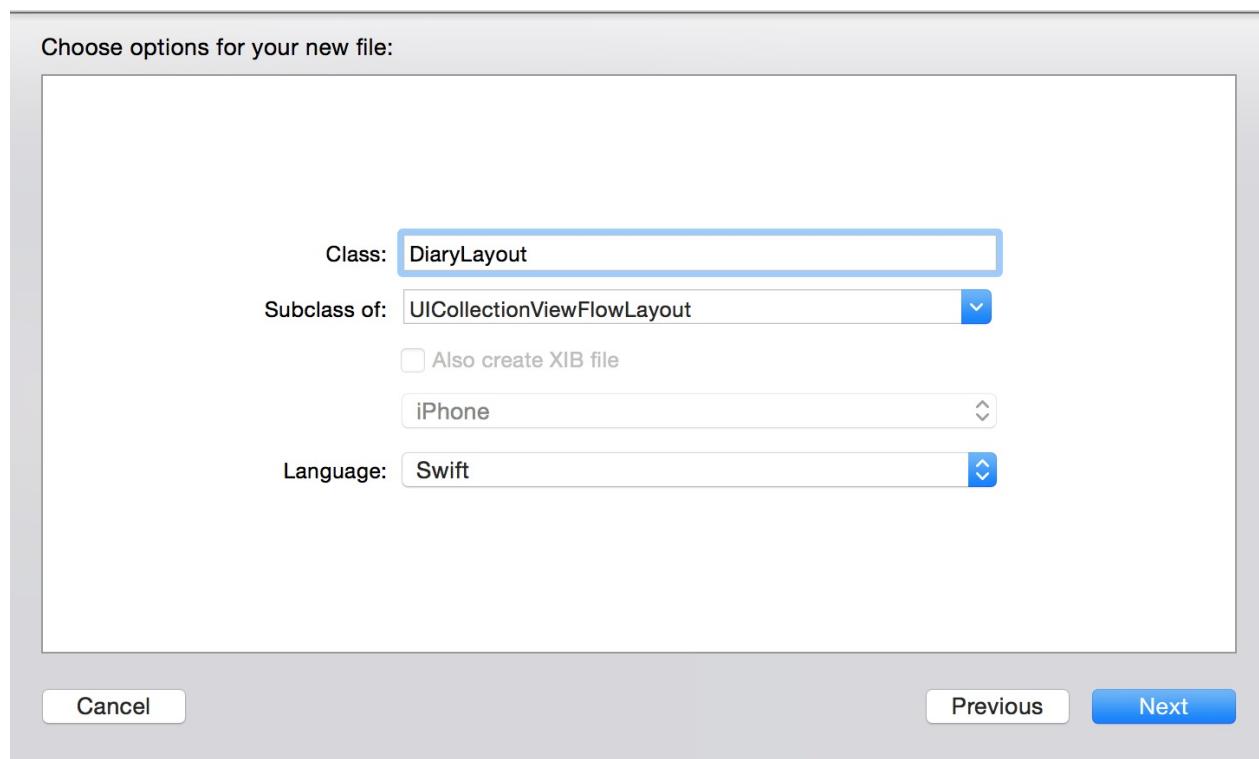
那么像 `self.textLabel.center = CGPointMake(30/2.0, 150.0/2.0)` 这种计算，就可以写成 `self.textLabel.center = CGPointMake(itemWidth/2.0, itemHeight/2.0)`

生活顿时容易多了。

但是到这里，CollectionView 还不知道用什么样的布局显示出来这些 Cell。

So Go For The Last Question





```
class DiaryLayout: UICollectionViewFlowLayout {
    override func prepareLayout() {
        super.prepareLayout()
        let itemSize = CGSizeMake(itemWidth, itemHeight)
        self.itemSize = itemSize
        self.minimumInteritemSpacing = 0.0
        self.minimumLineSpacing = 0
    }
}
```

这几个就是如字面意思了，Cell 的大小设置为 宽度 30.0 高度 150.0，item 之间不留空，Cell 的行之间，空 30。

在 ViewDidLoad 里面，把 Layout 信息给 CollectionView

```
var yearLayout = DiaryLayout()

yearLayout.scrollDirection = UICollectionViewScrollDirection.Horizontal
self.collectionView?.setCollectionViewLayout(yearLayout, animated: false)
```

这时候运行一下，你就可以看到屏幕显示着二零一五年了。

但是这个年并不是居中的，怎么居中呢？

我的思路是把 collectionView 按照三个 Cell 的宽度和高度重置 frame 然后给 collectionView 做一下 inset。

在 ViewDidLoad 里面

```
self.collectionView!.frame = CGRect(x:0, y:0,  
width: collectionViewWidth, height: itemHeight)  
  
self.collectionView!.center = CGPoint(x: self.view.frame.size.width/2.0,  
y: self.view.frame.size.height/2.0)
```

collectionView 有个方便的方法可以设置 inset

```
func collectionView(collectionView: UICollectionView,  
layout collectionViewLayout: UICollectionViewLayout,  
insetForSectionAtIndex section: Int) -> UIEdgeInsets {  
  
    var leftRightMargin = (collectionViewWidth - itemWidth)/2  
    return UIEdgeInsetsMake(0, leftRightMargin, 0, leftRightMargin);  
}
```

最后，按下 Xcode 左上角的播放键，就可以运行你的 App 了。

你可以[在这里](#)找到本章节的完成代码

## 深入理解 View Controller

[View Controller Programming Guide for iOS](#)

# Product Code: 产品的实现 - 使用 Git 管理你的代码

---

可能你已经迫不及待要进行下一部分的开发了，但是且慢！有一件很重要的事情此时需要你来做一下——用 Git 管理你的代码。

## Git 是啥？

---

Git 是由 Linux 之父 Linus Tovalds 为了更好地管理linux内核开发而创立的分布式版本控制／软件配置管理软件。

简单来说，Git 是一个管理你的代码的历史记录的工具。

在很久很久以前，管理代码的方式曾经非常老土，就是通过复制文件夹到不同的地方来备份，管理不同的版本——你是不是这么干过？

从 Git 开始你就可以摆脱丢代码、寻找历史记录这些麻烦了，到 [Github / Gitcafe](#) 注册你的账号，立刻步入现代化！

## 安装 Git 客户端

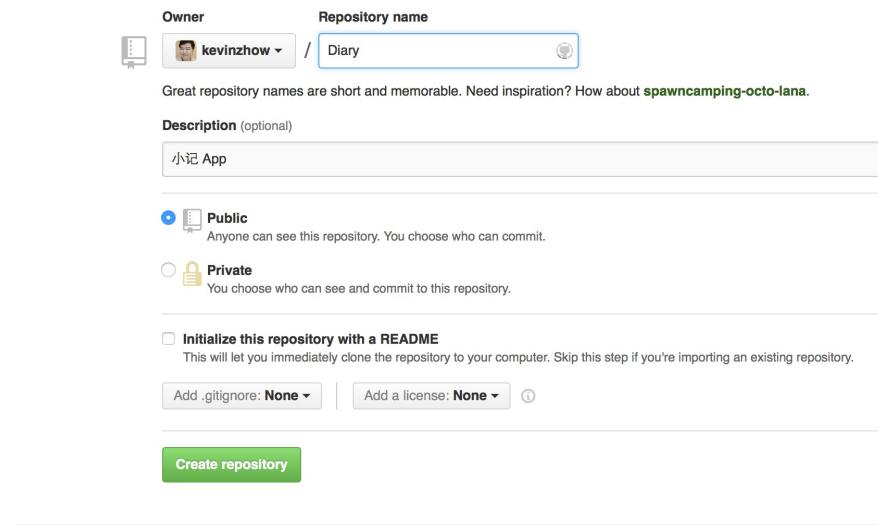
---

对于新手来说，使用 [Github 的 Mac 客户端](#) 安全又便捷，快下一个。

## 创建新项目

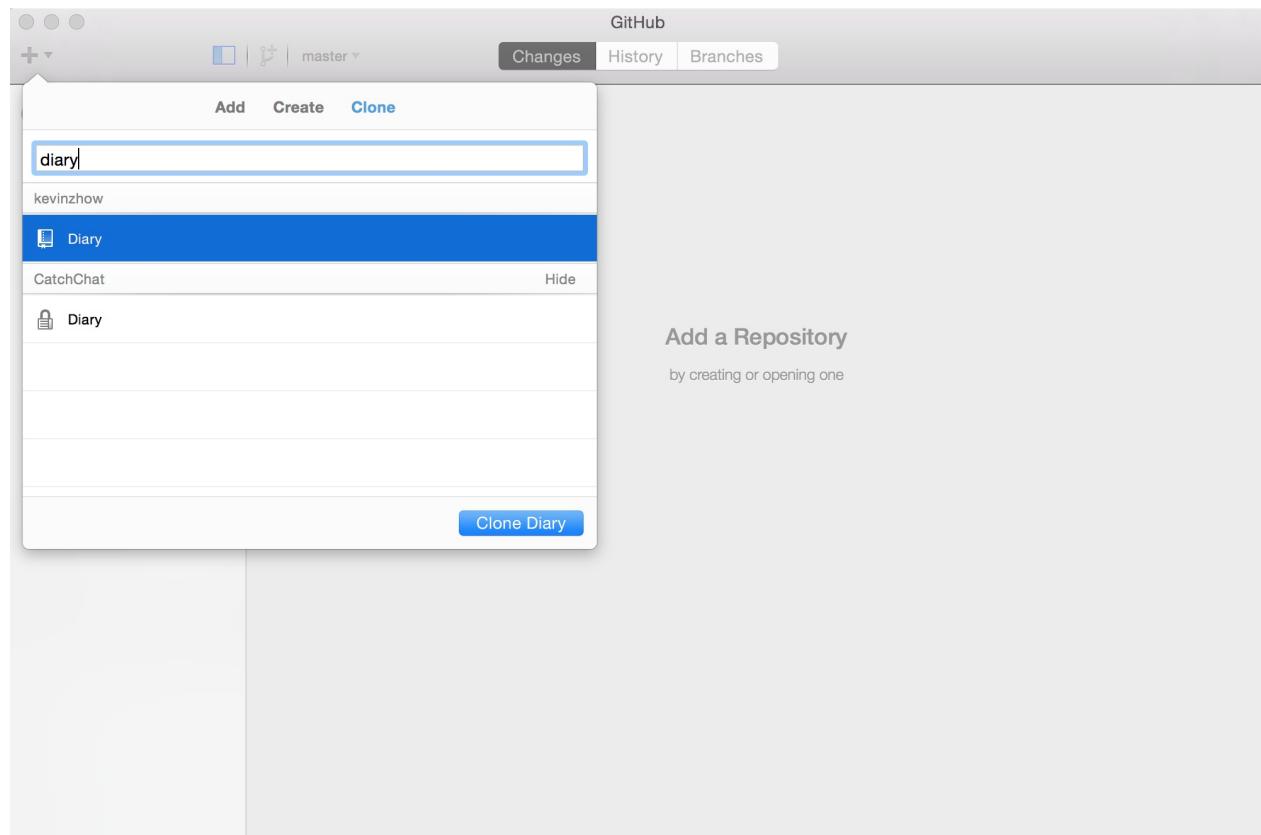
---

以 Github 为例，点击右上角的加号添加新项目



完成这步之后，就可以在刚安装的客户端上登录你的账号，一路 `continue` 之后，进入主界面。

点击左上角的加号，切换到 Clone 页面，在 filter 里输入项目的名字来定位到我们的仓库。

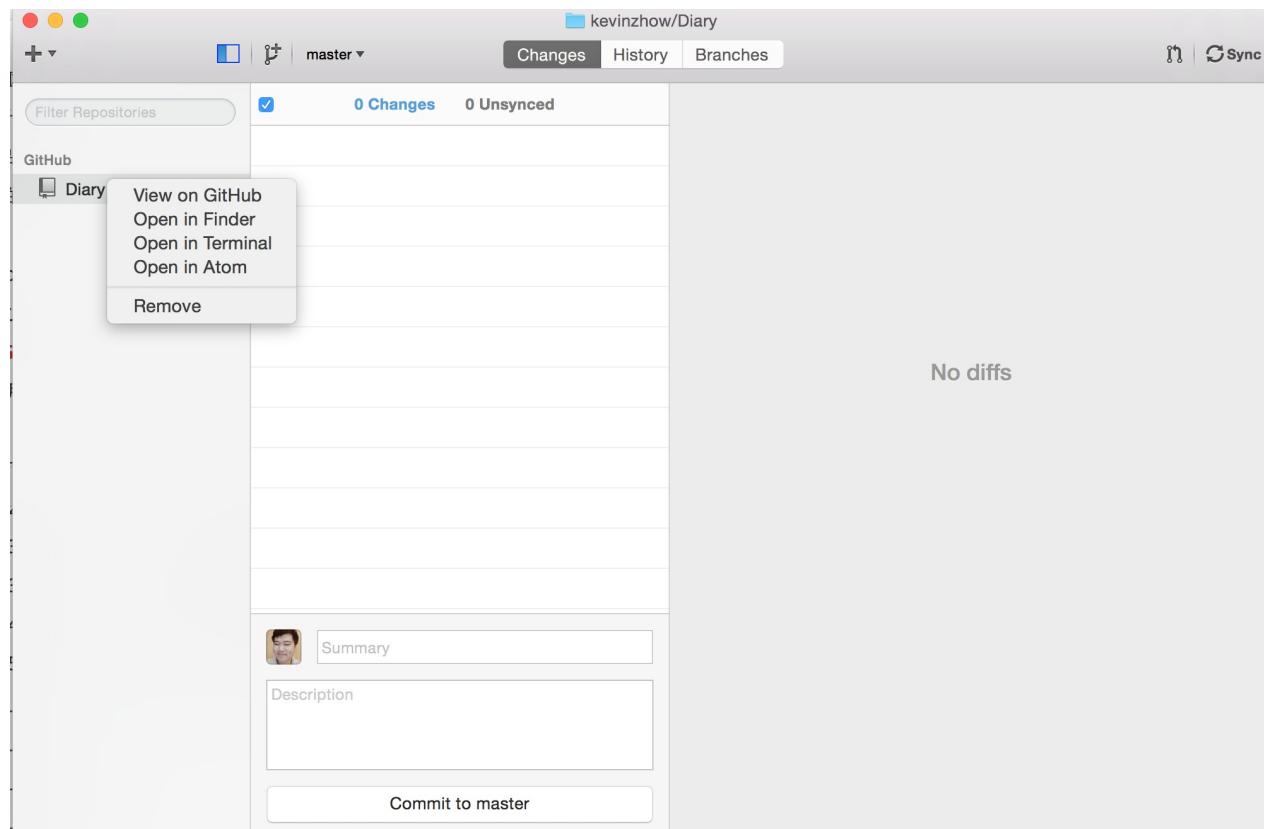


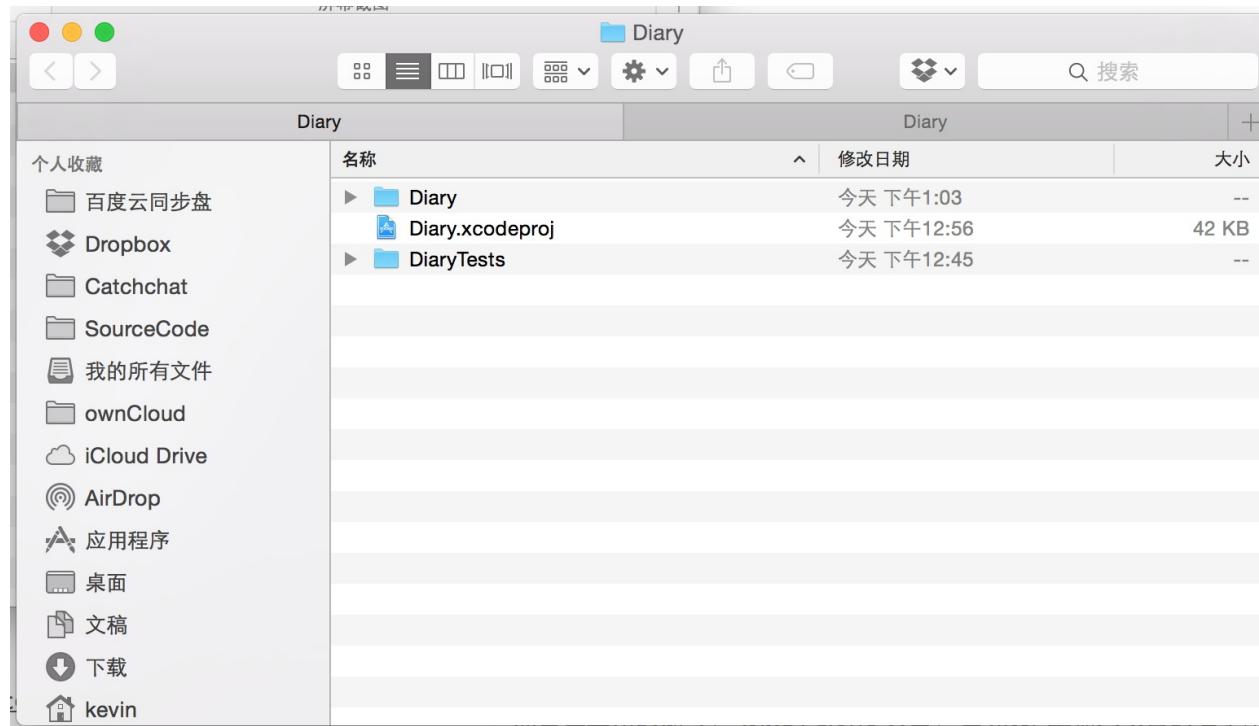
点击 `Clone Diary` 把仓库克隆到我们本地。

# Git 基础

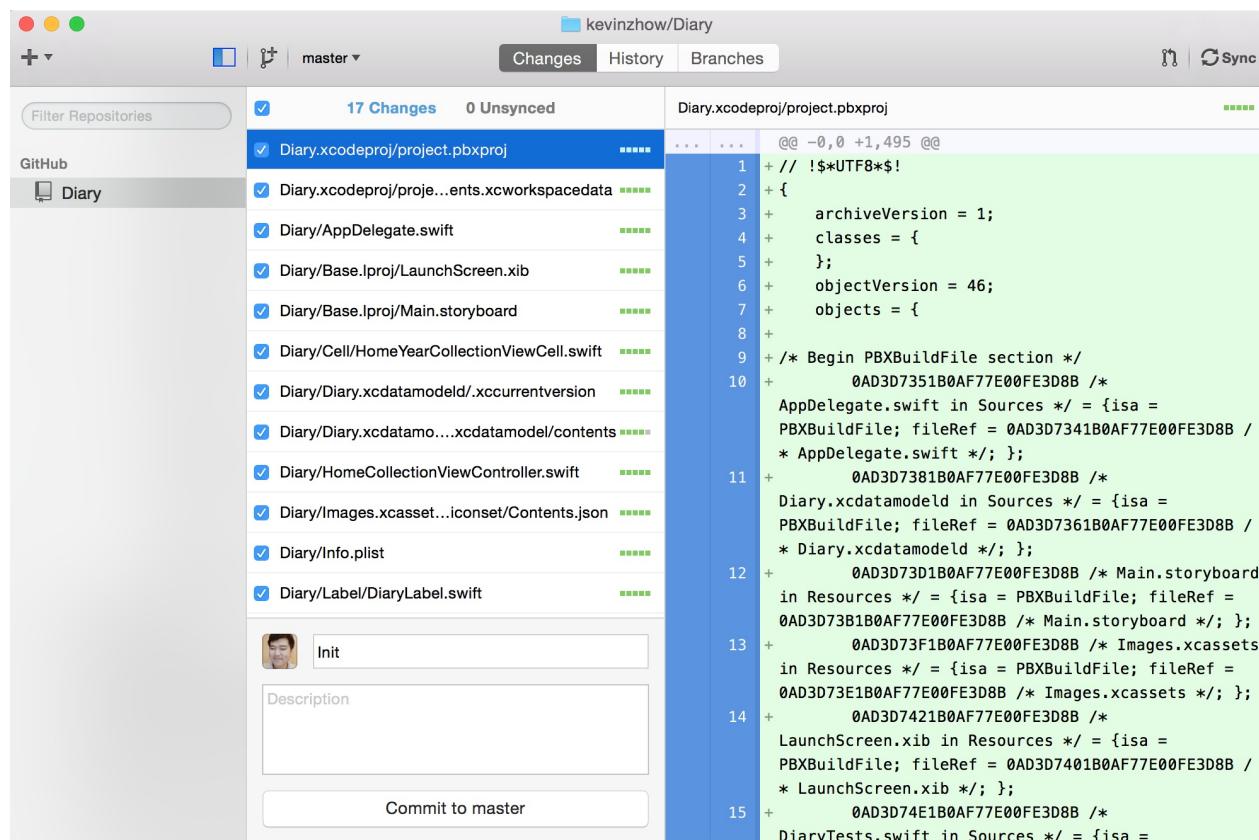
Git 有着诸多优秀的特性，不过对于新手来说，有一个对你特别有用，就是可以随时还原到某一个你提交的版本。

在仓库上右键，Open in Finder，发现里面是空的，没关系，把我们刚刚完成的项目复制进来





返回到 Github 的客户端，可以看到 App 已经开始询问我们是否把这些新文件提交到仓库里。



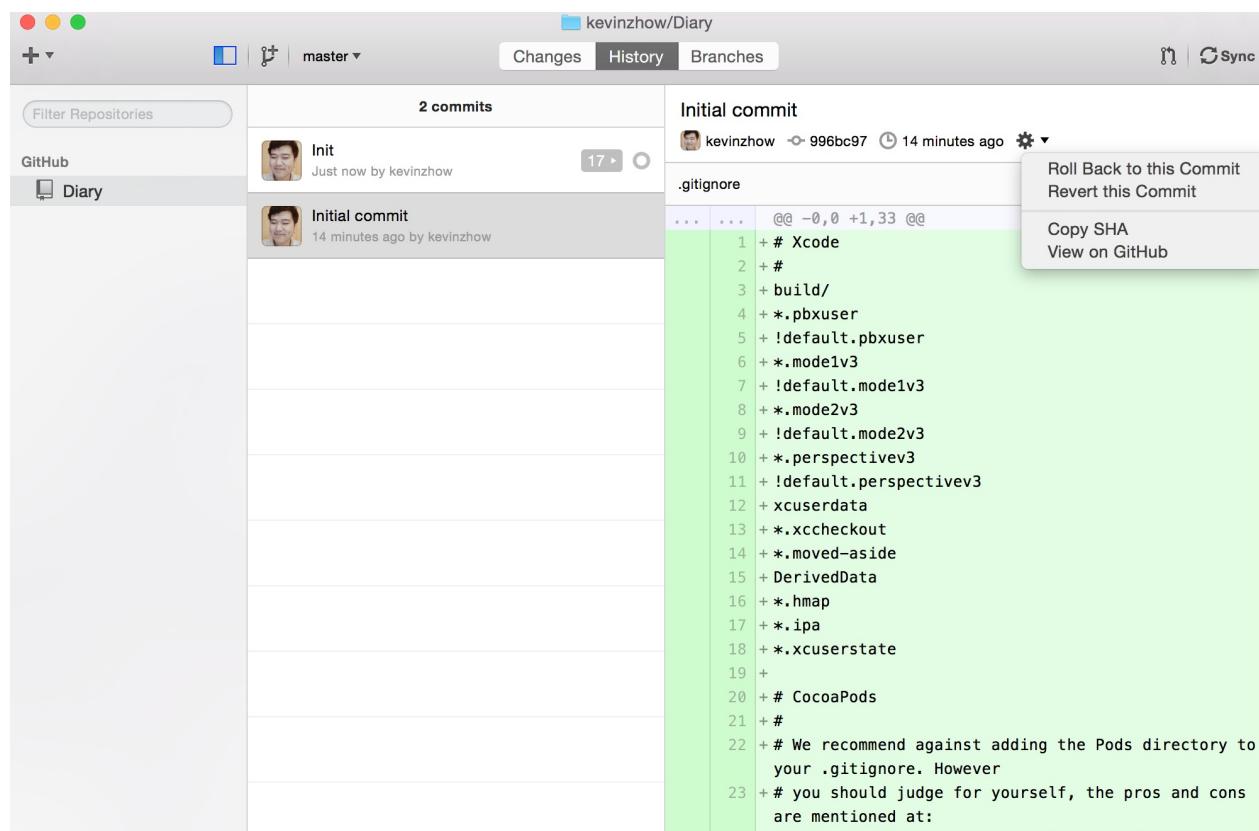
在 Summary 里输入本次提交的内容，然后潇洒的点击 commit to master，就可以提交上去了。

完成之后，你的代码就在本地和服务器上都有了版本控制的功能。

版本控制怎么使用呢？

点击 History

选择我们最早的 Initial commit 那时候我们还没有提交任何内容，点击右边的齿轮，选择 roll back to this commit



此时你再打开 Finder 就会发现所有文件都不见了，因为在那个时间点上，我们没有任何文件。

放心，你的文件不会丢失，在 Init 这条纪录上重复刚才的 roll back to this commit，你的文件就又都回来了。

## 命令行

如果你想挑战一下难度，也可以选择命令行。

首先你需要安装 Homebrew

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

然后运行

```
brew install git
```

## Init

接下来你需要配置你的 Git 信息

```
git config --global user.name "YOUR NAME"  
git config --global user.email "YOUR EMAIL ADDRESS"
```

进入项目目录

```
git init
```

这时你的项目就处在 Git 的管理之下了。

## Add

使用

```
git add FILE_NAME
```

来将需要追踪的文件加入到 Git 里。

如果你希望追踪所有文件，那么可以使用

```
git add .
```

这个命令来实现。

## Commit

使用

```
git commit -a
```

这个命令会先让你 Review 下变动，输入 Summary 内容就可以完成确认。如果你是个新手，可能此时命令行的编辑器该怎么用都已经把你搞晕了。

哎，我真是太坏了，为什么要把你领进命令行？嗯，其实知识想告诉你，还存在这么一个只有字符界面的世界，有兴趣的话，不妨了解一下。

此时你只是在本地，要提交到云端，还需要几步设置，不过既然 Github 客户端可以如此简单地完成这些事情，我们还是先去用客户端吧！

## 深入学习 Git

---

[GitCafe 官方帮助文档](#)

# Product Code: 产品的实现 - 年 和 月

点击年就是显示月份咯，而月和年并没有很大区别。除了字体从康熙字典换成了文悦古仿宋。

当然，此刻你点击是没有什么发生，因为还有一个问题需要回答。当点击的时候，该做什么。

```
override func collectionView(collectionView: UICollectionView,
    didSelectItemAtIndexPath indexPath: NSIndexPath) {  
}  
}
```

在上面的代码里写入如下片段

```
var identifier = "DiaryYearCollectionViewController"  
var dvc = self.storyboard?.  
instantiateViewControllerWithIdentifier(identifier)  
as! DiaryYearCollectionViewController  
  
// 获取 DiaryYearCollectionViewController  
  
dvc.year = 2015  
  
// 指定是 2015 年的月份  
  
self.navigationController!.pushViewController(dvc, animated: true)  
  
// 页面跳转
```

这时候点击年…… 依旧不能跳转，因为 `DiaryYearCollectionViewController` 这个东西还没有。

仿照之前在年之外的创建方法，我们创建一个 `DiaryYearCollectionViewController`，`DiaryYearCollectionViewController` 同时需要一个 `year` 的 `Int` 属性，存储是哪一年。

不过完成了这步还是依旧不能跳转，因为 `self.navigationController` 这个东西还不存在。

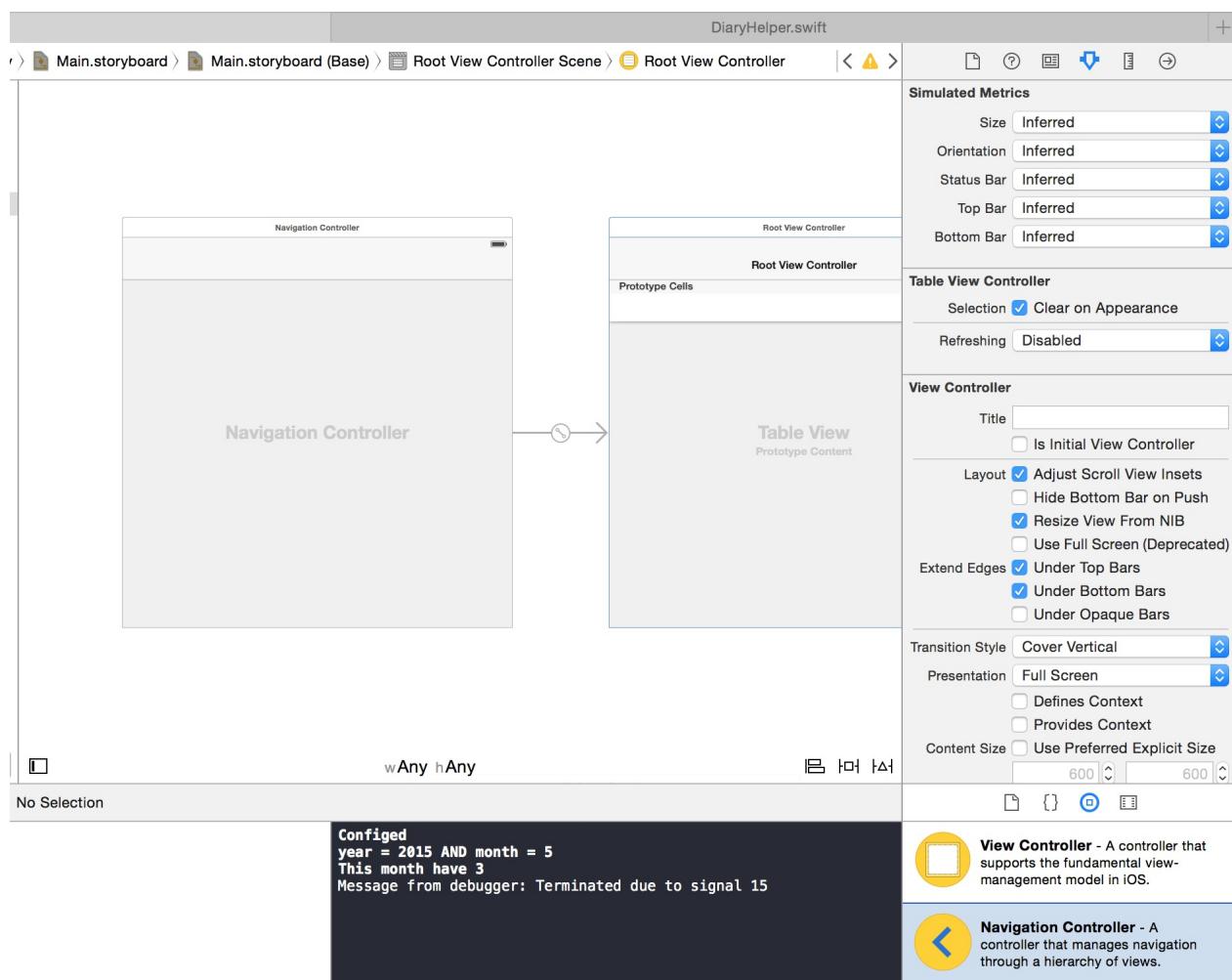
# UINavigationController

UINavigationController 是 iOS 里管理页面切换的家伙，pushViewController 可以切换到下一个页面，popViewController 可以返回上一页。

先在要给我们的 App 增加一个最外围的 UINavigationController。

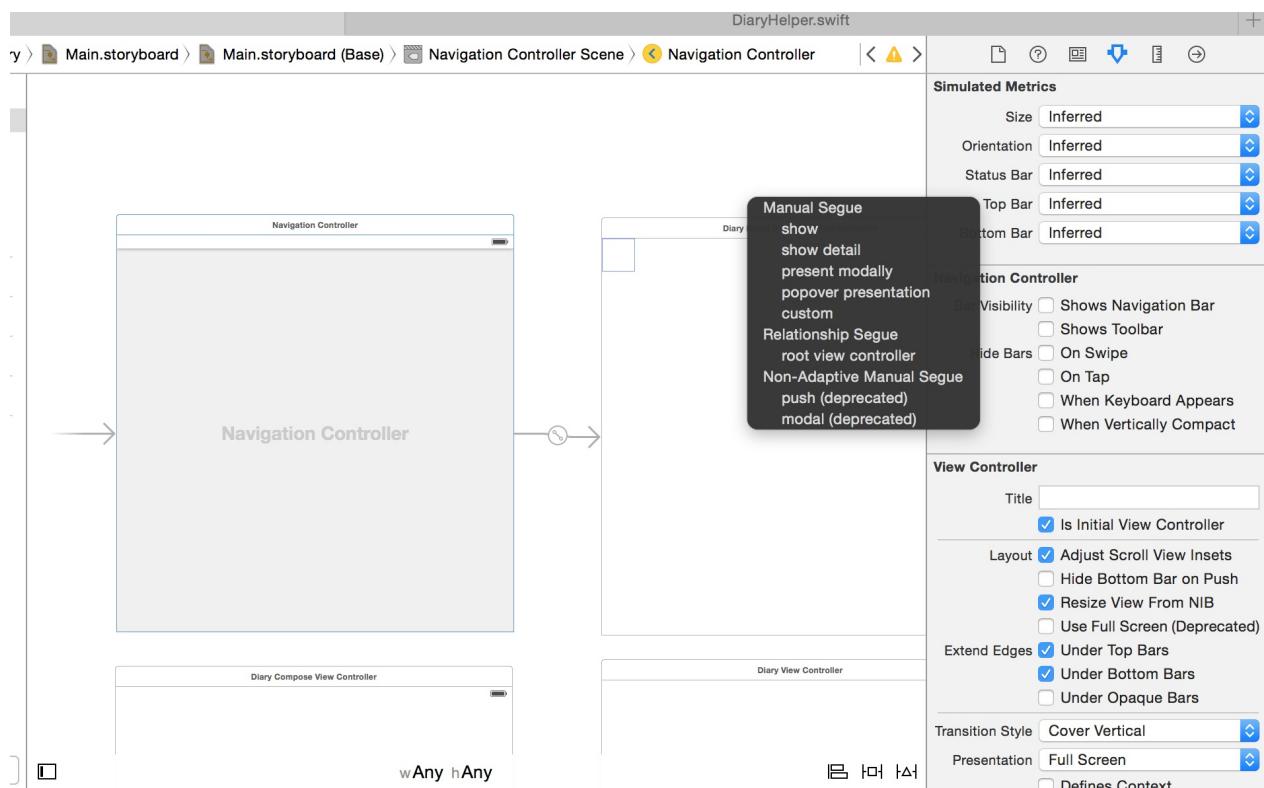
回到 Main.storyboard

增加一个 UINavigationController

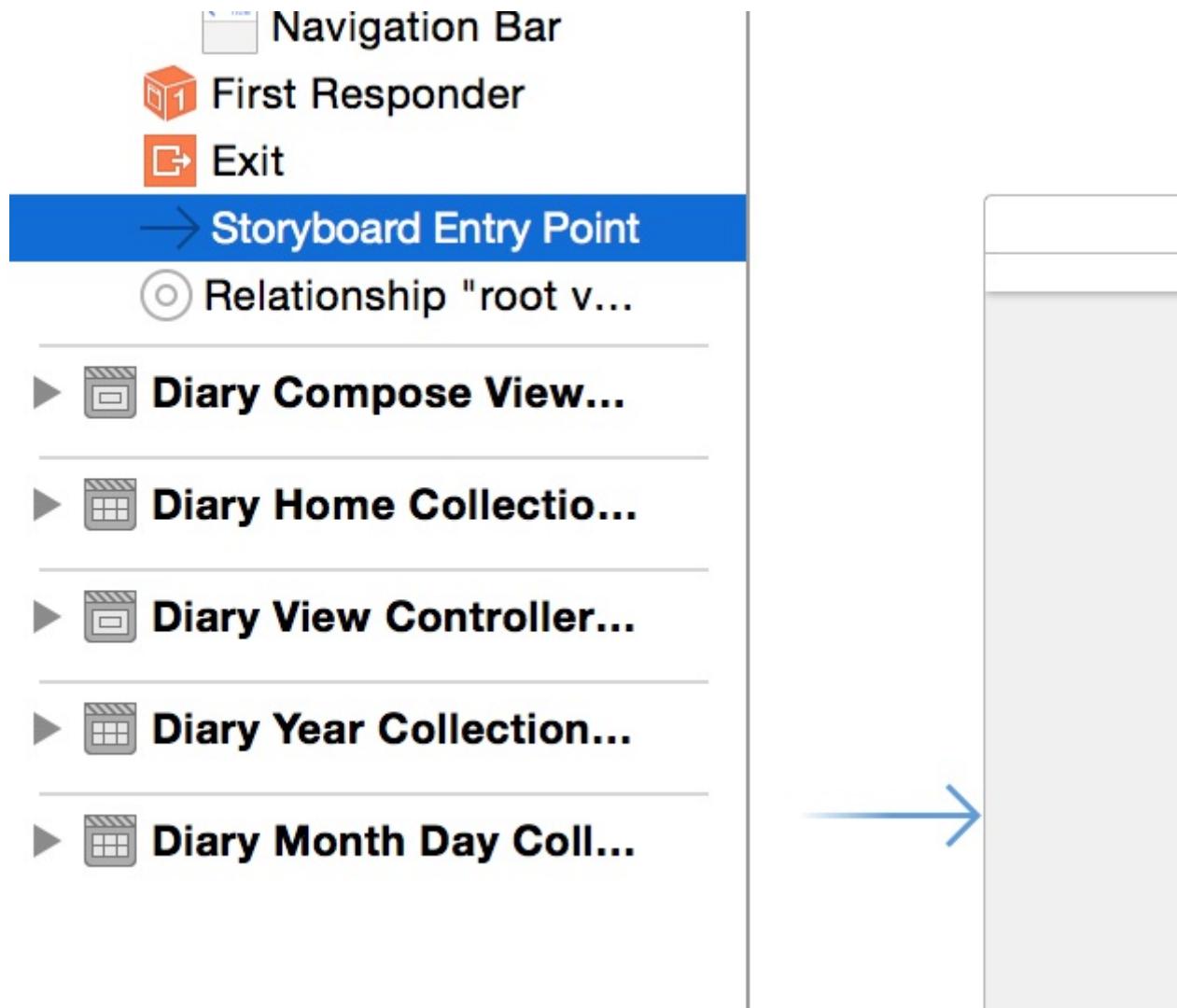


然后删掉它附赠的 UITableViewController

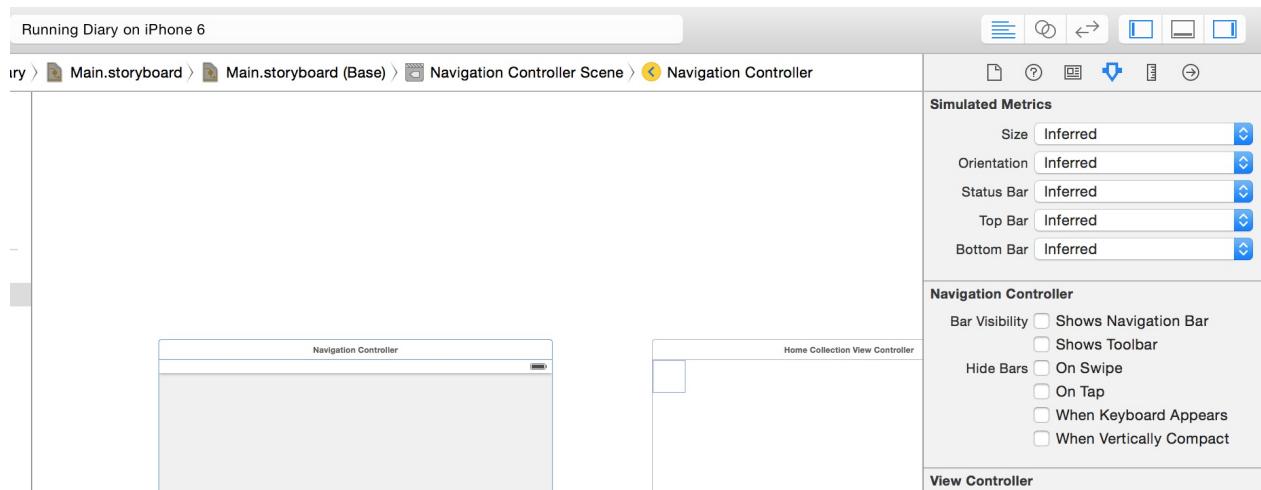
按住 Control 然后在 UINavigationController 上按住左键，拖拽到 HomeCollectionViewController 上选择 rootViewController



确认已经把 Storyboard Entry Point 移动到 UINavigationController 上



另外一个需要注意的就是要 Hide 掉 Navigation Bar，在右边的 Storyboard 上取消 Show Navigation Bar 上的对勾即可。



重新运行 App，页面就可以切换到年视图了。

点击月份进入这个月视图，月视图里面，显示的就是日记了。

我们需要在月和年的视图右上角加上年和月，月单独的有个红色的月份显示在右边。

年份和月份并无太大不同，只是颜色和字体略有区别，月这个按钮倒是值得一提。

## UIButton

因为这个 Button 会在多处出现，所以弄一个方法来简易生产这样的 Button

```
func diaryButtonWith(#text: String,
    #fontSize: CGFloat, #width: CGFloat,
    #normalImageName: String,
    #highlightedImageName: String) -> UIButton
{
    var button = UIButton.buttonWithType(UIButtonType.Custom)
        as! UIButton

    //创建自定义 Button

    button.frame = CGRectMake(0, 0, width, width)

    //设定 Button 的大小

    var font = UIFont(name: "WYUE-GutiFangSong-NC",
        size: fontSize) as UIFont!

    let textAttributes: [NSObject : AnyObject] =
        [NSFontAttributeName: font,
        NSForegroundColorAttributeName: UIColor.whiteColor()]

    var attributedText = NSAttributedString(string: text,
        attributes: textAttributes)

    button.setAttributedTitle(attributedText,
        forState: UIControlState.Normal)

    //设置 Button 字体

    button.setBackgroundImage(
        UIImage(named: normalImageName),
        forState: UIControlState.Normal)

    //设置默认 Button 样式
    button.setBackgroundImage(
        UIImage(named: highlightedImageName),
        forState: UIControlState.Highlighted)

    // 设置 Button 被按下时候的样式

    return button
}
```

```
}
```

在 `viewDidLoad` 里面，通过以下代码添加这个 Button

```
var composeButton = diaryButtonWith(text: "撰",
fontSize: 14.0,
width: 40.0,
normalImageName: "Oval",
highlightedImageName: "Oval_pressed")

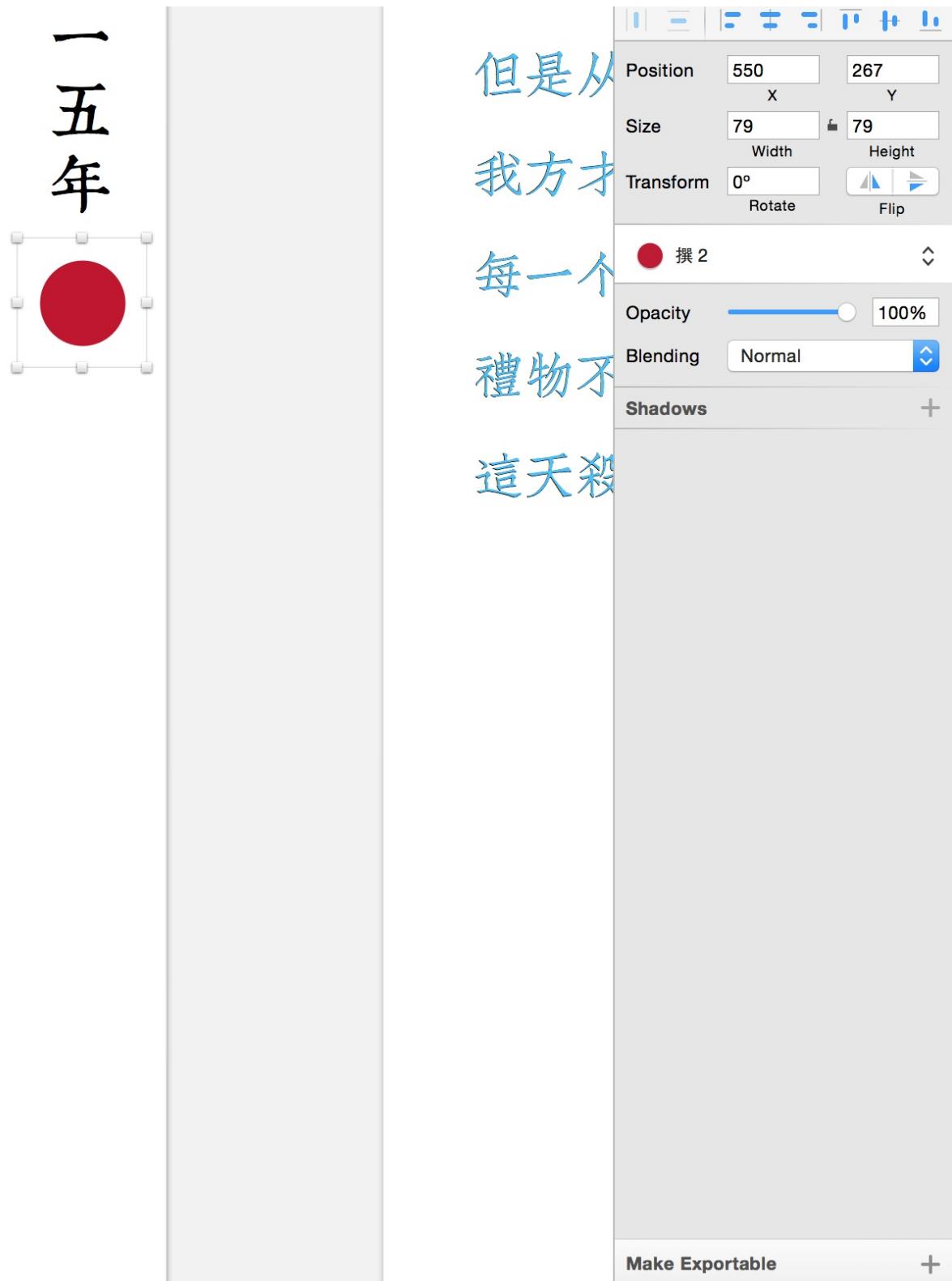
composeButton.center = CGPointMake(yearLabel.center.x,
38 + yearLabel.frame.size.height + 26.0/2.0)

composeButton.addTarget(self, action: "newCompose",
forControlEvents: UIControlEvents.TouchUpInside)

self.view.addSubview(composeButton)
```

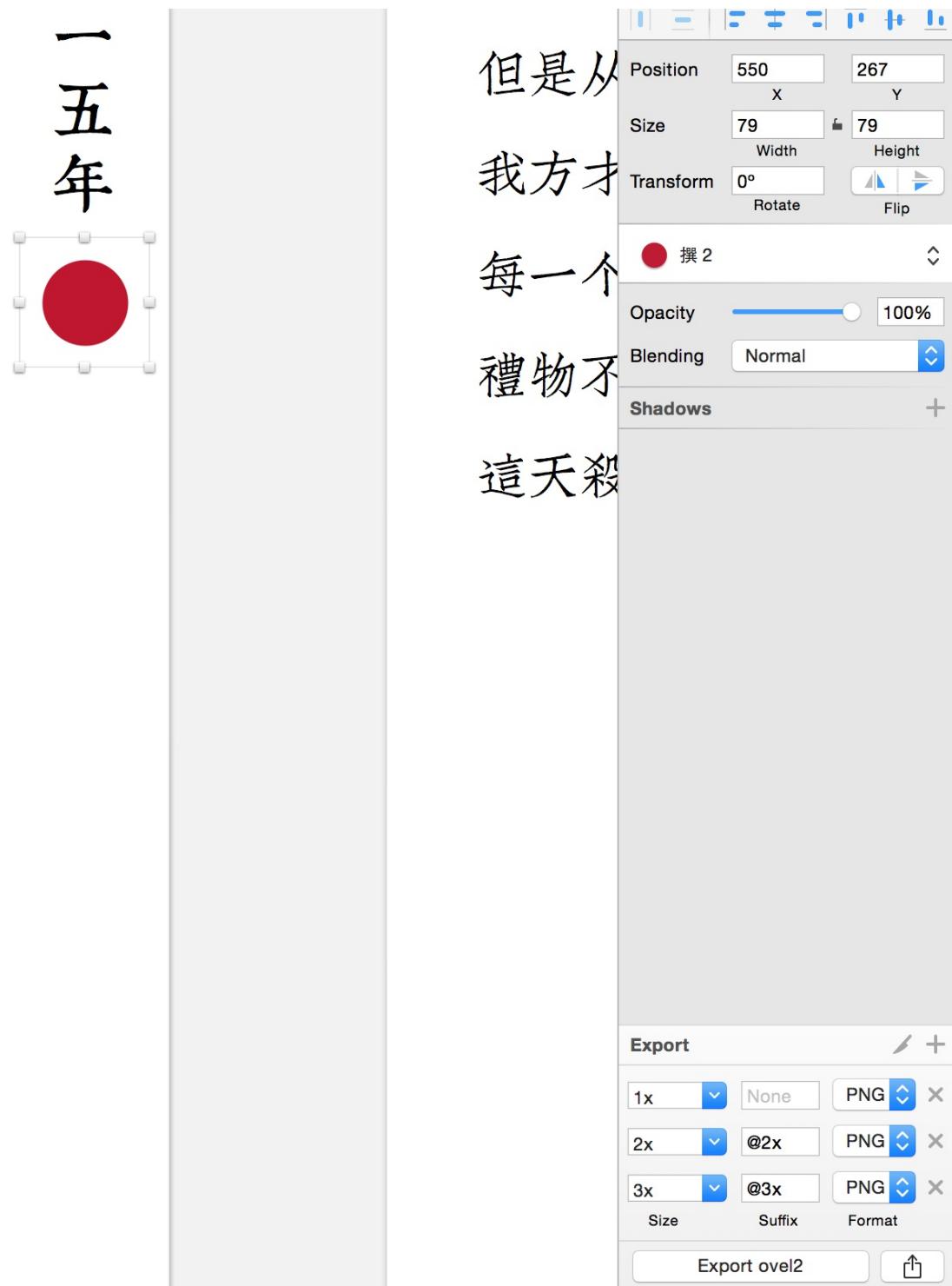
Oval 是单独绘制的一个红色圆形，pressed 则是颜色略有不同的圆形，在 Sketch 里，你可以这样简单的导出图片。

选中你要导出的图形

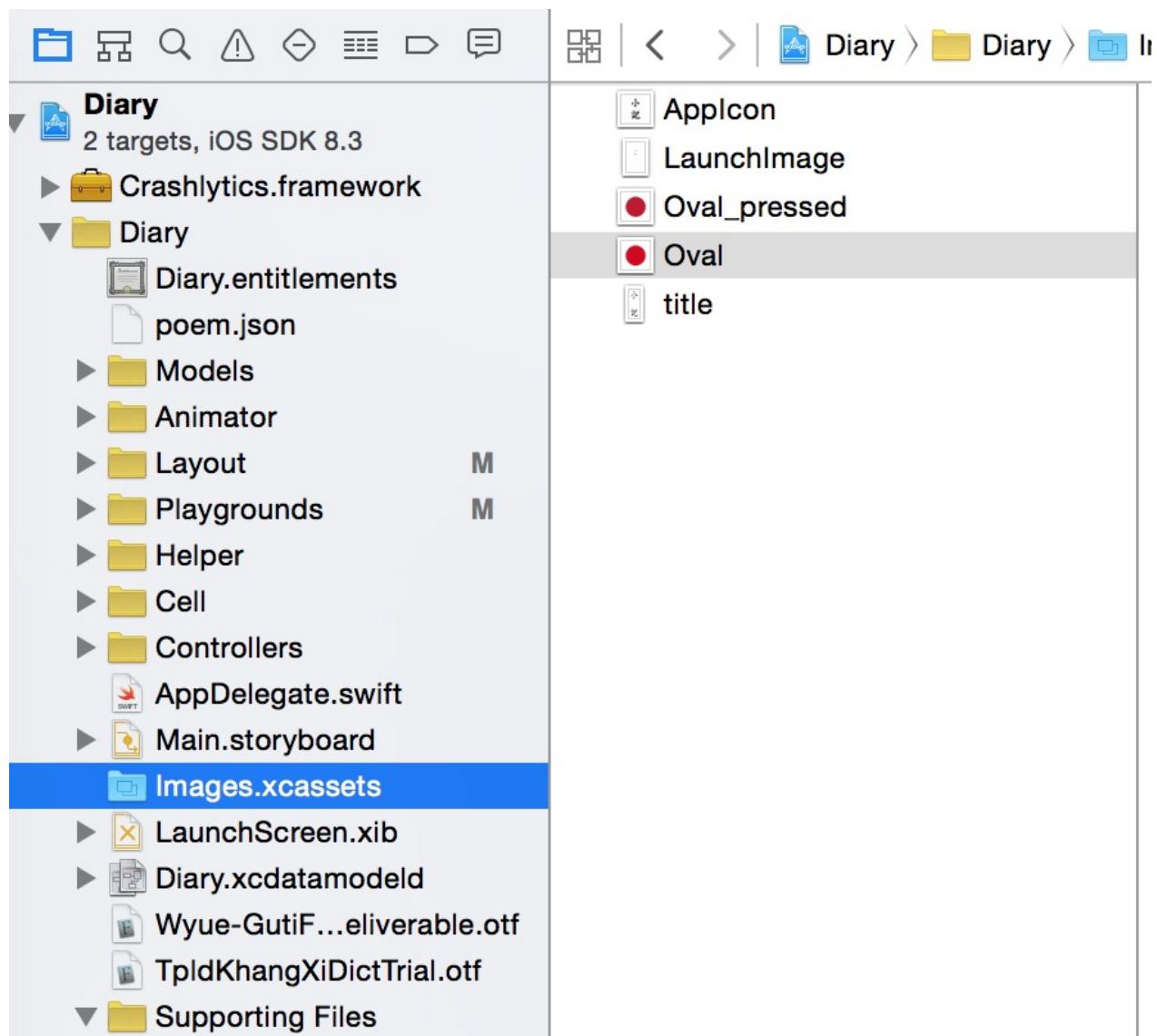


点击 Make Exportable

然后增加你需要的尺寸

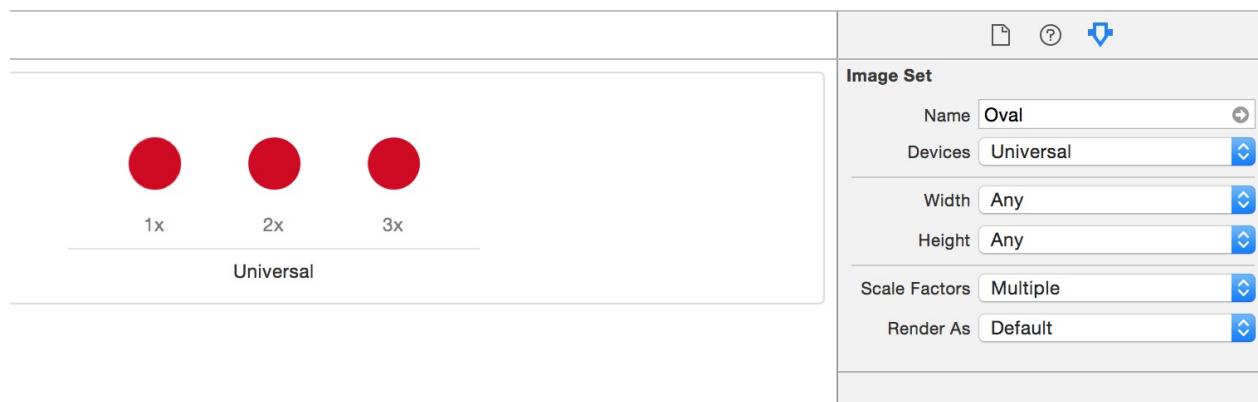


导出后，把他们拖到 XCode 的 Images.xcassets 里面即可



PNG 和 PDF 都是不错的类型，PDF 只需要一张，由 XCode 在编译的时候自动生成多个尺寸，而 PNG 需要你手动添加多个尺寸，对于不同的类型，PNG 的 Scale Factors 是 Multible，PDF 的是 Single Vector。





不过 PDF 也有一定的局限性，例如不能体现复杂的图层效果，对于透明的处理上也会有些问题。

## 自定义转场

按照我们的交互设计，这里的页面切换方式不应该是从右边推进进来，好在 iOS 里已有自定义转场这样的非常有趣的技术。

首先在 HomeCollectionViewController 的 `viewDidLoad` 里面指定

```
self.navigationController!.delegate = self
```

在 HomeCollectionViewController 的底部增加一个 Extension

```
extension HomeCollectionViewController: UINavigationControllerDelegate {
}
```

Extension 可以扩展类能够实现的功能。

这时候 navigationController 就会询问具体的转场方法。

```
func navigationController(navigationController: UINavigationController,
    animationControllerForOperation operation: UINavigationControllerOperation,
    fromViewController fromVC: UIViewController,
    toViewController toVC: UIViewController) -> UIViewControllerAnimatedTransitioning? {
}
```

我们先增加一个 DiaryAnimator

```
import UIKit

class DiaryAnimator: NSObject, UIViewControllerAnimatedTransitioning {

    var operation:UINavigationControllerOperation!

    //转场时长

    func transitionDuration(transitionContext:
        UIViewControllerContextTransitioning) -> NSTimeInterval {
        return 0.4
    }

    //转场的参数变化

    func animateTransition(transitionContext: UIViewControllerContextTransitioning) {
        let containerView = transitionContext.containerView()

        //获取转场舞台

        let fromVC = transitionContext.
            viewControllerForKey( UITransitionContextFromViewControllerKey)

        let fromView = fromVC!.view

        //获取从哪个场景开始转

        let toVC    = transitionContext.
            viewControllerForKey( UITransitionContextToViewControllerKey)

        let toView = toVC!.view

        //获取要转去哪个场景

        toView.alpha = 0.0

        //设置新场景透明度

        // UINavigationControllerOperation.Pop 用来判断是转入还是转出

        if operation == UINavigationControllerOperation.Pop {
            toView.transform = CGAffineTransformMakeScale(1.0,1.0)

            // 如果是返回旧场景，那么设置要转入的场景初始缩放为原始大小

        }else{
            toView.transform = CGAffineTransformMakeScale(0.3,0.3);
        }
    }
}
```

```

        // 如果是转到新场景，设置新场景初始缩放为 0.3
    }

    containerView.insertSubview(toView, aboveSubview: fromView)

    // 在舞台上插入场景

    // 进行动画

    UIView.animateWithDuration(transitionDuration(transitionContext),
    delay: 0,
    options: UIViewAnimationOptions.CurveEaseInOut,
    animations:
    {
        if self.operation == UINavigationControllerOperation.Pop {
            fromView.transform = CGAffineTransformMakeScale(3.3,3.3)

            // 放大要转出的场景

        }else{
            toView.transform = CGAffineTransformMakeScale(1.0,1.0);

            // 设置新场景为原始大小
        }

        toView.alpha = 1.0

    }, completion: { finished in
        transitionContext.completeTransition(true)
        // 通知 NavigationController 已经完成转场
    })
}
}

```

接下来把转场代码写进去，就可以生效啦

```

func navigationController(navigationController: UINavigationController,
animationControllerForOperation operation: UINavigationControllerOperation,
fromViewController fromVC: UIViewController,
toViewController toVC: UIViewController) -> UIViewControllerAnimatedTransitioning? {

    var animator = DiaryAnimator()
    animator.operation = operation
    return animator

}

```

按照这个流程，我们还需要再实现一个 `DiaryMonthDayCollectionViewController` 来显示月份下的日记。

你可以在[这里](#)找到我们本章的工程文件

# Product Code: 产品的实现 - 撰写

为了让点击撰的这个事件被捕捉，我们需要给 composeButton 增加如下代码

```
composeButton.addTarget(self,
    action: "newCompose",
    forControlEvents: UIControlEvents.TouchUpInside)
```

这个事件的处理分为了三步

1. Target
2. Action
3. Event

Action 就是 Method，即运行什么方法的意思，Target 是指哪个对象会响应这个事件，Event 是指由什么事件触发，这里 TouchUpInside 的意思就是当用户按在了按钮的正中，然后抬起手指。

UIControlEvents 有很多事件，你可以探索一下。

在外围，也要增加一个响应的函数，函数的名称正是上面 Action 的文字。

```
func newCompose() {
    var composeViewController = self.storyboard?.
        instantiateViewControllerWithIdentifier("DiaryComposeViewController")
    as! DiaryComposeViewController

    self.presentViewController(composeViewController,
        animated: true,
        completion: nil)
}
```

又出现了我们缺少的 DiaryComposeViewController，这个 ViewController 可能是这个 App 里最复杂的部分。

## 日记的数据结构

日记大概需要这四个构成

1. 名称
2. 内容
3. 位置信息
4. 创建时间

一十五日

一吃了好吃的，看了電影  
睡覺|



这里我都使用了 `UITextView` 来创建前三个信息输入框，原因是复制方便。

```
composeView = UITextView(frame: CGRectMake(
    0,
    contentMargin + titleTextViewHeight, screenRect.width,
    screenRect.height),
    textContainer: container)

composeView.font = DiaryFont
composeView.editable = true
composeView.userInteractionEnabled = true
composeView.delegate = self
composeView.textContainerInset = UIEdgeInsetsMake(contentMargin,
    contentMargin,
    contentMargin,
    contentMargin)

//添加 LocationTextView

locationTextView = UITextView(frame: CGRectMake(0,
    composeView.frame.size.height - 30.0, screenRect.width - 60.0, 30.0))
```

```

locationTextView.font = DiaryLocationFont
locationTextView.editable = true
locationTextView.userInteractionEnabled = true
locationTextView.alpha = 0.0
locationTextView.bounces = false

```

这里值得一提的是 `textContainerInset`, 几乎 `UIView` 的子类都支持类似的方法, 可以将内容挤压产生边距。

`bounces` 是指是否可以产生回弹效果, 当你在 iPhone 滑动邮件列表的时候, 到了顶端并不会立刻停止, 而是会产生一个像弹簧一样的拉动效果, 这个就是 `bounces`。

## NSNotification

布局完成后, 当用户点击了 `UITextView` 要输入文字的时候, `locationTextView` 可能会被挡在键盘下面, 这时候我们就需要监听键盘事件来移动 `locationTextView` 的位置。

```

NSNotificationCenter.defaultCenter().addObserver(self,
    selector: "keyboardDidShow:",
    name: UIKeyboardDidShowNotification,
    object: nil)

```

监听一个通知主要需要四个

1. Observer
2. NotificationName
3. Selector
4. Object

`Observer` 是指由谁负责监听, `NotificationName` 是指监听什么事件, `Selector` 是监听到的信息由哪个方法来处理。

如果是自己发消息, `Object` 可以填入你想传送的数据, 而这里因为是监听一个系统消息, 所以并不需要我们填入数据。

`keyboardDidShow` 后面加了个冒号, 意思是指这个方法接收参数, 即下面的 `notification: NSNotification`

```

func keyboardDidShow(notification: NSNotification) {
    if let rectValue = notification.userInfo?[UIKeyboardFrameEndUserInfoKey] as? NSValue {
        keyboardSize = rectValue.CGRectValue().size
        updateTextViewSizeForKeyboardHeight(keyboardSize.height)
    }
}

```

NSNotification 是我非常喜欢的一个特性，在 App 内传递消息变得非常方便。不管是在哪个地方，只要做一下响应的监听，就可以响应其他地方发出的消息。

获取传递过来的信息有两种方法，一种是 notification.userInfo 一种是 notification.object。

userInfo 一般是系统消息，object 则是你放入 Object 里的对象，你可以用 as 方法转换成你需要的类型。

## Animation

iOS 的动画有许多实现方式，不过我想你最开始肯定想要一个最简单的方式。

updateTextViewSizeForKeyboardHeight 这个方法需要移动 LocationTextView 的位置，这里就可以通过 UIViewAnimation 来实现。

```

var newKeyboardHeight = keyboardHeight

UIView.animateWithDuration(1.0, delay: 0,
options: UIViewAnimationOptions.CurveEaseInOut,
animations:
{
    self.finishButton.center = CGPointMake(
        self.view.frame.width - self.finishButton.frame.size.height/2.0 - 10,
        self.view.frame.height - newKeyboardHeight
        - self.finishButton.frame.size.height/2.0 - 10)

    self.locationTextView.center = CGPointMake(
        self.locationTextView.frame.size.width/2.0 + 20.0,
        self.finishButton.center.y)

}, completion: nil)

```

不只是 center，其实很多属性都可以在这里移动，color、frame 大小、alpha 等。

# Location

获取地理位置我们可以通过 iOS 自带的地理位置服务，不过首先你需要在 Info.plist 里增加个 NSLocationWhenInUseUsageDescription 字段说明为什么你需要位置权限，这段文字也会在系统向用户要求权限的时候展示给用户。

Bundle version	String	10
Application requires iPhone environment	Boolean	YES
NSLocationWhenInUseUsageDescription	String	在撰写日记的时候自动添加位置
▼ Fonts provided by application	Array	(2 items)

我们又一次用到了NSNotification 发送我们自定义的 DiaryLocationUpdated 来通知编辑界面记录位置信息。

```
import CoreLocation

class DiaryLocationHelper: NSObject, CLLocationManagerDelegate {

    var locationManager:CLLocationManager = CLLocationManager()
    var currentLocation:CLLocation?
    var address:String?
    var geocoder = CLGeocoder()

    override init() {
        super.init()
        locationManager.delegate = self
        locationManager.distanceFilter = kCLLocationAccuracyNone
        //位置更新的位移触发条件
        locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
        //精度
        locationManager.pausesLocationUpdatesAutomatically = true
        //如果用户停止移动则停止更新位置
        locationManager.headingFilter = kCLHeadingFilterNone
        //指南针触发条件
        locationManager.requestWhenInUseAuthorization()
        //请求位置权限
        println("Location Right")
        if (CLLocationManager.locationServicesEnabled()){
            //判断是否有位置权限
            locationManager.startUpdatingLocation()
            // 开始更新位置
        }
    }

    func locationManager(manager: CLLocationManager!, didUpdateToLocation newLocation: CLLocation!, fromLocation oldLocation: CLLocation!) {
        // 位置更新后通过 CLGeocoder 获取位置描述, 例如 广州市
    }
}
```

```

geocoder.reverseGeocodeLocation(newLocation,
completionHandler: { (placemarks, error) in

    if (error != nil) {
        println("reverse geocode fail: \(error.localizedDescription)")
    }

    if let pm = placemarks as? [CLPlacemark] {
        if pm.count > 0 {

            var placemark = pm.first

            self.address = placemark?.locality
            NSNotificationCenter.defaultCenter().
            postNotificationName("DiaryLocationUpdated", object: self.address)
            //发送位置更新的通知
        }
    }
}

})
}
}

```

## CoreData

Core Data 是 iOS 默认的数据库技术，也是 iOS 里相对复杂的技术，不过我们这里只用一个基本功能。

在创建 App 的时候，我们默认启用了 Core Data，这时候你的项目里会有一个 Diary.xcdatamodeld 文件。

点击这个文件后，右边就可以编辑数据库的数据结构，点击 Add Entity 添加一个 Diary 记录类型。

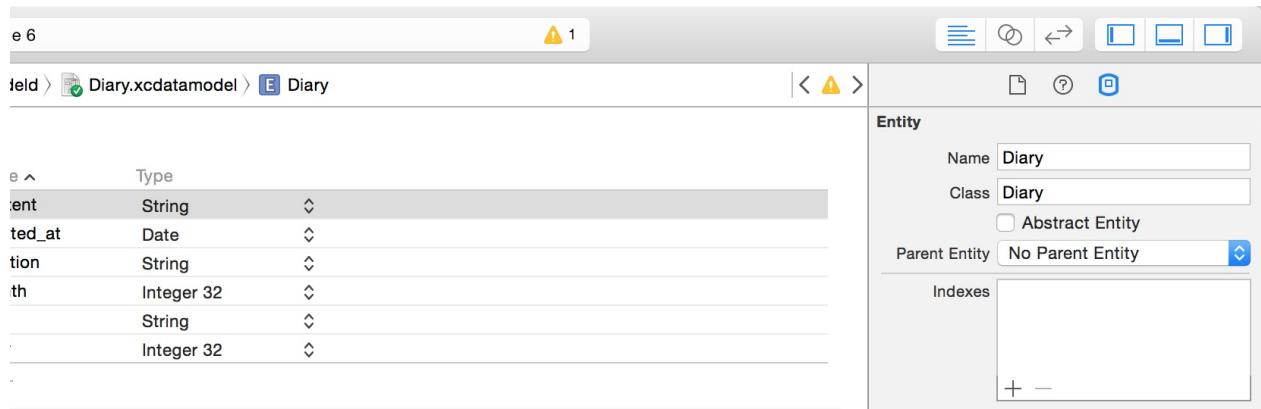
The screenshot shows the Xcode Core Data editor interface. On the left, there's a sidebar with sections for 'ENTITIES', 'FETCH REQUESTS', and 'CONFIGURATIONS'. Under 'ENTITIES', 'Diary' is selected. On the right, there's a table titled 'Attributes' with the following columns: 'Attribute ^' and 'Type'. The table lists six attributes:

Attribute ^	Type
S content	String
D created_at	Date
S location	String
N month	Integer 32
S title	String
N year	Integer 32

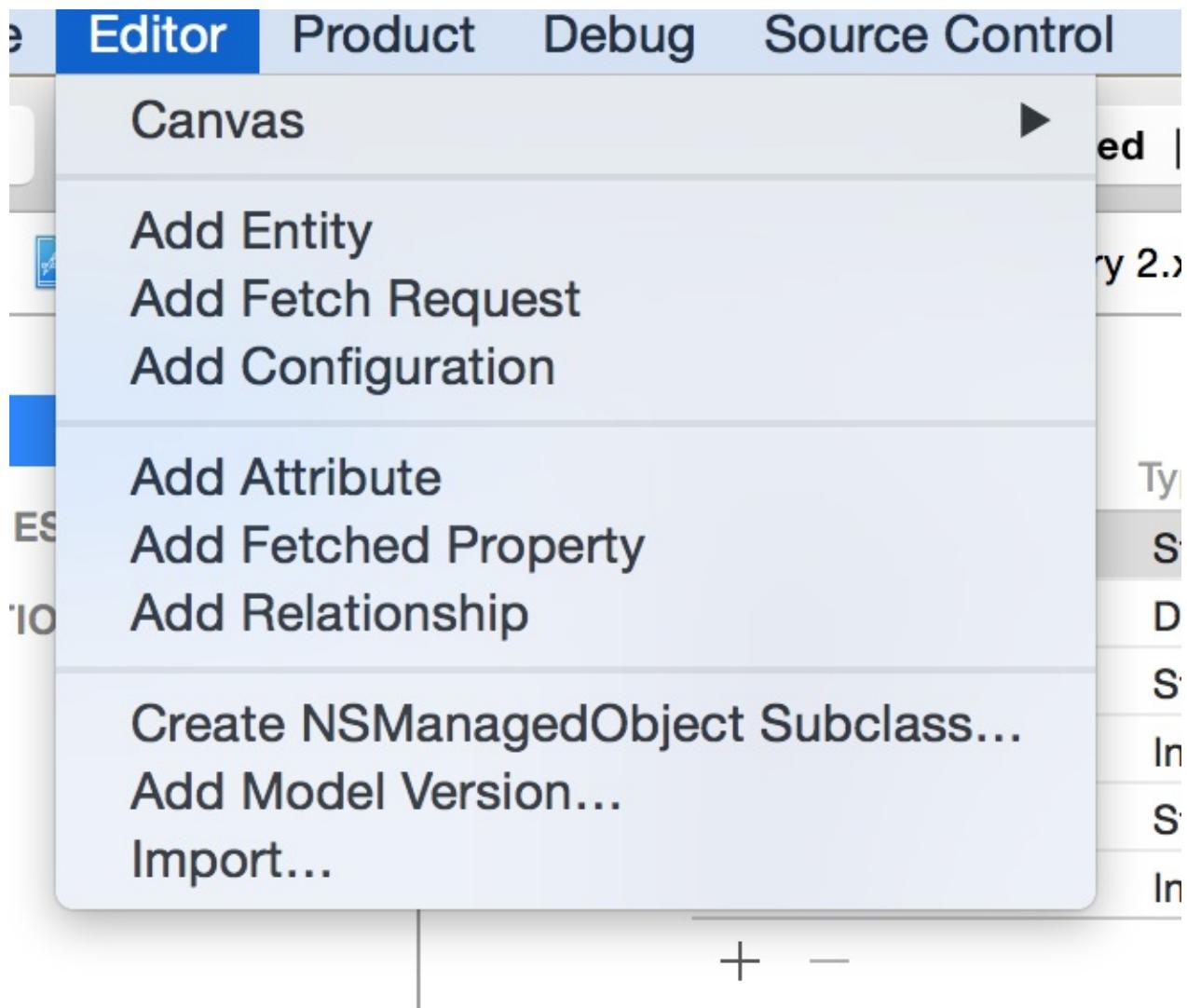
Below the table are two small buttons: a plus sign (+) and a minus sign (-).

Attribute 是这个类型的一些属性结构，content 以 String 类型来存放日记正文，created\_at 以 Date 类型存放创建日期，这里我们还添加了 month、title、year 这三个属性方便后面搜索。

在右边将 Class 名称改成 Diary



编辑完毕后，就可以导出成 NSManagedObject，为了能在 Swift 中正常使用，你还需要增加一个 @objc(Diary) 标识。



```

import Foundation
import CoreData

@objc(Diary)
class Diary: NSManagedObject {

    @NSManaged var title: String?
    @NSManaged var content: String
    @NSManaged var created_at: NSDate
    @NSManaged var location: String
    @NSManaged var year: NSNumber
    @NSManaged var month: NSNumber

}

```

这个文件和 Diary 的 Attribute 是一一对应的关系。

在 iOS 的 AppDelegate 的顶端，可以创建两个静态变量

```
//Coredata
let appDelegate =
UIApplication.sharedApplication().delegate as! AppDelegate

let managedObjectContext = appDelegate.managedObjectContext!
```

managedContext 是数据库的管理对象，通过这个对象就可以操作数据库的增删查改。

当你想要保存的日记的时候，就可以用下面的代码轻松解决

```
let entity = NSEntityDescription.entityForName("Diary",
    inManagedObjectContext: managedObjectContext)

let newdiary = Diary(entity: entity!,
    insertIntoManagedObjectContext:managedObjectContext)

newdiary.content = composeView.text
newdiary.location = locationTextView.text
newdiary.title = titleTextView.text
newdiary.updateTimeWithDate(NSDate())

var error: NSError?
if !managedObjectContext.save(&error) {
    println("Could not save \(error), \(error?.userInfo)")
}
```

你可以[在这里](#)找到本章的工程文件。

## 深入了解 CoreData 和 Animations

[Introduction to Core Data Programming Guide Animations](#)

# Product Code: 产品的实现 - 浏览年，月

浏览主要涉及到三个方面，查询，分类，展示。

## 查询

存了这么多日记，早就想读取出来看看了，进入 `DiaryHomeCollectionViewController`，添加以下属性

```
var diarys = [NSManagedObject]()

var fetchedResultsController : NSFetchedResultsController!

var yearsCount: Int = 1

var sectionsCount: Int = 0
```

在 `viewDidLoad` 中添加如下代码

```
let fetchRequest = NSFetchedResultsController(entityName:"Diary")

var error: NSError?

var year = 2015 // 可以先硬编码成当前年份
var month = 5 // 当前月份

let fetchedResults =
managedObjectContext.
executeFetchRequest(fetchRequest, error: &error) as! [NSManagedObject]?

fetchRequest.sortDescriptors = [NSSortDescriptor(key: "created_at",
ascending: true)]

// 排序方式

fetchRequest.predicate = NSPredicate(format: "year = \\\(year) AND month = \\\(month)")

// 查询条件

fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
managedObjectContext: managedObjectContext, sectionNameKeyPath: "year",
cacheName: nil) // 根据 Year 来分 Section
```

```

if (!fetchedResultsController.performFetch(&error)){
    println("Error: \(error?.localizedDescription)")
}else{
    var fetchedResults = fetchedResultsController.
    fetchedObjects as! [NSManagedObject]

    if (fetchedResults.count == 0){
        println("Present empty year")
    }else{

        if let sectionsCount = fetchedResultsController.sections?.count {
            yearsCount = sectionsCount
            diarys = fetchedResults
        }else {
            sectionsCount = 0
            yearsCount = 1
        }
    }
}

```

NSPredicate 是一个相当强大的查询语句，同样可以用在 NSArray 上，不过不需要上来就完全了解这个东西，在开始，会简单的条件查询就够了。

## 获取分组信息

分组已经由 NSFetchedResultsController 帮我们自动完成，那么接下来就是在年、月界面正确的显示。

年的数量

```

override func collectionView(collectionView: UICollectionView,
    numberOfItemsInSection section: Int) -> Int {
    return yearsCount
}

```

年的 Cell 配置

```

override func collectionView(collectionView: UICollectionView,
    cellForItemAtIndexPath indexPath: NSIndexPath) -> HomeYearCollectionViewCell {

    let cell = collectionView.
    dequeueReusableCellWithReuseIdentifier(reuseIdentifier, forIndexPath: indexPath)
    as! HomeYearCollectionViewCell
}

```

```

var components = NSCalendar.currentCalendar().
component(NSCalendarUnit.CalendarUnitYear, fromDate: NSDate())

var year = components
if sectionsCount > 0 {
    if let sectionInfo = fetchedResultsController.
sections![indexPath.row] as? NSFetchedResultsSectionInfo {

        println("Section info \(sectionInfo.name)")
        year = sectionInfo.name!.toInt()!

    }
}

cell.textInt = year
cell.labelText = "\(numberToChinese(cell.textInt)) 年"

// Configure the cell

return cell
}

```

点击年的时候

```

override func collectionView(collectionView: UICollectionView,
didSelectItemAtIndexPath indexPath: NSIndexPath) {

    var dvc = self.storyboard?.
instantiateViewControllerWithIdentifier("DiaryYearCollectionViewController")
as! DiaryYearCollectionViewController

    var components = NSCalendar.currentCalendar().
component(NSCalendarUnit.CalendarUnitYear, fromDate: NSDate())

    var year = components

    if sectionsCount > 0 {

        if let sectionInfo = fetchedResultsController.sections![indexPath.row] as? NSFetchedResultsSectionInfo {

            println("Section info \(sectionInfo.name)")
            year = sectionInfo.name!.toInt()!

        }
    }

    dvc.year = year
}

```

```
    self.navigationController!.pushViewController(dvc, animated: true)  
}
```

self.storyboard?.

instantiateViewControllerWithIdentifier("DiaryYearCollectionViewController") 这个方法可以从 StoryBoard 里取出 Identifier 为 DiaryYearCollectionViewController 的实例对象。

月份采用了类似的方式实现。

月的查询略有变化

```
fetchRequest.predicate = NSPredicate(format: "year = \year AND month = \month")
```

增加了确切的年和月份

当点击月份的某篇日记的时候，将对应的 Diary 传递过去

```
var dvc = self.storyboard?.  
instantiateViewControllerWithIdentifier("DiaryViewController")  
as! DiaryViewController  
  
var diary = fetchedResultsController.objectAtIndexPath(indexPath) as! Diary  
  
dvc.diary = diary  
  
self.navigationController!.pushViewController(dvc, animated: true)
```

你可以[在这里](#)找到本章的工程项目

## 深入了解 **NSPredicate** , **NSFetchedResultsController**

[NSPredicate](#)

[NSPredicate Class Reference](#)

[NSFetchedResultsController Class Reference](#)

[Core Data Tutorial for iOS: How To Use NSFetchedResultsController](#)

# Product Code: 产品的实现 - 浏览日记

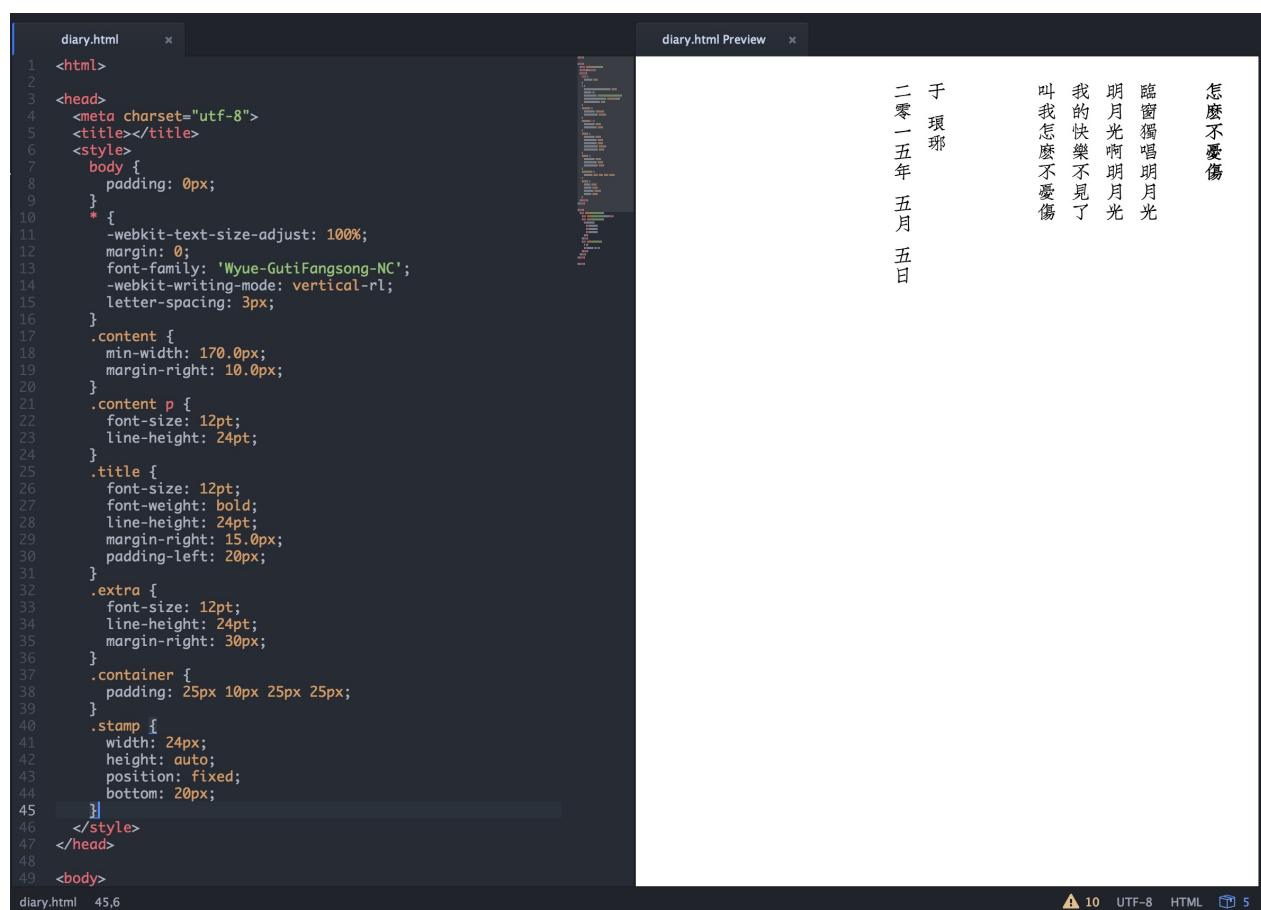
在文字竖排方面，我原本是打算用 CoreText 来实现，源码里有一个 DiaryVerticalTextView，就是竖排效果，但是考虑到 UIWebView 可以简单灵活的实现，我们还是采用 UIWebView 来搞定这个问题。

在使用 WebView 之前，我们需要先实现这个网页的样式。

写网页，推荐编辑器 Atom。

## HTML & CSS

First of all, 先看下效果



The screenshot shows the Atom code editor on the left and a UIWebView preview on the right. The code editor displays the following HTML and CSS:

```

1 <html>
2   <head>
3     <meta charset="utf-8">
4     <title></title>
5     <style>
6       body {
7         padding: 0px;
8       }
9       * {
10         -webkit-text-size-adjust: 100%;
11         margin: 0;
12         font-family: 'WYue-GutiFangSong-NC';
13         -webkit-writing-mode: vertical-rl;
14         letter-spacing: 3px;
15       }
16       .content {
17         min-width: 170.0px;
18         margin-right: 10.0px;
19       }
20       .content p {
21         font-size: 12pt;
22         line-height: 24pt;
23       }
24       .title {
25         font-size: 12pt;
26         font-weight: bold;
27         line-height: 24pt;
28         margin-right: 15.0px;
29         padding-left: 20px;
30       }
31       .extra {
32         font-size: 12pt;
33         line-height: 24pt;
34         margin-right: 30px;
35       }
36       .container {
37         padding: 25px 10px 25px 25px;
38       }
39       .stamp {
40         width: 24px;
41         height: auto;
42         position: fixed;
43         bottom: 20px;
44       }
45     </style>
46   </head>
47   <body>
48     <p>臨窗獨唱明月光</p>
49     <p>明月光响明月光</p>
50     <p>我的快樂不見了</p>
51     <p>叫我怎麼不憂傷</p>
52     <p>于環琊</p>
53     <p>二零一五年五月五日</p>
54   </body>

```

The preview window shows a vertical text layout with the following content:

臨窗獨唱明月光  
明月光响明月光  
我的快樂不見了  
叫我怎麼不憂傷  
于環琊  
二零一五年五月五日

代码也是相当的简单，我最早学会的和写代码有关的就是做网页。

```
<html>
```

```

<head>
  <meta charset="utf-8">
  <title></title>
  <style>
    body {
      padding: 0px;
    }
    * {
      -webkit-text-size-adjust: 100%;
      margin: 0;
      font-family: 'WYUE-GUTI FANGSONG-NC';
      -webkit-writing-mode: vertical-rl;
      letter-spacing: 3px;
    }
    .content {
      min-width: 170.0px;
      margin-right: 10.0px;
    }
    .content p {
      font-size: 12pt;
      line-height: 24pt;
    }
    .title {
      font-size: 12pt;
      font-weight: bold;
      line-height: 24pt;
      margin-right: 15.0px;
      padding-left: 20px;
    }
    .extra {
      font-size: 12pt;
      line-height: 24pt;
      margin-right: 30px;
    }
    .container {
      padding: 25px 10px 25px 25px;
    }
    .stamp {
      width: 24px;
      height: auto;
      position: fixed;
      bottom: 20px;
    }
  </style>
</head>

<body>
  <div class="container">
    <div class="title">怎麼不憂傷</div>
    <div class="content">
      <p>臨窗獨唱明月光
        <br>明月光啊明月光
        <br>我的快樂不見了
        <br>叫我怎麼不憂傷
      </p>
    </div>
  </div>
</body>

```

```

</div>
<div class="extra">
    于 琅琊
    <br>二零一五年 五月 五日
</div>
</div>
</body>

</html>

```

HTML 和 CSS 在语言上并没没有什么难度，HTML 就是用一个个成对的标签把内容标记起来，通过 `class` 来指定类。CSS 用 `.class {}` 的形式指定这个类的样式。

难处在于排版经验的积累，如果你有兴趣学习网页，那么可以从 [W3cSchool](#) 开始。

## UIWebView

---

新建一个 `DiaryViewController`，添加一个 `webview` 的属性

```

webview = UIWebView(frame: CGRectMake(0, 0, self.view.frame.size.width, self.view.frame.size.height))
webview.scrollView.bounces = true
webview.backgroundColor = UIColor.whiteColor()
webview.scrollView.delegate = self

```

创建一个 `Diary` 来填充内容

```

let diary = Diary()
diary.content = "沒道理\n是一枚太平洋的暖濕空氣\n飄散了我們的心\n在青春的墓地\n就這樣平行下去"
diary.location = "于 琅琊"
diary.title = "要有光"
diary.updateTimeWithDate(NSDate())

```

然后把我们制作好的网页载入进去

```

let headertags = "<!DOCTYPE html><html><meta charset='utf-8'><head><title></title></head><body>"
let bodyCSS = "body{padding:\(bodyPadding)px;} "
let allCSS = "* {-webkit-text-size-adjust: 100%;"

```

```

margin:0; font-family: '\(fontStr)';
-webkit-writing-mode: vertical-rl;
letter-spacing: 3px;}"

let contentCSS = ".content { min-width: \$(minWidth)px;
margin-right: \$(contentMargin)px;}
.content p{ font-size: 12pt;
line-height: 24pt;}"

let titleCSS = ".title {font-size: 12pt;
font-weight:bold;
line-height: 24pt;
margin-right: \$(titleMarginRight)px;
padding-left: 20px; }"

let extraCSS = ".extra{ font-size:12pt;
line-height: 24pt;
margin-right:30px; }"

let stampCSS = ".stamp {width:24px;
height:auto;
position:fixed;
bottom:20px; }"

let extraHTML = "<div class='extra'>\$(diary.location)
<br>\$(timeString) </div>"

let contentHTML = "<div class='container'>\$(title)
<div class='content'>
<p>\$(newDiaryString)</p></div>"

webView.loadHTMLString("\$(headertags)\$(bodyCSS)
\$(allCSS) \$(contentCSS) \$(titleCSS) \$(extraCSS)
.container { \$(containerCSS) } \$(stampCSS) </style>
</head> <body>\$(contentHTML) \$(extraHTML)</body>
</html>", baseURL: nil)

```

你可以[在这里](#)找到本章的工程文件

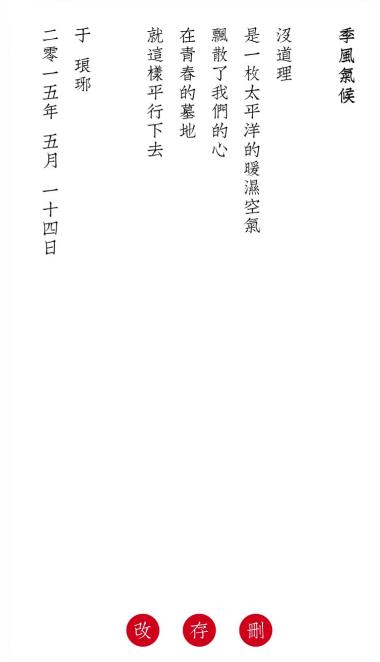
## 深入了解 UIWebView

[UIWebView Class Reference](#)

[WebViewJavascriptBridge](#)

# Product Code: 产品的实现 - 编辑日记

在实现了基本的创建浏览之后，对日记的编辑就是最后一件事情的了。三个红色的点和直书结合后的展示效果我非常喜欢。



## 添加按钮

首先我们有三个红色的按钮需要添加，这三个按钮统一放在 buttonsView 里面方便控制。

添加属性

```
var saveButton:UIButton!
var deleteButton:UIButton!
var editButton:UIButton!
var buttonsView:UIView!
// 在 `viewDidLoad` 里添加具体控件
buttonsView = UIView(frame: CGRectMake(0,
screenRect.height,
```

```
screenRect.width,
80.0))

buttonsView.backgroundColor = UIColor.clearColor()
buttonsView.alpha = 0.0

saveButton = diaryButtonWith(text: "存",
    fontSize: buttonFontSize,
    width: 50.0,
    normalImageName: "Oval",
    highlightedImageName: "Oval_pressed")

saveButton.center = CGPointMake(buttonsView.frame.width/2.0,
    buttonsView.frame.height/2.0)

saveButton.addTarget(self, action: "saveToRoll",
forControlEvents: UIControlEvents.TouchUpInside)

buttonsView.addSubview(saveButton)

editButton = diaryButtonWith(text: "改",
    fontSize: buttonFontSize,
    width: 50.0,
    normalImageName: "Oval",
    highlightedImageName: "Oval_pressed")

editButton.center = CGPointMake(saveButton.center.x - 56.0,
    saveButton.center.y)

editButton.addTarget(self, action: "editDiary",
forControlEvents: UIControlEvents.TouchUpInside)

buttonsView.addSubview(editButton)

deleteButton = diaryButtonWith(text: "刪",
    fontSize: buttonFontSize,
    width: 50.0,
    normalImageName: "Oval",
    highlightedImageName: "Oval_pressed")

deleteButton.center = CGPointMake(saveButton.center.x + 56.0,
    saveButton.center.y)

deleteButton.addTarget(self, action: "deleteThisDiary",
forControlEvents: UIControlEvents.TouchUpInside)

buttonsView.addSubview(deleteButton)

self.view.addSubview(buttonsView)
```

改

修改日记只需要把日记传给我们之前的创建界面

```
func editDiary() {
    var composeViewController = self.storyboard?.
        instantiateViewController(withIdentifier("DiaryComposeViewController"))
    as! DiaryComposeViewController

    if let diary = diary {
        composeViewController.diary = diary
    }

    self.presentViewController(composeViewController,
        animated: true, completion: nil)
}
```

对应的，当创建界面显示的时候，如果已经有 diary，那么显示即可

```
if let diary = diary {
    composeView.text = diary.content
    self.composeView.contentOffset = CGPointMake(0,
        self.composeView.contentSize.height)

    locationTextView.text = diary.location
    locationTextView.alpha = 1.0
    if let title = diary.title {
        titleTextView.text = title
    }
}
```

Finished 的时候，针对不同的情况进行处理

```
func finishCompose(button: UIButton) {
    self.composeView.endEditing(true)
    self.locationTextView.endEditing(true)

    if (composeView.text.lengthOfBytesUsingEncoding(NSUTF8StringEncoding) > 1){

        if let diary = diary {

            diary.content = composeView.text
            diary.location = locationTextView.text
            diary.title = titleTextView.text

        }else{
    }}
```

```

let entity = NSEntityDescription.entityForName("Diary",
inManagedObjectContext: managedContext)

let newdiary = Diary(entity: entity!,
insertIntoManagedObjectContext:managedContext)
newdiary.content = composeView.text

if let address = locationHelper.address {
    newdiary.location = address
}

if let title = titleTextView.text {
    newdiary.title = title
}

newdiary.updateTimeWithDate(NSDate())
}

var error: NSError?
if !managedContext.save(&error) {
    println("Could not save \(error), \(error?.userInfo)")
}
}

self.dismissViewControllerAnimated(true, completion: nil)
}

```

## 存

存所需要的其实就把 WebView 全尺寸展开然后转换成 UIImage

```

extension UIWebView {

func captureView() -> UIImage{

// 存储初始大小
var tmpFrame = self.frame

// 新的 Frame

var aFrame = self.frame

aFrame.size.width = self.sizeThatFits
(UIScreen.mainScreen().bounds.size).width

// 展开 Frame
self.frame = aFrame

// 初始化 ImageContext
}

```

```

 UIGraphicsBeginImageContextWithOptions(
    self.sizeThatFitsUIScreen mainScreen().bounds.size),
    false,
    UIScreen mainScreen().scale)

// 创建新的 Context

var resizedContext = UIGraphicsGetCurrentContext()
self.layer.renderInContext(resizedContext)

// 重新渲染到新的 resizedContext
var image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

// 还原 Frame
self.frame = tmpFrame
return image
}

}

```

最终利用系统的分享框架让用户选择适当的处理方式

```

func saveToRoll() {

    let offset = self.webview.scrollView.contentOffset.x
    var image = webview.captureView()
    self.webview.scrollView.contentOffset.x = offset

    var sharingItems = [AnyObject]()
    sharingItems.append(image)
    println("Do Share")

    let activityViewController = UIActivityViewController(
        activityItems: sharingItems, applicationActivities: nil)

    activityViewController.
    popoverPresentationController?.sourceView = self.saveButton

    self.presentViewController(activityViewController,
        animated: true, completion: nil)

}

```

删

```

func deleteThisDiary() {
    managedObjectContext.deleteObject(diary)
}

```

```
managedContext.save(nil)
hideDiary()
}

func hideDiary() {
    self.navigationController?.popViewControllerAnimated(true)
}
```

删除要简单得多，使用 `deleteObject()` 方法删除目前正在查看的 `Diary`，接着调用 `save()` 方法保存操作。最后实现 `hideDiary()` 返回上一层。

你可以[在这里](#)找到本章的工程文件

# Product ReDesign 5.1 产品的迭代 - 优化

在开发阶段，开发者往往会为了迅速实现功能而忽略结构上的合理性。

到了在这个阶段你就可以对整个程序的无用代码进行清理，对架构进行优化，例如之前我们有个问题就是 UICollectionView 不能全屏滑动。

删除之前更改 Frame 的做法，我们可以靠 UICollectionViewFlowLayout 来实现这个功能

## 自定义 UICollectionViewFlowLayout

```
import UIKit

var edgeInsets = (screenRect.width - collectionViewWidth)/2.0

class DiaryLayout: UICollectionViewFlowLayout {
    override func prepareLayout() {
        super.prepareLayout()
        let itemSize = CGSizeMake(itemWidth, itemHeight)
        self.itemSize = itemSize
        self.minimumInteritemSpacing = 0.0
        self.minimumLineSpacing = itemSpacing
        self.scrollDirection = .Horizontal
    }

    // 每次 Cell 的位置发生变化的时候都会执行
    // layoutAttributesForElementsInRect 询问 Cell 应该放在什么位置

    override func layoutAttributesForElementsInRect(rect: CGRect) -> [AnyObject]? {

        let layoutAttributes = super.layoutAttributesForElementsInRect(rect)
        as! [UICollectionViewLayoutAttributes]
        // 获取所有需要显示的 Cell 的位置信息

        let contentOffset = collectionView!.contentOffset
        // 获取 collectionView 的滑动情况

        for (index, attributes) in enumerate(layoutAttributes) {

            let center = attributes.center

            let cellPositinOnScreen = (center.x - itemWidth/2.0) - contentOffset.x

            if cellPositinOnScreen >= (edgeInsets - itemWidth/2.0)
                && cellPositinOnScreen < (edgeInsets + collectionViewWidth ) {

                // 计算 Cell 是不是在应该显示的区域
            }
        }
    }
}
```

```

        attributes.alpha = 1

    } else {
        attributes.alpha = 0
    }
}

return layoutAttributes
}

override func shouldInvalidateLayoutForBoundsChange(newBounds: CGRect) -> Bool {
    return true
}
}

```

通过这种方式就可以实现全屏滚动了。

## UITapGestureRecognizer & Touch Events

对于每个按钮和 Label 的点击，我希望可以有细腻的按下效果——按下时候缩小，放开时候还原，带有一点弹簧效果。

普通的实现方式是先添加一个 UITapGestureRecognizer

```

var tapGesture = UITapGestureRecognizer(target: self, action: "SomeMethod:")
self.view.addGestureRecognizer(tapGesture)

```

这时候你可以通过判断 UITapGestureRecognizer 的不同 state 来添加不同的动画

```

func SomeMethod(gesture: UITapGestureRecognizer) {
    if gesture.state == UIGestureRecognizerState.Began {
    } else if gesture.state == UIGestureRecognizerState.End {
    }
}

```

另外一种方式是通过较底层的 Touch Events

```

class DiaryLabel: UILabel {

    override func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent) {
        var anim = POPSpringAnimation(propertyNamed: kPOPLayerScaleXY)

```

```
anim.springBounciness = 10
anim.springSpeed = 15
anim.fromValue = NSValue(CGPoint: CGPointMake(1.0, 1.0))
anim.toValue = NSValue(CGPoint: CGPointMake(0.9, 0.9))
self.layer.pop_addAnimation(anim, forKey: "PopScale")
super.touchesBegan(touches as Set<NSObject>, withEvent: event)
}

override func touchesEnded(touches: Set<NSObject>, withEvent event: UIEvent) {
    var anim = POPSpringAnimation(propertyNamed: kPOPLayerScaleXY)
    anim.springBounciness = 10
    anim.springSpeed = 15
    anim.fromValue = NSValue(CGPoint: CGPointMake(0.9, 0.9))
    anim.toValue = NSValue(CGPoint: CGPointMake(1.0, 1.0))
    self.layer.pop_addAnimation(anim, forKey: "PopScaleback")
    super.touchesEnded(touches as Set<NSObject>, withEvent: event)
}

}
```

通过 `override DiaryLabel` 类的 `touchesBegan` 和 `touchesEnded` 方法来实现这个效果。这样每个 `DiaryLabel` 都可以实现一个精致的交互效果。

你可以在[这里](#)找到本章的最终工程文件，想要正确的打开这个工程文件，你还需要先阅读完 ReDesign 关于 CocoaPods 的介绍。

# Product ReDesign: 产品的迭代 - 动画库 Facebook POP

上一章我们使用了 Facebook 的 POP 动画库，下面一起来一探究竟。趁此机会，你也可以了解一下 Objective – C。

Facebook 在发布了 Paper 之后，似乎还不满足于只是将其作为一个概念性产品，更进一步开源了其背后的动画引擎 [POP](#)，此举大有三年前发布的 iOS UI 框架 [Three20](#) 的意味。而 POP 开源后不负 Facebook 的厚望，在 Github 上不足一个月的时间，就已经拥有了 6000+ 个 Star，非常火爆。

POP 背后的开发者是 [Kimon Tsinteris](#)，Push Pop Press 的联合创始人，曾经在 Apple 担任高级工程师，并参与了 iPhone 和 iPad 上软件的研发（iPhone的指南针以及地图）。2011年的时候 Facebook 收购了他的公司，此后他便加入了 Facebook 负责 Facebook iOS 版本的开发。

如果你打开 Push Pop Press 开发的 AI Gore 这款 App，你会发现交互和动画与 Paper 几乎如出一辙。对，他们都是 Kimon Tsinteris 开发的。

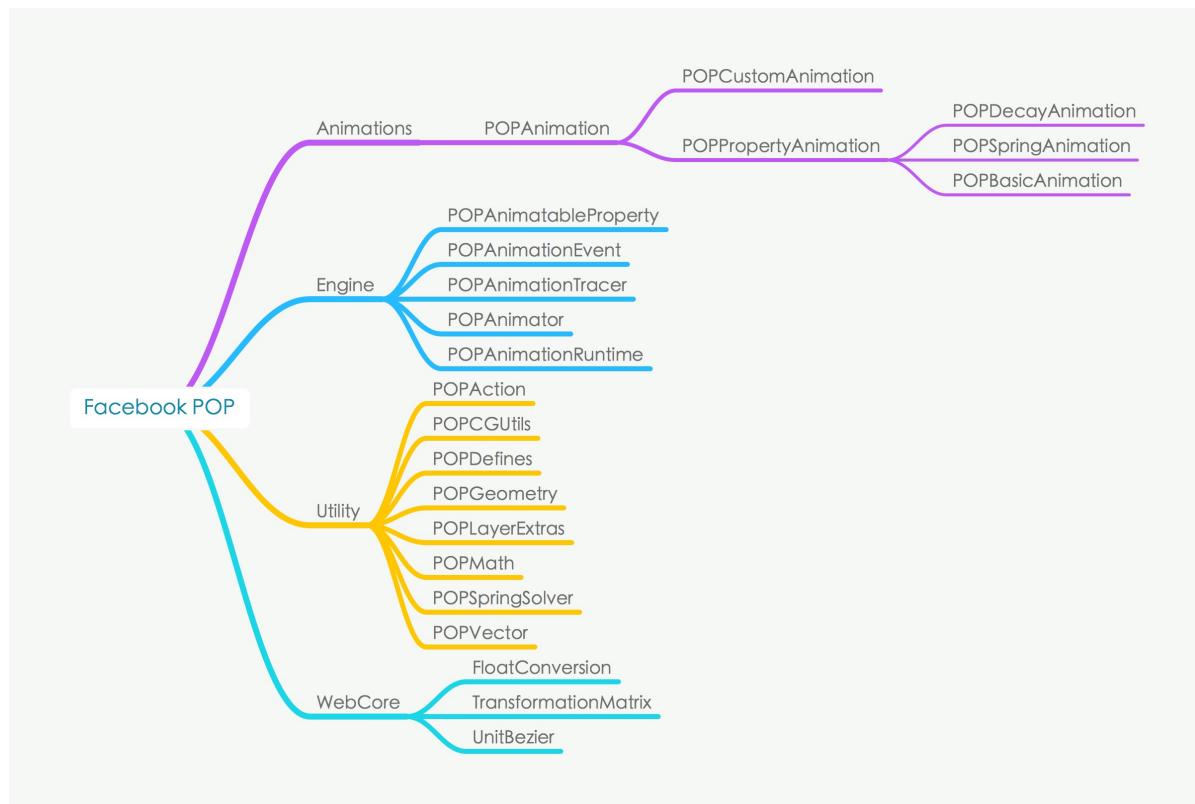
不满于 Apple 自身动画框架的单调，Push Pop Press 致力于创造一个逼真的，充满物理效应的体验。POP 就是这个理念下最新一代的成果。

POP 使用 Objective-C++ 编写，Objective-C++ 是对 C++ 的扩展，就像 Objective-C 是 C 的扩展。而至于为什么他们用 Objective-C++ 而不是纯粹的 Objective-C，原因是他们更喜欢 Objective-C++ 的语法特性所提供的便利。

## POP 的架构

POP 目前由四部分组成 (图1.1)

1. Animations
2. Engine
3. Utility
4. WebCore



POP 动画极为流畅，其秘密就在于这个 Engine 中的 `POPAimator` 里，POP 通过 CADisplayLink 高达 60 FPS 的特性，打造了一个游戏级的动画引擎。

CADisplayLink 是类似 NSTimer 的定时器，不同之处在于，NSTimer 用于我们定义任务的执行周期，资料的更新周期，他的执行受到 CPU 的阻塞影响，而 CADisplayLink 则用于定义画面的重绘，动画的演变，他的执行基于 frames 的间隔，更加稳定。

通过 CADisplayLink，Apple 允许你将 App 的重绘速度设定到和屏幕刷新频率一致，由此你可以获得非常流畅的交互动画，这项技术的应用在游戏中非常常见，著名的 Cocos-2D 也应用了这个重要的技术。

WebCore 里包含了一些从 Apple 的开源的网页渲染引擎里拿出的[源文件](#)，与 Utility 里的组件一并，提供了 POP 的各项复杂计算的基本支持。

由此通过 Engine、Utility、WebCore 三个基石，打造了 Animations。

POPAimation 有着和 CALayer 非常相似的 API。如果你知道 CALayer 的动画 API，那么你对下面的接口一定非常熟悉，想必你一定开始迫不及待想试试 POP 了，我们现在就 Jump right in。

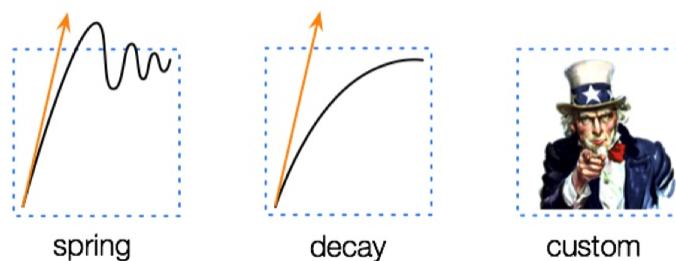
因为篇幅原因，下面的代码并不是完整代码，你可以到 <https://github.com/kevinzhou/pop-handapp> 获取我们的示例 App。

# 基本类型

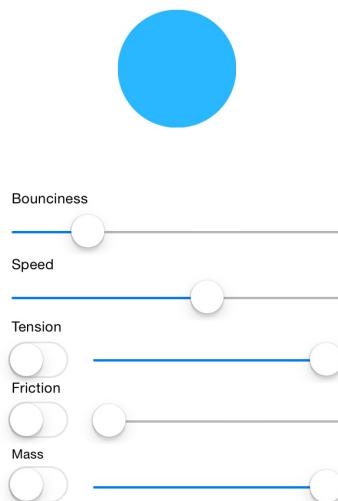
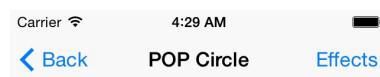
## Spring Animation

Ease-in Ease-out 这些可能你已经非常熟悉，这是动画的动作标配了，不过 POP 觉得只是这样显然太无聊，提供了两个非常不同的动画模式，第一个就是 Spring Animation。

(图 2.1)



Spring Animation 由诸多的复杂参数来控制，展现了一个非常风骚的姿势。（图 2.2）



- Bounciness 反弹 - 影响动画作用的参数的变化幅度
- Speed 速度
- Tension 拉力 - 影响回弹力度以及速度
- Friction 摩擦力 - 如果开启，动画会不断重复，幅度逐渐削弱，直到停止
- Mass 质量 - 细微的影响动画的回弹力度以及速度

Tension, Friction, Mass 这三个参数的作用很微妙，需要你在示例程序里去仔细体会。

使用 Spring Animation 的方式非常简单。（图 2.3）

```
POPSpringAnimation *anim = [POPSpringAnimation
animationWithPropertyName:kPOPLayerScaleXY];

anim.toValue = [NSValue valueWithCGPoint:CGPointMake(2.0, 2.0)];

anim.springBounciness = 4.0;

anim.springSpeed = 12.0;

anim.completionBlock = ^(POPAnimation *anim, BOOL finished) {

    if (finished) {NSLog(@"%@",@"Animation finished!");}};
}
```

通过 `[POPSpringAnimation animationWithPropertyName:kPOPLayerScaleXY]` 我们创建了一个二维平面上分别沿着 X 和 Y 坐标轴进行缩放的动画。

因此我们要使用 `toValue` 来告诉 POP 我们希望分别缩放几倍，如果你不提供 `fromValue`，那么 POP 将默认从当前的大小为依据进行缩放。值得一提的是，`toValue` 这里的值要和动画作用的属性一样的结构。如果我们操作 `bounds`，那么这里应该是 `[NSValue valueWithCGRect:CGRectMake(0.0, 0.0, 200.0, 400.0)]`。

`completionBlock` 提供了一个 Callback，动画的执行过程会不断调用这个 block，`finished` 这个布尔变量可以用来做动画完成与否的判断。

最后我们使用 `pop_addAnimation` 来让动画开始生效，如果你想删除动画的话，那么你需要调用 `pop_removeAllAnimations`。与 iOS 自带的动画不同，如果你在动画的执行过程中删除了物体的动画，那么物体会停在动画状态的最后一个瞬间，而不是闪回开始前的状态。

## Decay Animation

Decay Animation 就是 POP 提供的另外一个非常特别的动画，他实现了一个衰减的效果。这个动画有一个重要的参数 `velocity`（速率），一般并不用于物体的自发动画，而是与用户的交互共生。这个和 iOS7 引入的 UIDynamic 非常相似，如果你想实现一些物理效果，这个也是非常不错的选择。

Decay 的动画没有 `toValue` 只有 `fromValue`，然后按照 `velocity` 来做衰减操作。如果我们

想做一个刹车效果，那么应该是这样的。图（2.4）

```
POPDecayAnimation *anim = [POPDecayAnimation animWithPropertyNamed:kPOPLayerPosition]
anim.velocity = @(100.0);
anim.fromValue = @(25.0);
//anim.deceleration = 0.998;
anim.completionBlock = ^(POPAnimation *anim, BOOL finished) {
    if (finished) {NSLog(@"Stop!");}}
```

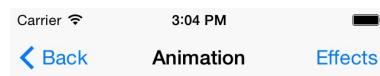
这个动画会使得物体从 X 坐标的点 25.0 开始按照速率 100 点/s 做减速运动。这里非常值得一提的是，velocity 也是必须和你操作的属性有相同的结构，如果你操作的是 bounds，想实现一个水滴滴到桌面的扩散效果，那么应该是 [NSValue valueWithCGRect:CGRectMake(0, 0, 20.0, 20.0)]。

如果 velocity 是负值，那么就会反向递减。

deceleration（负加速度）是一个你会很少用到的值，默认是就是我们地球的 0.998，如果你开发给火星人用，那么这个值你使用 0.376 会更合适。

## Property Animation & Basic Animation

POP 号称可以对物体的任何属性进行动画，其背后就是这个 Property Animation 驱动。Spring Animation 和 Decay Animation 都是继承自这个类，接下来我们通过一个 Counting Label 的例子来展现它神奇的能力。（图 2.5.0）



# 3.74

与此同时我们也使用了 Basic Animation，经典的 ease-in-out 此刻发挥了重要的作用，因为我们并不需要计数器的数值进行回弹。（图 2.5）

```

POPBasicAnimation *anim = [POPBasicAnimation animation];
anim.duration = 10.0;
anim.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

POPAAnimatableProperty * prop = [POPAAnimatableProperty propertyWithName:@"count"
initializer:^(POPMutableAnimatableProperty *prop) {
    prop.readBlock = ^(id obj, CGFloat values[]) {
        values[0] = [[obj description] floatValue];
    };

    prop.writeBlock = ^(id obj, const CGFloat values[]) {
        [obj setText:[NSString stringWithFormat:@"%.2f", values[0]]];
    };
    prop.threshold = 0.01;
}];

anim.property = prop;
anim.fromValue = @(0.0);
anim.toValue = @(100.0);

```

通过 POPBasicAnimation 的 timingFunction，我们定义了动画的方式，慢开慢停。随后通过 POPOAnimatableProperty 定义了 POP 如何操作 Label 上的数值。

readBlock 中， obj 就是我们的 Label， values 这个是动画作用的属性数组，其值必须是 CGFloat，之前我们在 Decay Animation 中操作了 bounds。

那么 values[0], values[1], values[2], values[3] 就分别对应 CGRectGetMake(0, 0, 20.0, 20.0) 的 0, 0, 20.0, 20.0。

这里我们只需要操作 Label 上显示的文字，所以只需要一个参数。通过 `values[0] = [[obj description] floatValue]` 我们告诉 POP 如何获取这个值。

相应地，我们通过 `[obj setText:[NSString stringWithFormat:@"%.2f", values[0]]]` 告诉了 POP 如何改变 Label 的属性。

threshold 定义了动画的变化阈值，如果这里使用 1，那么我们就不会看到动画执行的时候小数点后面数字的变化。到此为止，我们的 Counting Label 就完成了，是不是超简单？

## 实战

---

### PopUp & Decay Move

这个实例中我们介绍下如何将 Decay 动画和用户的操作结合起来，实现一个推冰壶的效果。

首先我们给我们的物体添加个 UIPanGestureRecognizer 的手势操作其，处理方式如下（图2.6）

```

case UIGestureRecognizerStateChanged: {
    [self.popCircle.layer pop_removeAllAnimations];
    CGPoint translation = [pan translationInView:self.view];
    CGPoint center = self.popCircle.center;
    center.x += translation.x;
    center.y += translation.y;
    self.popCircle.center = center;
    [pan setTranslation:CGPointZero inView:self.popCircle];
    break;
}
case UIGestureRecognizerStateChanged:
case UIGestureRecognizerStateCancelled: {
    CGPoint velocity = [pan velocityInView:self.view];
    [self addDecayPositionAnimationWithVelocity:velocity];
    break;
}

```

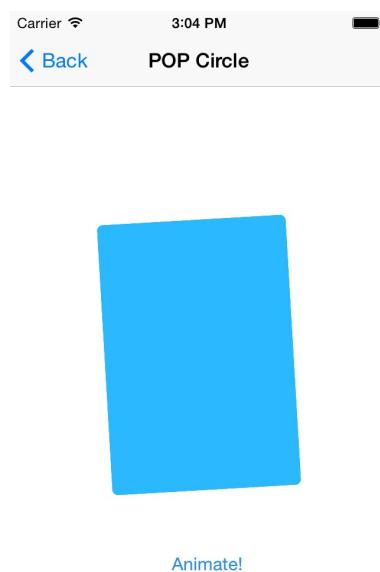
当用户触摸这个冰壶的时候，所有动画会立刻停止，然后跟随用户的指移动。通过 [pan velocityInView:self.view]; 我们获取了用户手指移动的速率然后在 addDecayPositionAnimationWithVelocity 中处理动画（图 2.7）。

```
POPDecayAnimation *anim = [POPDecayAnimation
animationWithPropertyNamed:kPOPLayerPosition];
anim.velocity = [NSValue valueWithCGPoint:CGPointMake(velocity.x, velocity.y)];
```

当用户松开手之后，冰壶会依照地球的重力在低摩擦的状态下前进逐渐停止。如果想增大摩擦力，你可以把速率乘以一个摩擦系数。

## Fly In

在这个实例中，我们介绍下如何结合两个动画。实现一个像 Path 的卡片飞入的效果(图 2.8.0)。



同样保留了 Decay Move 的效果，你可以甩走这张卡片（图 2.8）。

```
POPSpringAnimation *anim = [POPSpringAnimation
animationWithPropertyNamed:kPOPLayerPositionY];
anim.fromValue = @-200;
anim.toValue = @(self.view.center.y);
```

```

POPBasicAnimation *opacityAnim = [POPBasicAnimation
animationWithPropertyNamed:kPOPLayerOpacity];

opacityAnim.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

opacityAnim.toValue = @1.0;

POPBasicAnimation *rotationAnim = [POPBasicAnimation
animationWithPropertyNamed:kPOPLayerRotation];

rotationAnim.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

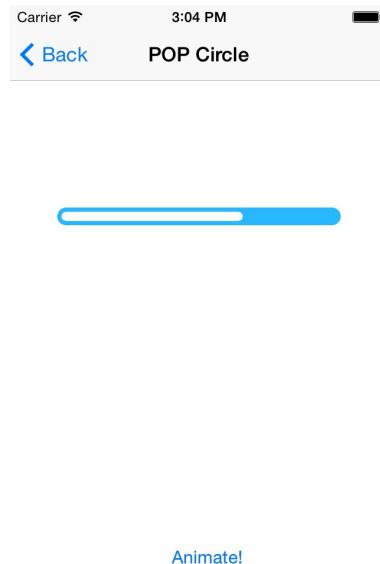
rotationAnim.beginTime = CACurrentMediaTime() + 0.1;
rotationAnim.toValue = @(0);

```

首先把我们的冰壶变成卡片，旋转一点角度。这里需要注意的是，我们使用了 duration 来定义 Basic Animation 的执行时间，beginTime 来定义动画的开始时间。beginTime 接受的是一个以秒为单位的时间，所以我们使用了 `CACurrentMediaTime()` 获取了当前时间，然后加上延迟时间。

## Transform

这个实例是真的酷极了的效果，我们将实现一个用户点击后播放按钮转换为进度条容器的变形效果。（图 2.9.0）



首先我们创建一个进度条，这个真是我最拿手的事情了。（梗请见 [PNChart](#)）通过 `lineCap` 和 `lineWidth` 我们调整进度条的样式，然后使用 `UIBezierPath` 定义了进度条的走向。（图2.9）

```
CAShapeLayer *progressLayer = [CAShapeLayer layer];
progressLayer.strokeColor = [UIColor colorWithRed:1.0 green:0.98 blue:0.98 alpha:1.0].CGColor;
progressLayer.lineWidth = 26.0;

UIBezierPath *progressline = [UIBezierPath bezierPath];
[progressline moveToPoint:CGPointMake(25.0, 25.0)];
[progressline addLineToPoint:CGPointMake(700.0, 25.0)];
progressLayer.path = progressline.CGPath;

POPSpringAnimation *scaleAnim = [POPSpringAnimation
animationWithPropertyNamed:kPOPLayerScaleXY];

scaleAnim.toValue = [NSValue valueWithCGPoint:CGPointMake(0.3, 0.3)];

POPSpringAnimation *boundsAnim = [POPSpringAnimation
animationWithPropertyNamed:kPOPLayerBounds];

boundsAnim.toValue = [NSValue valueWithCGRect:CGRectMake(0, 0, 800, 50)];
boundsAnim.completionBlock = ^(POPAnimation *anim, BOOL finished) {
    if (finished) {
        UIGraphicsBeginImageContextWithOptions(self.popCircle.frame.size, NO, 0.0);

        POPBasicAnimation *progressBoundsAnim = [POPBASICAnimation
animationWithPropertyNamed:kPOPShapeLayerStrokeEnd];

        progressBoundsAnim.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseIn];
        progressBoundsAnim.toValue = @1.0;
        progressBoundsAnim.completionBlock = ^(POPAnimation *anim, BOOL finished) {
            if (finished) {
                UIGraphicsEndImageContext();
            }
        };
    }

    [progressLayer pop_addAnimation:progressBoundsAnim forKey:@"AnimateBounds"];
}
};
```

首先是一起进行的 `scale` 和 `bounds` 的变化效果，播放按钮将缩小然后改变外形成为进度条的容器，在变形结束后，我们触发进度条的动画。

这里我们使用 `UIGraphicsBeginImageContextWithOptions(self.popCircle.frame.size, NO, 0.0);` 开启了绘画上下文，动画结束后使用 `UIGraphicsEndImageContext();` 清空了绘画上下文。这个主要是影响了画板的大小。

这里我们没有使用 `UIGraphicsBeginImageContext()` 而是使用 `UIGraphicsBeginImageContextWithOptions()` 以此获取一个更清晰的绘图效果。

## 值得关注的 POP 周边

- [POP-HandApp](#) 这就是本文的示例App，包含了大量动画的操作方法和上述介绍的实例。
- [AGGeometryKit-POP](#) 通过 POP 对图片进行变形操作，非常酷。
- [POP-MCAnimate](#) POP 的一个封装，可以让你更方便的使用 POP。
- [Rebound](#) POP 的 Android 部分实现，主要是 Spring 的效果，移植自 Facebook 的 rebound-js。

## 结语

POP 是一个新的里程碑，通过 POP，动画的开发门槛大大降低，并且实现了丰富的属性操作，其倡导的可中断式动画交互会革命性也值得我们仔细研究体会，想必不久就会涌现大量富有活力的 App，感谢 Facebook，感谢开源。Long live Opensource.

# Product ReDesign: 产品的迭代 - CocoaPods

到底如何引入第三方库也是很有趣的事情。直觉来说，应该直接把要引入的库拖进我们的当前项目，但是这样一来，会引发一些不同项目之间的编译问题，以及版本管理的问题。

所以 CocoaPods 作为一种优雅的解决方案就显得格外清新。

## 安装 CocoaPods

打开系统的 终端 (Terminal)

```
gem install cocoapods
```

## 初始化 Podfile

在终端里进入我们的项目文件夹，Diary 我存放在 Home 的 SourceCode 文件夹下

```
cd SourceCode/Diary  
pod setup
```

这时候就会为你生成一个 Podfile

```
# Uncomment this line to define a global platform for your project  
platform :ios, '8.0'  
  
use_frameworks!  
  
target 'Diary' do  
  
pod 'pop'  
  
end
```

'use\_frameworks!' 的意思是将第三方库以 Framework 的方式引用，这样不同 App 间如果都引用了同一个第三方库，那么就不针对每个 App 都再创建一个那个库的实例，以达到节省系统内存的作用。

在 target 'Diary' 里加入 pod 'pop' 就完成了 pop 的引用添加。

## 安装依赖

---

最后，执行

```
pod install
```

就可以完成安装。

安装完成后帮你生成一个 .xcworkspace 的文件，在这里就是 Diary.xcworkspace，从现在开始，就要使用 Diary.xcworkspace 来打开我们的项目了。

## 第三方库

---

这可能为刚开始学习 iOS 的你打开了一个世界的大门，全世界有非常多优秀的程序员在分享他们的代码。

### Cocoa Controls

[CocoaControls](#) 是非常老牌的一个分享社区

The screenshot shows the Cocoa Controls website interface. At the top, there's a navigation bar with links for Controls, Apps, Authors, Blog, CocoaPods, and About. On the right side of the top bar are buttons for Submit Control, Search Controls, Search, Log In via GitHub, and a dropdown menu. Below the navigation is the main header with the Cocoa Controls logo and the text "3527 open source and commercial UI components for iOS and OS X.". Underneath the header are several filter options: Sort (Date), Rating (Apps), License (All, Apache 2.0, BSD, CC BY 3.0, CC BY-SA 3.0, CC BY-SA 4.0, Commercial, Custom), Platform (All, iOS, OS X), Filter CocoaPods (Yes, No), and Language (All, JavaScript (React Native), Objective-C, Swift). The main content area displays four UI component cards: BDAstroCalc (a dark-themed calculator app), BDGradientNode (a node-based gradient editor), Starburst (a circular UI element), and a fourth component whose icon is partially visible.

阅读别人的代码是最快捷的学习途径，你不妨随时浏览一下。

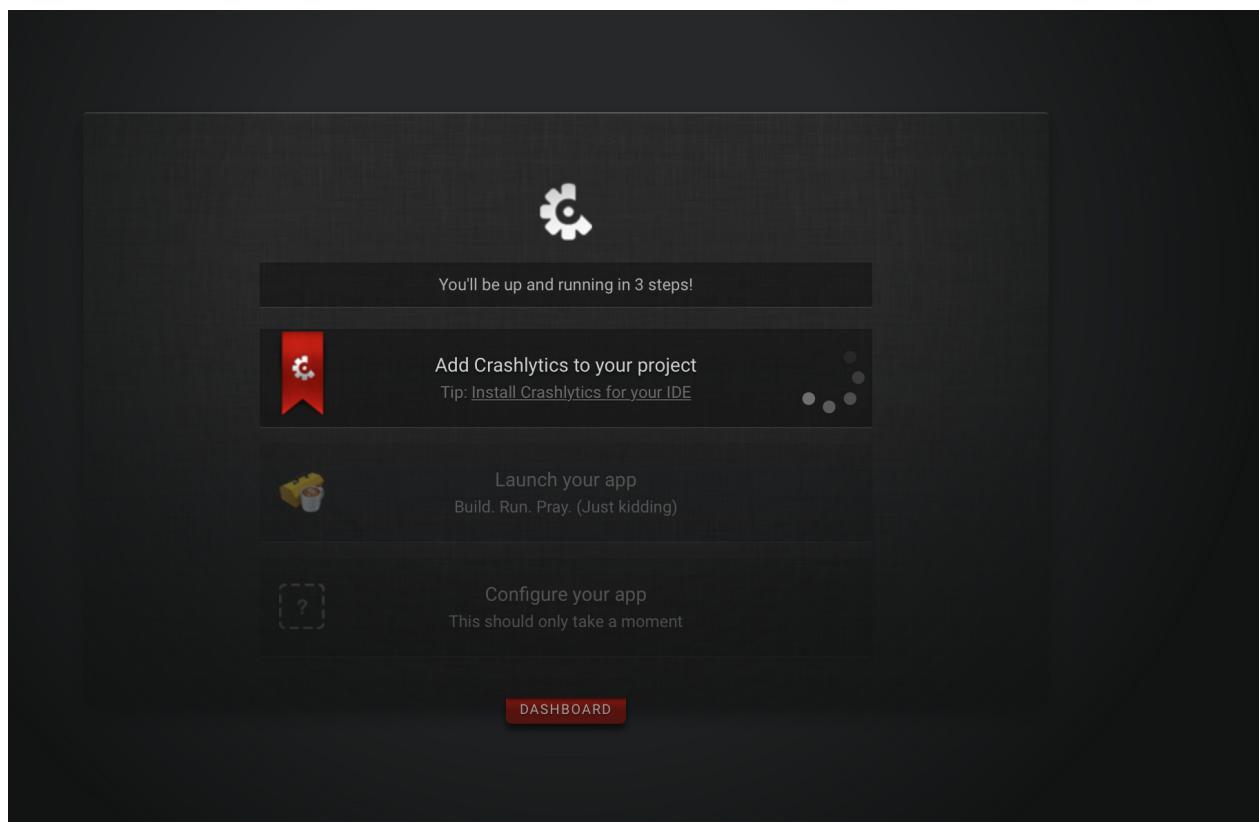
# Product ReDesign: 产品的迭代 - Crashlytics

每个 App 都有崩溃的时候，在你手里崩溃并不可怕，在用户手里崩溃才是最恐怖的。更恐怖的是，他们高喊 App 崩溃啦，你却不知道原因。

Crashlytics 是非常优秀的错误和使用情况统计服务。

## 集成

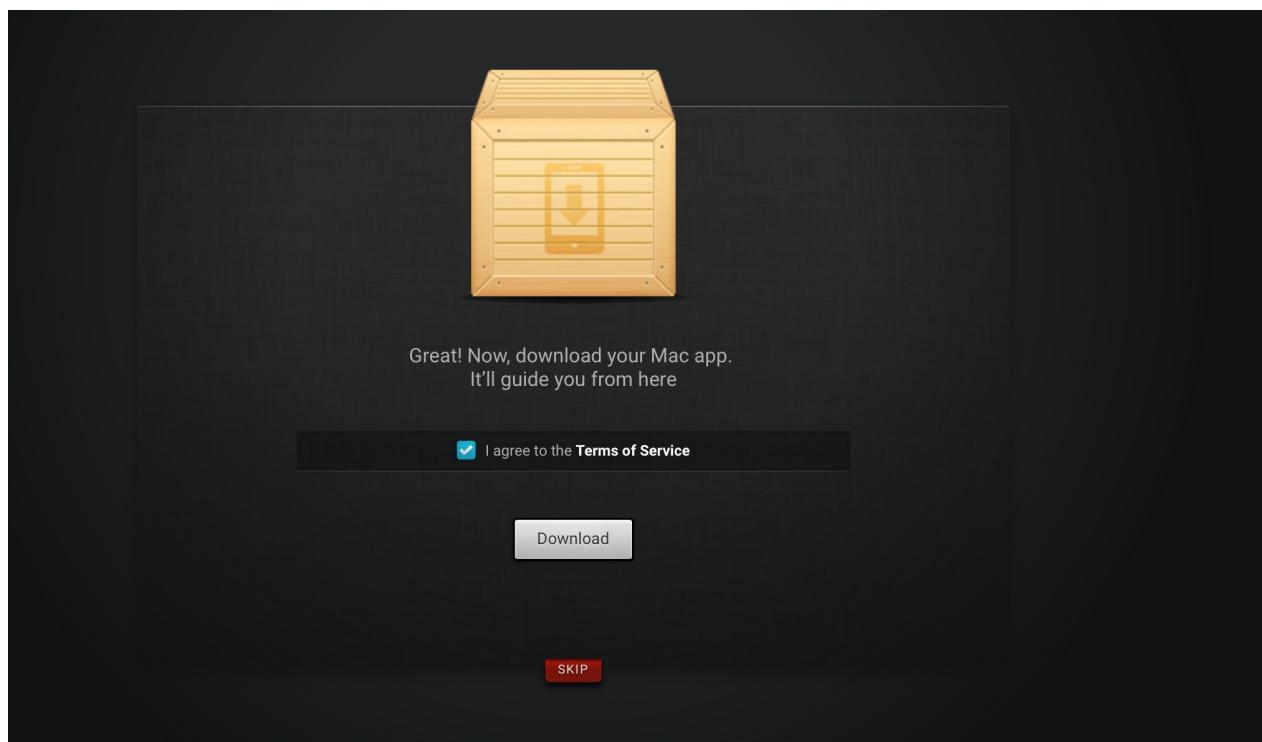
首先打开[添加 App 页面](#)



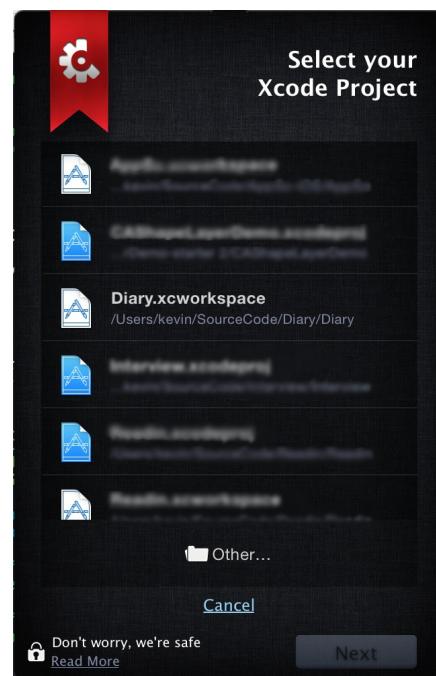
点击 [Install Crashlytics for your IDE](#) 选择 Xcode



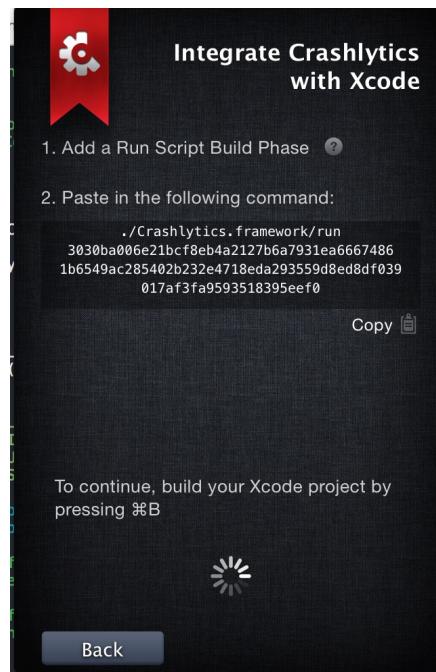
下载安装并打开



这时候系统右上角就会出现一个 Crashlytics 的托盘，选择我们的工程文件



## 复制 Run Script



## 添加到工程文件并运行 App

The screenshot shows the Xcode interface with the "Build Phases" tab selected. At the top, there are buttons for General, Capabilities, Info, Build Settings, Build Phases (which is blue and bolded), and Build Rules. Below these are sections for "Copy Bundle Resources" (3 items), "Embed Pods Frameworks", and "Copy Pods Resources". A search bar at the top right contains the text "Search".

**Build Phases**

- ▶ **Copy Bundle Resources (3 items)**
- ▶ **Embed Pods Frameworks**
- ▶ **Copy Pods Resources**

**Link Binary With Libraries (3 items)**

**Copy Bundle Resources (7 items)**

**Copy Pods Resources**

**Run Script**

Shell `/bin/sh`

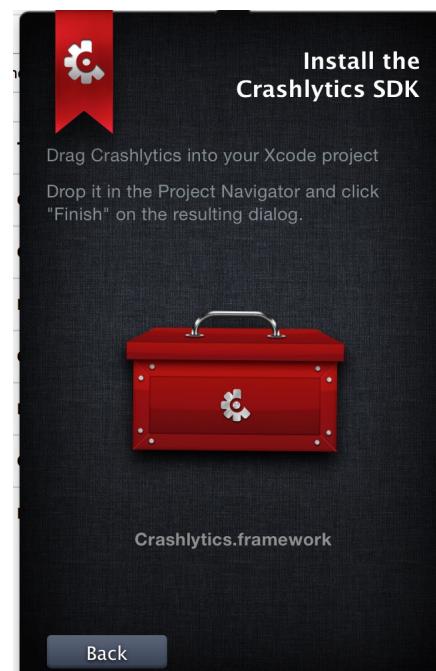
```
1 ./Crashlytics.framework/run xx
```

Show environment variables in build log

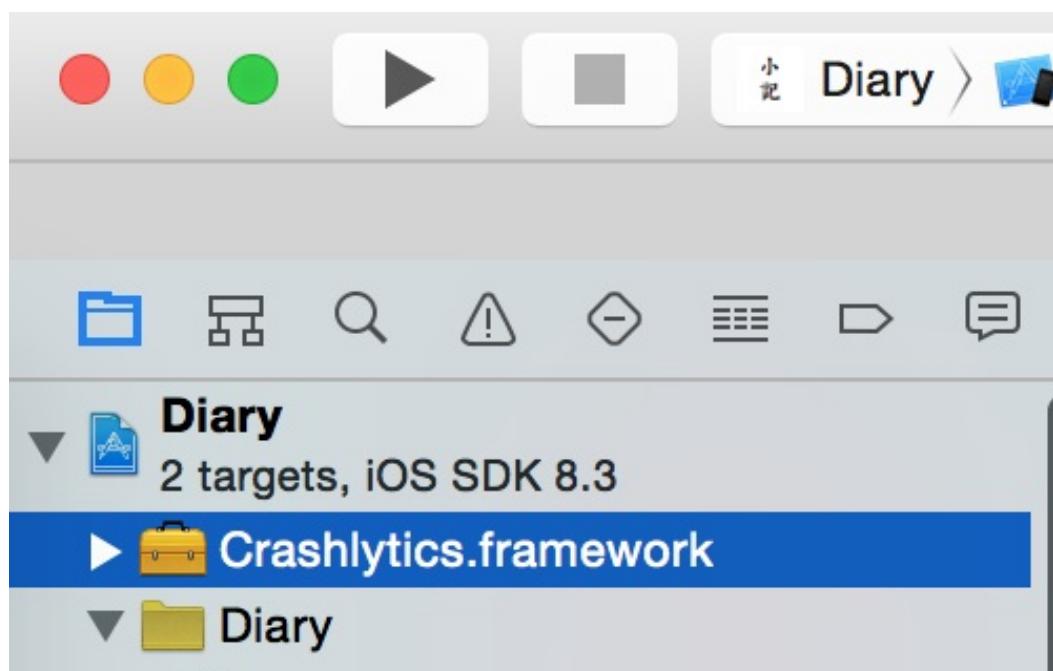
Run script only when installing

Input Files

当 Crashlytics 检测到你添加了新 App 的时候，就会提示你将 Framework 添加到工程文件



拖拽添加到你的工程里



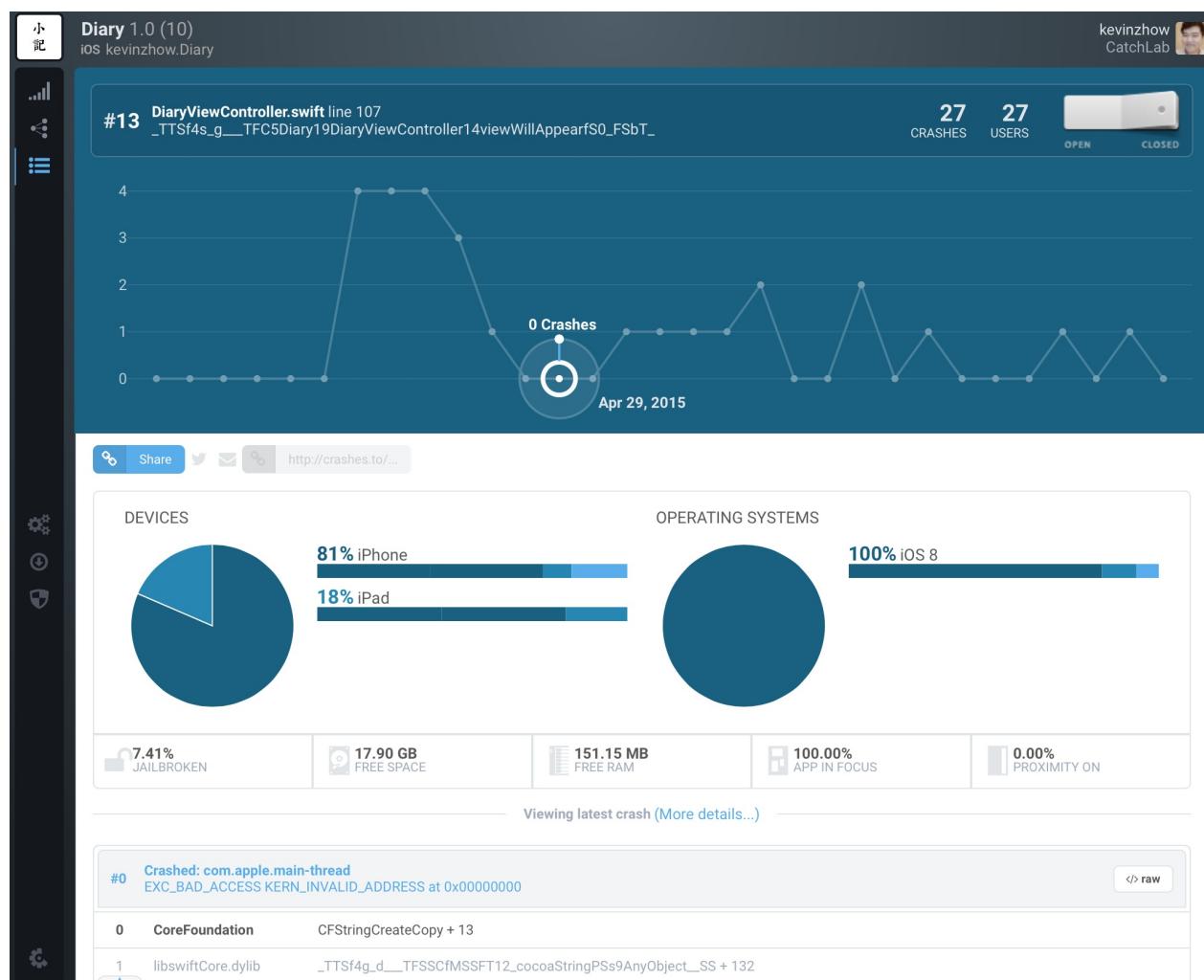
再次运行，Crashlytics 会提示你添加运行代码到 AppDelegate



完成这步之后，再次运行 App，你的 Crashlytics 就准备就绪了。

## 错误处理

当用户发生崩溃的时候，你会收到邮件提示，并且可能登陆 Crashlytics 官网查看错误信息。



Crashlytics 不仅会为你分析归类，并定位是哪里出了问题，而且还会告诉你 App 的用户留存等信息，非常实用。

# Product ReDesign: 产品的迭代 - 更通用

到这一章节，我们已经完成了产品的雏形，MVP 就是优雅的排版效果，你可以拿给朋友看一看，当我完成的时候，并没有想别人会多么喜欢这款软件，但是我很喜欢，所以当我拿给朋友的时候，收到强烈的赞赏是意料之外的。

## 好名字

名字往往将成为一款产品区分用户的标志。

小记一开始并不叫小记，它叫日记，之所以这么普通，是因为我一开始只是为了做给自己记录点东西。

当决定把这款软件推出给更多人用的时候，名字就成了一个问题。

日记并没有吸引力，也无法体现这种特质，在我睿智的朋友的建议下，我该作了小记，小字非常可爱，也能表现这款软件适合于小文本的记录与分享。

而当时朋友提出了另外一个问题，就是如果打开 App 的时候是空的，他不知道该干嘛。

所以我决定加入三首例子，来解决这个疑惑——这到底是什么，怎么用。

## 走向用户

当产品要给用户使用的时候，就要多考虑几个环节

1. 用户是否能直接理解使用方式
2. 教程的表现形式
3. 收集用户反馈

如果你的产品交互非常直白，那么可能就完全不需要教程。像小记采用的交互有一个很特殊的地方，就是双击返回。

这是为了保持界面的干净，我自己使用的话并不需要什么显式的交互。如果你希望所有人都会使用，那么就要加很多明显的按钮，文字提示。最终我选择了加入下拉返回，这个交互和 Inbox by Google 是一样的。当然这依旧很不明显，所以我们最终可能还是要加入教

程。

当产品真的发布之后，收集用户的反馈是很有用的，一方面可以知道他们的困难在哪里，另外也可以从他们提出的意见里寻找灵感。

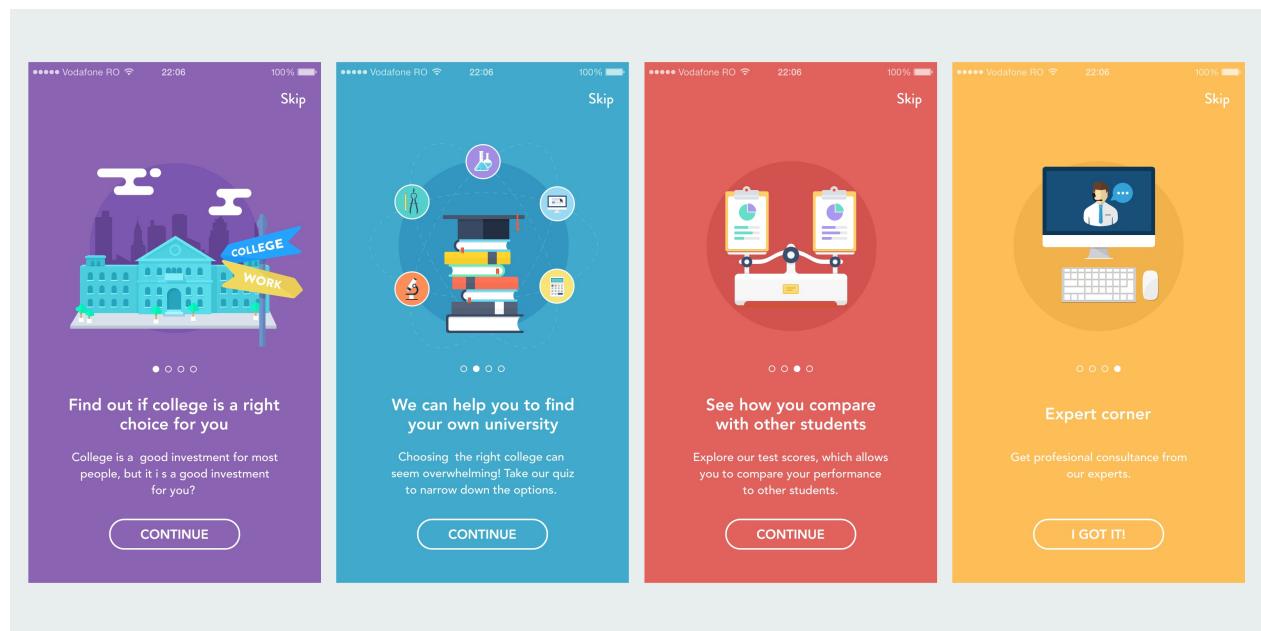
但要注意的是，他们提出的解决方案往往并不是行之有效的，解决问题的办法往往是简单粗暴，未经仔细考量。并且，如果你的产品的核心功能并不能真的解决问题，那改进所有周边的毛病都不能帮助你的产品走向成功。

## 教程

教程的形式主要有三种

1. 说明书
2. 引导式教程
3. 交互式教程

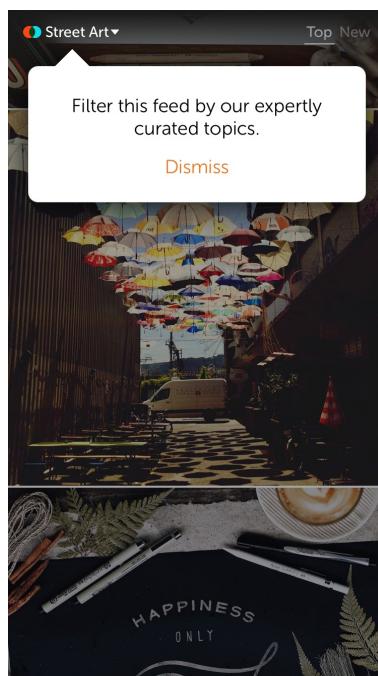
说明书式就是最常见的 App 打开几个页面滑过去的教程，这种很多人不看，但好处是让用户知道你的 App 大概是干什么用的。



很多 App 现在也开始使用这种教程来制作更新说明。



引导式教程主要是在界面增加一些文字来提示你，例如下拉刷新的时候，会有文字标识来提示状态。

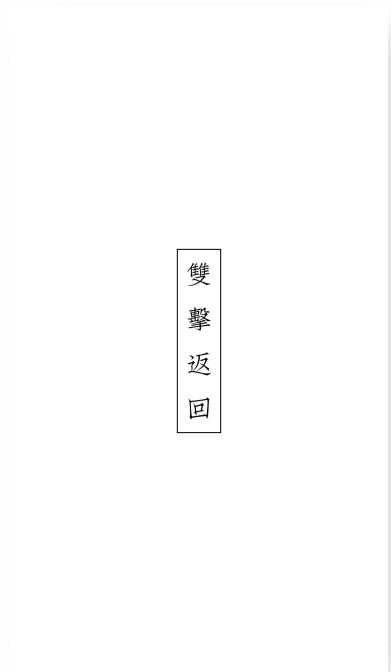


交互式的教程强制用户跟着你的指示做一遍操作，适用于如果你的产品很有交互颠覆性，不符合以前的习惯的时候。这种教程技术难度比较高——因为往往你并不知道用户会怎么点你的 App，并且，对于聪明的用户，这种交互会让人感觉极为烦躁。

教程很重要的一点是让用户知道整体的进度情况，在说明书式教程里，进度往往一目了然，而在交互式教程里，往往用户并不知道教程在何处结束，这也是用户失去耐心的一个

重要原因。

而最终给我决定给小记增加一个提示，在查看文章的时候，弹出了一个提示双击返回的页面。



# Product Market: 产品的营销 - 特质

---

到底是营销成就产品，还是产品的优秀才让营销可行是个非常有趣的问题。

有几个问题在确定营销方案的时候非常值得思考：

1. 产品解决了什么问题
2. 产品定位的人群到底是谁
3. 找谁为你的产品背书
4. 产品是否能够自发传播
5. 当我不营销的时候，产品会怎么样

这几个问题，因为产品的特质不同，都有着千差万别的答案。

## 解决的问题

---

小记 解决的问题是——一个纯粹的自我记录的地方，这种记录可以用独一无二的方式分享出去。

而对于凑热闹使用 小记 的人来说，他们往往是因为后者，而对于长久使用 小记 的人来说，是前者，分享会逐渐稳定在一个比较低的频率上。

## 定位的人群

---

而 小记 的定位的人群，我想就是一些喜欢孤独的人。

他们以前或许有能够倾诉的对象，但是随着岁月的流逝，精神的独立，他们变得非常孤独。这种孤独不是找不到可以倾诉的对象，而是他们不再相信身外这个善变的世界，但是又很想这个世界可以懂得他们。

这时候，一个懂他们的事物，成了接受度最高的东西。

## 代言人

---

代言人有时候有用，有时候没用，尤其是在互联网时代，代言的方式尤为重要。以前明产品的营销 - 产品的特质

星代言产品，大众还会相信他们使用这款产品。但是现在，可能自拍的时候漏出某个品牌的一角能引爆微博。这已经是一个反权威的时代。

所以，代言人真的使用你的产品，流露在日常的生活里成为了最好的方式。

小记 内测时候，我找到了一位非常睿智的前辈，送给他使用，当他发到朋友圈后，这种独树一帜的风格成为了焦点。

一款大家以前都没见过的产品，往往是最有潜质成为爆款的。

当朋友问我是否期待 小记 成为爆款的时候，我说， 小记 有成为爆款的潜质，但是缺少必要条件：

1. 缺少 Android 版本
2. 形式上受众并不广

中国 iOS 和 Android 的用户大概在 1:3，偏远一些地方的就更低了，这种情况下，很难形成指数级的传播。

考虑是不是做 Android 的时候，我又遇到了另外一个苦恼，当人人都可以分享这种形式的时候，必然造成一部分用户的逃离，而这部分用户，正是我的核心用户。

所以最终我并没有做 Android 版本。

## 自发传播

---

自发传播又有个专业的讲法叫自我营销，优秀的产品只需要一定的推力，用户之间的口碑传播，就可以让你的用户量滚起雪球。

## 深入了解定位

---

[《定位》](#)

# Product Market: 产品的营销 - 营销的方法

当产品特质可以形成自我营销的时候，寻找一个效果放大的渠道可以迅速积累用户。

这些渠道各种产品都在用，但产品本身会决定是不是真的有用。从这里来看，如果产品本身不够好，营销就会变成过度营销，最后早死了事。

## App 推荐

AppSolution 和 少数派 都成为了最早推荐 小记 的公众号，这为 小记 带来持续五天的大量下载。

App 推荐帐号往往有大量的粉丝以及稳定的流量，因这些帐号运营者的气质不同，也有着特定气质的受众群体。

App 推荐还存在两个特殊的群体

- 段子手
- 自媒体

段子手通常有着海量的粉丝，不过这些粉丝并不吃硬广告这一套，当然也有段子手能做到让粉丝喜欢的硬广告，比如一一天才小熊猫。这类人群能给你带来瞬间的高曝光，大流量，不过价格不菲。

自媒体人是微信时代开始火热的群体，通常都深耕在自己擅长的领域，有着比较稳定的粉丝群，如果你的 App 有很明确的使用领域，那么去找那个领域的自媒体人推荐再适合不过了。

## 产品发现社区

ProductHunt 之后，这类社区在国内开始兴起，ifanr 的 MindStore 和 36Kr 的 NEXT 都是非常好的产品发现社区。

这里往往聚集着同行业的人群。这里的用户也往往是周围朋友中的意见领袖，好玩的产品总是他们第一个发现，所以不要放过这群人。

# 广告

广告是一种成本非常高的推广方式，它更适合于电商类的产品，每个用户都可以转化成一定比例的消费者。社交产品和工具类在没有大量资金的前提下，投放这些广告需要非常谨慎。

目前转化比较好的有以下几种：

1. 微博广告
2. 应用市场广告
3. 应用市场竞价排名
4. 客户端广告
5. 微博转发抽奖

首先——如果可以提供同类产品中最好的体验，那么你完全不需要在早期尝试投放广告这种办法。

近移动端的广告对于 App 来说尤为友好，在以上几种方式里，性价比最高的就是微博转发抽奖，这也不外乎为什么每天微博上都有转发抽奖。

喵大的《Swifter》马上上市，什么都不多说了，转发本微博，抽 10 位送纸质版，祝你 Swift 之路愉快！@onevcat 亚马逊地址 [网页链接](#)

5月14日 15:19 来自 iPhone 6

阅读 21.6万 推广 | 转发 897 | 评论 139 | 喜欢 19

转发抽奖的奖品内容，抽奖方式，都会影响微博账号的形象。

你可以想象一个如果这个奖品的内容如果变成了充电宝，那么随之不管是《Swifter》这本书还是微博运营者，在 21.6 万的流量里，别说对这两者产生喜爱之情了，可能讨厌的感觉都不会有，绝大部分阅读者伸直会自动屏蔽对此类信息的思考。

转发抽奖的目的很简单，就是扩大流量增加曝光的机会，转化流量，我们通常要注意以下几个地方。

- 时间 8 - 10 点
- 转化入口（链接、地址、二维码）
- 启动（在发布的时候可以购买粉丝头条，如果有粉丝大号参与了转发，甚至可以帮助其购买粉丝头条）

## 自我营销

---

小记 是一款可以自我营销的产品，工具类的产品往往都有这种特性，当用户在社交网络分享工具制作的图文的时候，就会带来大量的下载。

当然，前提是——

1. 独特性
2. 符合用户口味的周期性变化
3. 情感共鸣

很不幸的是，当你面临一款不具备任何这些传播特性的产品的时候，营销会变得异常艰苦。

除了产品的特质，你也可以用一些方法帮助扩大这种效应

1. App 内加入明显的分享控件
2. 在微博和微信上注册你的 App 使分享出的消息有你的 App 标记
3. 在分享出的内容里打上你的 App Logo

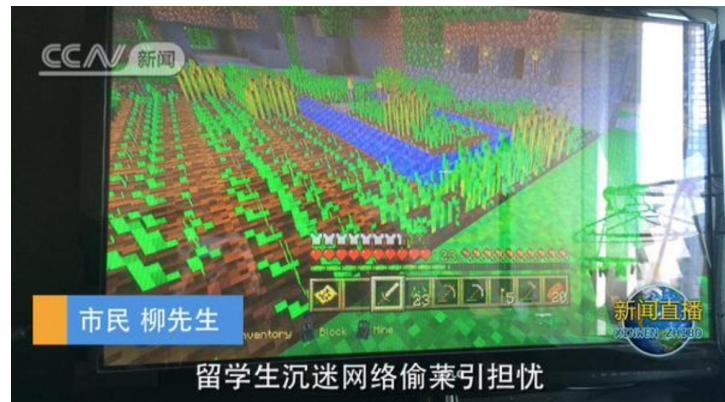
## 事件营销

---

工具类 App 的一大特色就是对特殊的、随机性的事件的融合，从而形成强大的传播效应。

这些效应形成后，如何转化成你自己的下载又是很重要的一部分。

例如 上电视，当时网友制作了一个 Minecraft 相关的段子



小记 和 上电视 都是不在用户生成的内容上打 Logo 的工具，所以用户分享出去后，对于热门内容的转化尤其重要。

这个图片被微博大号“抄”去后，我们去评论了一则“这是用一个叫 上电视 的 App 制作的，赞我让大家知道”。最终转化了 3000+ 下载，可谓经济高效。

## 营销的反思

营销并不是产品最终的解药，当产品本身不够成对需求的满足的时候，过度营销会使得用户的期待落空，你将面对人去楼空的困窘。

Google Glass 就是过度营销的典型例子，尽管这个设备一度让大家趋之若鹜，甚至可以说开启了智能设备的元年。当花费高昂费用的用户把这个产品买入手中的之后，却失望的发现与宣传的效果相去甚远。

而后微软推出的 HoloLens 更是让 Google Glass 看起来像一款玩具。

与过度营销相比，反其道而行之也未尝不可。

Apple Watch 在推出前曾经传出电池续航只有两个小时，但发布后实际的续航却可以达到两天。在聚光灯下以拉低预期的营销方式来制造惊喜，也未尝不是一招妙棋。

# Product Market: 产品的营销 - 营销的细节

---

找准了方向，掌握了方法，离真实的营销依旧有很多距离。如何吸引你的用户，让他们走向你布下的“甜蜜陷阱”来下载你的产品，是最艺术的一步。

在社交媒体时代，我们获得了大量一对多展示的机会，这本来是一件好事，但当所有人都有机会随时向用户展示东西的时候，用户就进入了被信息轰炸后的麻木状态。

## 文案！

---

文案就是你写在广告上的文字。

需要记住的第一件事情就是，所有文案上的元素，都是为了一个目的 —— 让用户阅读你的第一句话。

我们先来看一下文案一

### 标题：

用竖排来写日记

### 题图：

# 小 記



## 正文：

小记 提供了竖排的编写功能，能够展示优雅的竖排效果。

让你找到写诗的感觉，并且可以生成图片分享到朋友圈。

快来下载吧。

<http://diary.catchlab.io>

文案一是常见的推荐方式，你经常可以看到这样的微博，高喊着我们提供了什么功能，快来下载吧，然后留下了一个下载链接。

这种方式不仅不会产生用户的情感共鸣，而且还会成为用户自动过滤的内容，请接着看小记 实际使用的文案。

## 文案的个性

---

文案二

## 标题：

那年我们都想做诗人

题图：

# 小 記

正文：

小记 选了三首诗。

《季风气候》和《怎么不忧伤》，都来自我的一个发小，可能你也有这样一个曾经一起成长的朋友，然后下落不明。

人的理想并不是一成不变的，特别是青春期的时候，季风气候是他 2006 年写的，那年我们初二，那年我们都想做诗人。

他用铅笔写在了一张单薄的作业纸上，配合着歪扭的字体，连垃圾桶都嫌弃的做工，却洋溢着不羁的文采。其实那时他并不知道应该送给谁，他只是在想象那么一个姑娘，还记得那天我拜读之后，他回味良久，说道「我必须送给一个姑娘」。

于是他送给了班里那个肌肤像白玉一样的姑娘，戴着眼镜，文静，单纯。就是那个我没泡上的姑娘，被他这首诗感动了。

当我回想起下着大雪的那个下午，他告诉我他们在一把伞里牵了手，我不得不承认，我败

给了一张作业纸。

季风气候

臧甲彬

—

天空怎么灰了

我还没发觉

这落日的景色

送你的珊瑚海贝壳

攥久了像哭过

来自西西伯利亚的海风

吹过我的裤褶

这一刻

记忆开始发涩

凌乱的凋谢在坏死的脑前叶

“爱我好么”

—

“不可能的”

呵

拒不带任何罗嗦

我小心的将它剪切

然后在伤口粘贴

等待海风吹干最后一抹

体温的余热

—

没道理

是一枚太平洋的暖湿空气

飘散了我们的心

在青春的墓地

就这样平行下去

我估计我们的轨迹

差不多一米

我酝酿给这个距离

下个定义

叫做

爱你

或许这个世界上，我是最后一个拥有这首诗全文的。

2008 年，我们上了高中，分开到了不同的学校，在那之后，他得了抑郁症。

一切开始变得无力挽回，我们开始活在不同的世界。但他的才华依然在用他特有的方式横空着。

今年 2 月份的时候，我再也联系不上他，我不知道他去了哪里，也不知道他的亲人去了哪里，甚至没有人知道他去了哪里。

或许这就是他的忧伤

叫我怎么不忧伤

臧甲彬

—

那个男生宽肩膀 姑娘看的眼放光 可怜我这男朋友 空有一副热心肠 热心肠啊热心肠  
我的姑娘不见了 叫我怎么不忧伤

—

小时放肆真快活

打架撒谎耍混账

天黑之后跑回家

家里有我亲爹娘

亲爹娘啊亲爹娘

我的童年不见了

叫我怎么不忧伤

>

—

欢喜少来忧愁多

人生本来就这样

寂寞之人仍在此

临窗独唱明月光

明月光啊明月光

我的快乐不见了

叫我怎么不忧伤

在给 小记 选择例诗的时候，我脑中浮现了他的这两首诗。

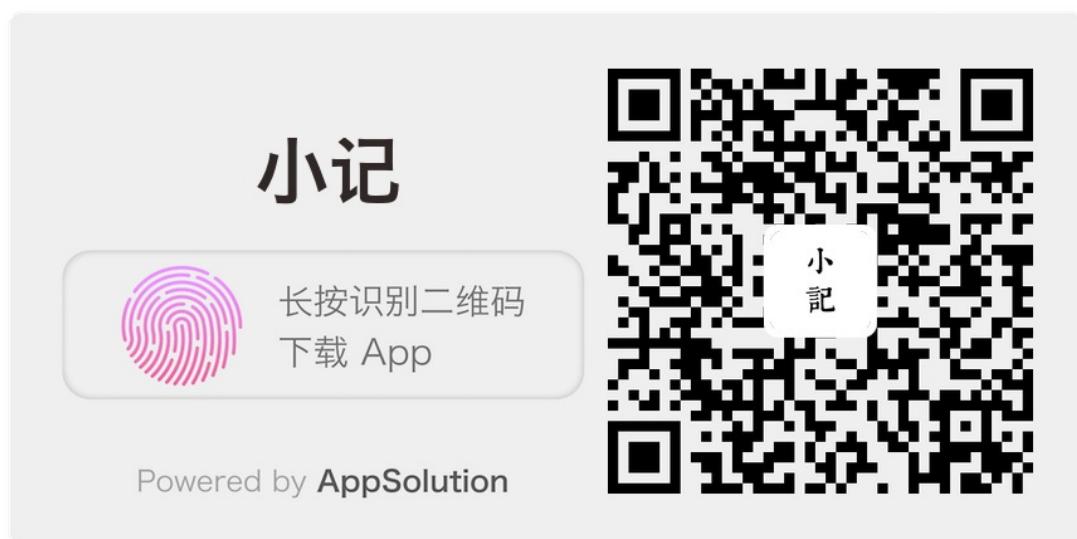
他从未有机会出版他的诗，所以我把他的诗放进 小记 里，让更多的人看到，让无论此刻在哪里的他看到。

第三首是最后选定的，圣经里的创世纪篇。

所有人都知道这两句——

上帝说，要有光，于是，便有了光。

有时候听着音乐，呆在一个寂静的地方，有些人会涌进你的脑海，可能他曾陪着你成长，可能她是一个守候在你背后的姑娘。这个世界虽有诸多纷扰，但是只有彼此之间在岁月里留下的印记才会让你恍然大悟：自己是谁，从哪里来，到哪里去。



文案二中，从标题和题图开始，这个文案都是特别的，口语化的独白，青涩的措辞，勾起了你对那些青葱岁月的回忆，用一片对友人的记忆把 小记 的概念塑造起来。

对于这样一款产品来说，谈论功能，界面，都不能让其脱颖而出， 小记 所代表的记忆符号才能真正给这款产品打上一个特殊的烙印。

而最终从用户的反馈和极高的下载转化率，也证明了这篇文案的成功。

文案，从用户注意了你的第一句话开始，就应该勾起他们的好奇心，像坐滑梯一样，一路读下去，直到最后。利用你的文字构造出一个情绪氛围，让最后的下载成为一件不得不做的事情。

## 完美的的下载环境

---

文案一中附带的下载链接要经过浏览器多次跳转，难以点按，并且在微信或者其他平台，还可能遭遇跳转拦截。

二维码的形式可以放置一个优化后的可直接跳转 AppStore 的链接，并且更加人性化。

在文案里，非常有必要借助各种技巧来打造一个完美的下载环境。

## 深入了解文案写作

---

[《文案写作手册》](#)

# Product Market: 产品的营销 – 内测你的产品

现在你已经了解了营销的基本方法和思路，那么你可以先找一个小群体测试自己的点子。无论是积极的反馈还是令人沮丧的反馈，都可以帮助你进一步改进产品。

App 发布之前内测是必须的，一般人可能以为自己拿几个手机测试一下还不就得了？但是事实远不是这样，由于你对自己的 App 的行为非常稔熟，而且使用场景有限，设备有限，导致的结果就是——温室里的花朵测不出 Bug。

在用户中也存在一种人，总是能在你以为完美的版本里，找出重大的 Bug。

内测不仅能够除掉 Bug，也是唤起早期用户参与感的重要方式。

## TestFlight

曾经一度是最好的第三方测试平台，现在已经被 Apple 收购了。当你提交了到 iTunes Connect 后，可以在 预发行 这里开启 TestFlight



开启之后需要再点击 提交以供 Beta App 测试 填写基本的 App 资料，然后等待 1-2 天就可以向外部测试员发布。

点击外部测试人员来添加人员

版本 预发行 定价 App 内购买项目 Game Center 评论 报刊杂志 更多 ▾

构建版本 内部测试员 外部测试员

测试员（已添加 0 名，最多 1,000 名）[+](#)

点按“+”以添加外部测试员。

外部测试员可以使用 TestFlight 安装您测试的任何 App 的新的构建版本。  
他们不必是您所在机构的成员。

内部测试人员无需经过 Apple 的审核，但是也有着诸多限制条件，一个 Apple ID 只能参与同一个开发者发起的内部测试，而且可以登录你的 iTunes Connect 后台，存在一定的安全隐患。

# 寻找内测人员

通过审核之后你就可以到 [MindStore](#) 或者 [NEXT](#) 这样的社区寻找内测人员了。

内测的方式也极为重要，发起话题，积极互动，并且最好附上截图。

下面是 小记 内测时的内容形式：

周楷雯Kevin 发布于3月27日 14:52

分享到:

## 小记 (M)

纵书, 淳朴的书写体验。说一款你知道的「纯粹到飙泪」的 app, 并说出为什么它打动你, 就有机会获得邀请码。

### 产品图片

子 現在  
二零一五年三月二十八日

临窗倚榻明月光  
明月光明月光  
我的快樂見了  
叫我怎麼不憂傷

二零一五年三月二十八日

小記

App Store

### 65 个人投了票

然而仅仅发布到这些平台上还是不够的，找到你熟悉的有影响力的人帮你在社交网络上转发，如果你的作品足够独特，必然会有更多的人参与进来。



# Product Extra: Motion Graphic

这两天和 Ray 一起吃拉面的时候，除了关注那位可爱服务员之外，他还煞有介事的跟我讲——“我发现了一个超屌的设计师，balabala”，其实时至今日我已经不知道如何定义超屌的设计师这个概念了，就像超屌的开发者一样，这个问题探究到最后只会变成一句感叹——人类最屌竟然只能做成这个样子。

不过，还是捧捧 Ray 的场啦，我问道“是何许人也？”。

他拿过来几乎没有信号的 iPad 给我看，你看这个人，这个人，目光从 Safari 地址栏的 .cn 这个坑爹的后缀扫过后，赫然发现一个完全不知道的名字 [Marcus Eckert](#)。虽然不知道名字，但是接下来的内容绝对是如雷贯耳——[Wide Sky](#)。

这段留着你来点开上面的两个链接欣赏下



[链接](#)

个人英雄主义如果不是个人能力超强的话，往往会变成一个自我重伤的事情，但是很显然 Marcus Eckert 把这一点发挥到了极致， Meek for iOS 太震撼了。所以按捺不住心中的激动，我对 Motion Graphics 展开了一番问道。

# 寻路

Motion Graphics 这个词我不久前第一次听说，当时是一米天给 [CatchChat](#) 做介绍动画的时候，跟我说他们是高大上的 MG，不过我的理解是怎么现在搞啥都得发明个名词来方便忽悠了，不过只要是好东西做做忽悠也没啥，但是此事也就这样没放在心上。

这次我借机好好研究了一下。

这里正好有一段关于 MG 是啥的 Youtube 视频，应该是某个专业做 MG 的团队出品的，质量一般般，不过我觉得是体现了绝大部分从业者对这个的理解。



[视频链接](#)

所以我想了想，之前我应该也做过一个类似的东西，之前研究 Facebook POP 的时候，有个 [Transform](#) 的部分正是异曲同工。

但是说实在的，探究 Motion Graphics 的过程是不愉快的，因为看得越多，越发现了大家在用那么几种相似的变换形式来来回折腾。这种体验相似于其它的行业，最顶尖的天才研究出来一些新式的效果，其他人抄袭个改进。但是想想如果人类最顶尖的想象力和效果就是这样了，还是很伤心的。

# 问道

Motion Graphics 在 AE 里做应该是相对简单的，不过 Marcus Eckert 用代码实现了这些效果，就让我很感动了，他自己也说，其实对他而言更多的是技术上的实验，所以我也思考了下如何从技术上实现这些效果。

Actually，动效无论是软件还是在动画里，用多了其实都会让人吐的。

好好，先聊聊 iOS 里怎么实现动效。

## iOS 里的动效初探

简单的飞来飞去，大小改变什么的就很简单了，聊聊复杂的形变问题。

还好我之前玩过 Blender，我的思路第一个就是对 Bezier 曲线进行改变，这个就比较类似关键帧动画。

比如我们如何把多边形进行变形。

先画五边形

```
self.aPath = [UIBezierPath bezierPath];

// Set the starting point of the shape.
[self.aPath moveToPoint:CGPointMake(100.0, 0.0)];

// Draw the path.
[self.aPath addLineToPoint:CGPointMake(200.0, 40.0)];
[self.aPath addLineToPoint:CGPointMake(160, 140)];
[self.aPath addLineToPoint:CGPointMake(40.0, 140)];
[self.aPath addLineToPoint:CGPointMake(0.0, 40.0)];
[self.aPath closePath];
```

然后再搞一个变形后的曲线

```
//B path

self.bPath = [UIBezierPath bezierPath];

// Set the starting point of the shape.
[self.bPath moveToPoint:CGPointMake(0.0, 0.0)];

// Draw the lines.
[self.bPath addLineToPoint:CGPointMake(100.0, 40.0)];
[self.bPath addLineToPoint:CGPointMake(120, 140)];
```

```
[self.bPath addLineToPoint:CGPointMake(-10.0, 100.0)];
[self.bPath addLineToPoint:CGPointMake(0.0, 40.0)];
[self.bPath closePath];
```

那么先画出 A path

```
CAShapeLayer *shape = [CAShapeLayer layer];
shape.drawsAsynchronously = YES;
shape.frame = self.view.bounds;
shape.path = self.aPath.CGPath;
shape.lineWidth = 3.0f;
shape.lineCap = kCALineCapRound;
shape.lineJoin = kCALineJoinRound;
shape.strokeColor = [UIColor whiteColor].CGColor;
shape.fillColor = [UIColor redColor].CGColor;
[self.view.layer addSublayer:shape];
```

接下来要做的就是加入个动画让 A path 变成 B path

```
CABasicAnimation * pathAnimation = [CABasicAnimation animationWithKeyPath:@"path"];
pathAnimation.fromValue = (id)[self.aPath CGPath];
pathAnimation.toValue = (id)[self.bPath CGPath];
pathAnimation.duration = 0.5f;
pathAnimation.autoreverses = NO;
pathAnimation.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
[shape addAnimation:pathAnimation forKey:@"animationKey"];

shape.path = self.bPath.CGPath;
```

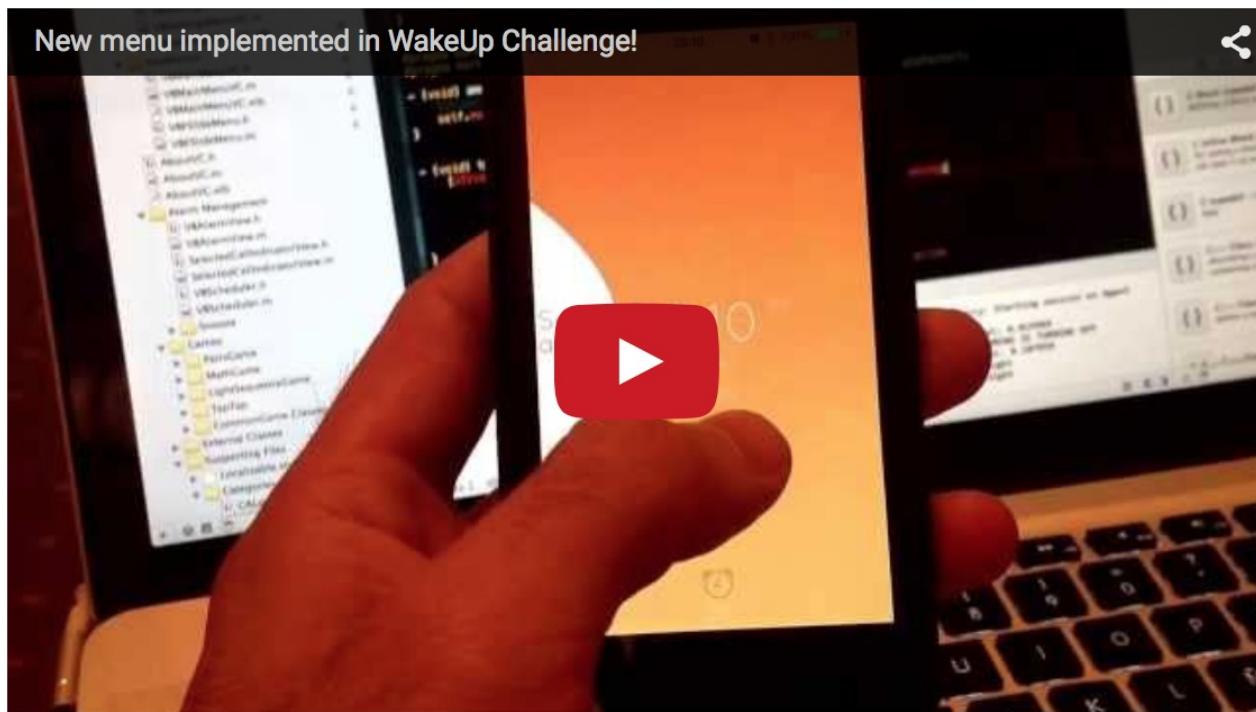
Build Run 一下就可以发生这个渐变动画了，瞬间有小学时候玩 Flash 的感觉有木有。  
timingFunction 可以经过复杂的定义实现一个比较生动的动画。

通过 addQuadCurveToPoint 这个也可以实现控制点方式对形状进行控制，也可以写一个函数进行 Path 的 Smooth 的操作。这方面我们已经有 [先行者](#) 做了尝试

同时我也给 [PNChart](#) 0.6 版本加入了一个实时更新数据的功能，用的就是这个方法。

## 动效物理效果进阶

之后我又发现了一位先驱。



[视频链接](#)

这效果看起来非常高大上，但是知道原理后又不是那么神奇。

```
[self.bPath moveToPoint:CGPointMake(0.0, 0.0)];
[self.bPath addQuadCurveToPoint:CGPointMake(0.0, ScreenHeight)
    controlPoint:CGPointMake(100.0, ScreenHeight/2.0)];
```

通过 `addQuadCurveToPoint` 增加控制点，然后添加一个 Dummy 的 `UIView` 跟随控制点，在释放的时候给 Dummy 一个 Spring 的 Animation 即可，或者使用 `UIDynamic` 增加一个高空坠落的效果。使用 `CADisplayLink` 同步 Path 和 Dummy View 的数值。

在 Ray 提到波纹没有像 Android L 里面那样跟手之后，我想到确实有个可以改进的地方，就是在移动手指的时候 `controlPoint` 的 Y 坐标可以跟随手的位置进行变动，这样波纹的尖端就会和手一直。

## 动效果冻效果

那么随着进一步探索，又找到一位复制 Skype 的 [果冻菜单](#) 效果的先驱。

实现的原理也是一样的，不过是创建两个 Dummy View，你可以研究下这篇文章。

后来作者创建了一个独立的库 [AHKBendableView](#)。

有兴趣的话可以研究下，实现的方式就更加复杂了，不过原理还是一样的。

```
let path = UIBezierPath()
path.moveToPoint(CGPoint(x: 0, y: 0))
path.addQuadCurveToPoint(CGPoint(x: width, y: 0),
controlPoint:CGPoint(x: width / 2.0, y: 0 + bendableOffset.vertical))
path.addQuadCurveToPoint(CGPoint(x: width, y: height),
controlPoint:CGPoint(x: width + bendableOffset.horizontal, y: height / 2.0))
path.addQuadCurveToPoint(CGPoint(x: 0, y: height),
controlPoint: CGPoint(x: width / 2.0, y: height + bendableOffset.vertical))
path.addQuadCurveToPoint(CGPoint(x: 0, y: 0),
controlPoint: CGPoint(x: bendableOffset.horizontal, y: height / 2.0))
path.closePath()
```

使用 CADisplayLink 同步 bendableOffset 和 Dummy View 的移动。

## 再续前缘

---

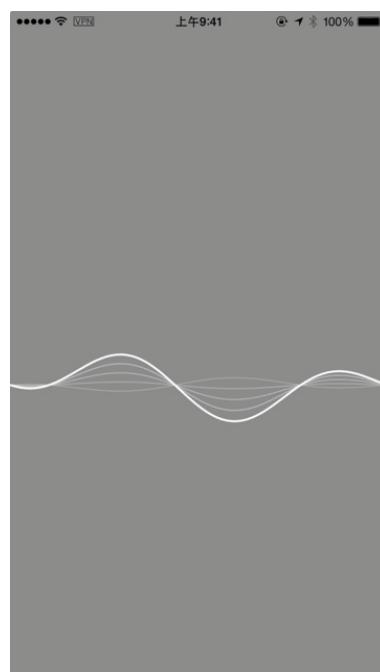
研究到这里，回想 Marcus Eckert 的工作，觉得越发碉堡，如果要在三维空间实现这些效果，那么还要再去搞 OpenGL，不禁为此觉得潸然泪下，无论是工程师还是设计师，局限于自己的领域是无法做出极具创造性的工作的。

望着窗外广州大道的车灯，被雨夜的风吹的更清醒一点了。

# Product Extra: Waver 声波效果

在上一篇研究了动效之后，这段时间一直在琢磨如何做一些更有趣的东西，所以，昨天我开源了一个新的声波库——[Waver](#)，拥有非常动人的声波效果，在此要感谢[SCSiriWaveformView](#)这个项目，Waver 在他的基础上改成了 Block 的使用方式，同时声波采用我最熟悉的 UIBezierPath 和 CAShapeLayer 实现，并做了一些逻辑上的优化，实现了 8 倍的性能提升。

采用 UIBezierPath & CAShapeLayer 的另外一个好处是更方便对初始形态进行调整，像 Siri 那样可以从圆形变成线条。



不过对此而言，怎么使用不是最重要的，重要的是怎么实现这样的效果，So，Let's have some fun！

## 原理

在大概半年多前的时候，iOS 群里曾有过关于如何实现像 Siri 的声波效果的讨论，当时提出的第一个解决方案是 [FFT](#)，网易公开课有斯坦福[相关的课程](#)。

傅立叶原理：任何连续测量的时序或信号，都可以表示为不同频率的正弦波信号的无限叠加。

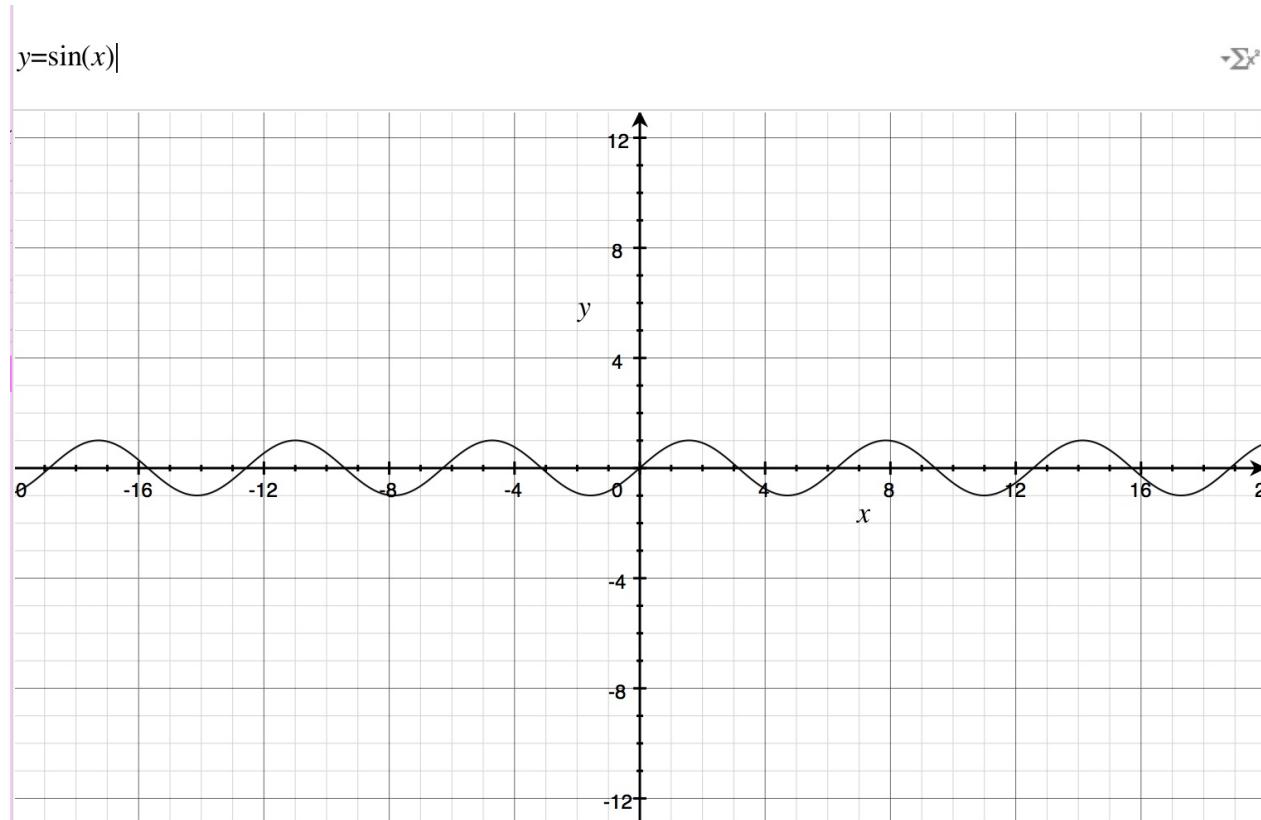
不过解释或者推导再应用这个原理显然不那么有趣，我想从一个纯粹的逻辑角度出发去解决这个问题。

Mac 上有个非常有趣的软件，叫做 Grapher，你可以在里面输入我们玩的公式，也可以[直接下载](#)我们的试验文件。

## 屏幕上的波纹

长按 Home 叫出 Siri，千里之行的第一步就是在屏幕上画出一个从左边缘到右边缘的周期性线条。

要不就试试三角函数 Sin 呀。



对这个基本函数我们需要以下几个操作做基本调整

### 1. 函数周期变化的 x 范围限制

符合手机屏幕的宽度，假设为 32。

### 2. 在 x 内变化的周期数限制

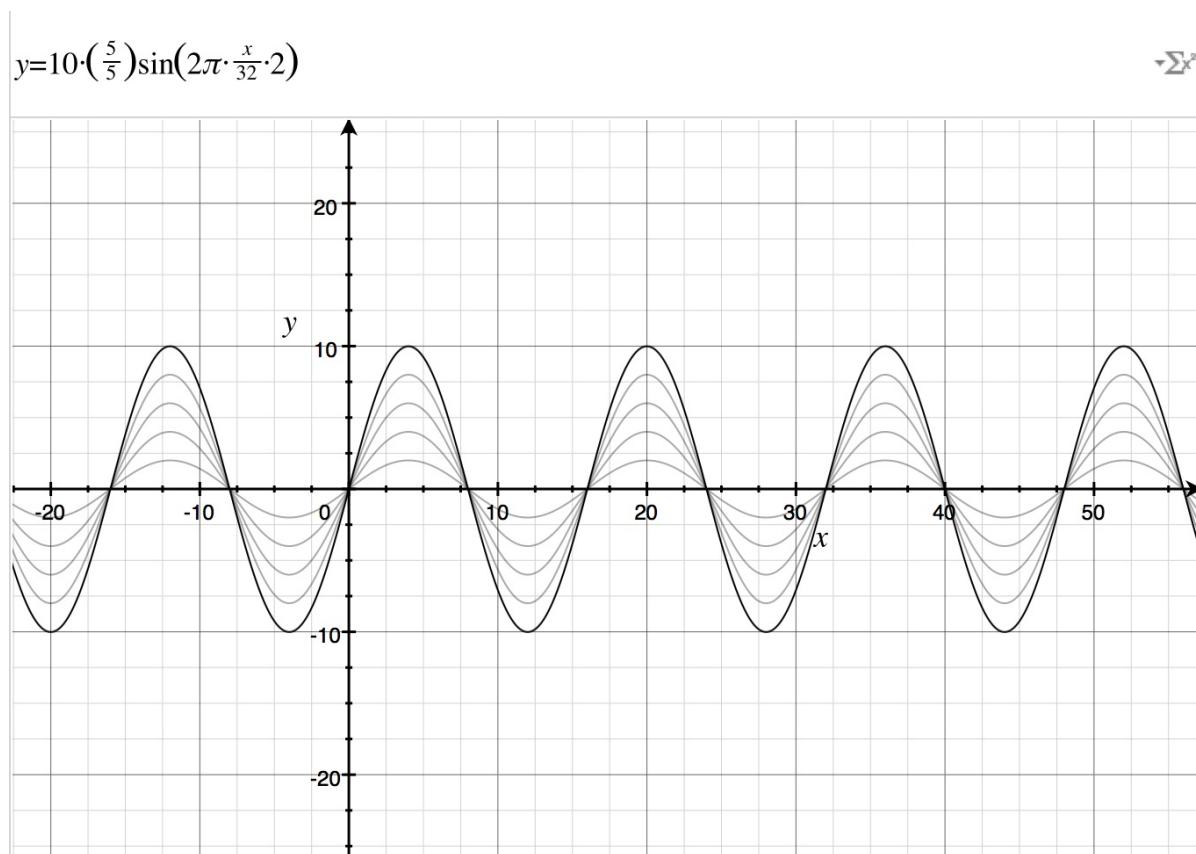
假设我们需要 2 个周期变化。

### 3. 波峰限制

我们需要峰值不超过我们 UIView 容器的高度，所以假设 UIView 高是 20，那么峰值应该限制在 10 以内。

### 4. 五个波纹

依次波峰递减 1/5。



## 波纹的限制

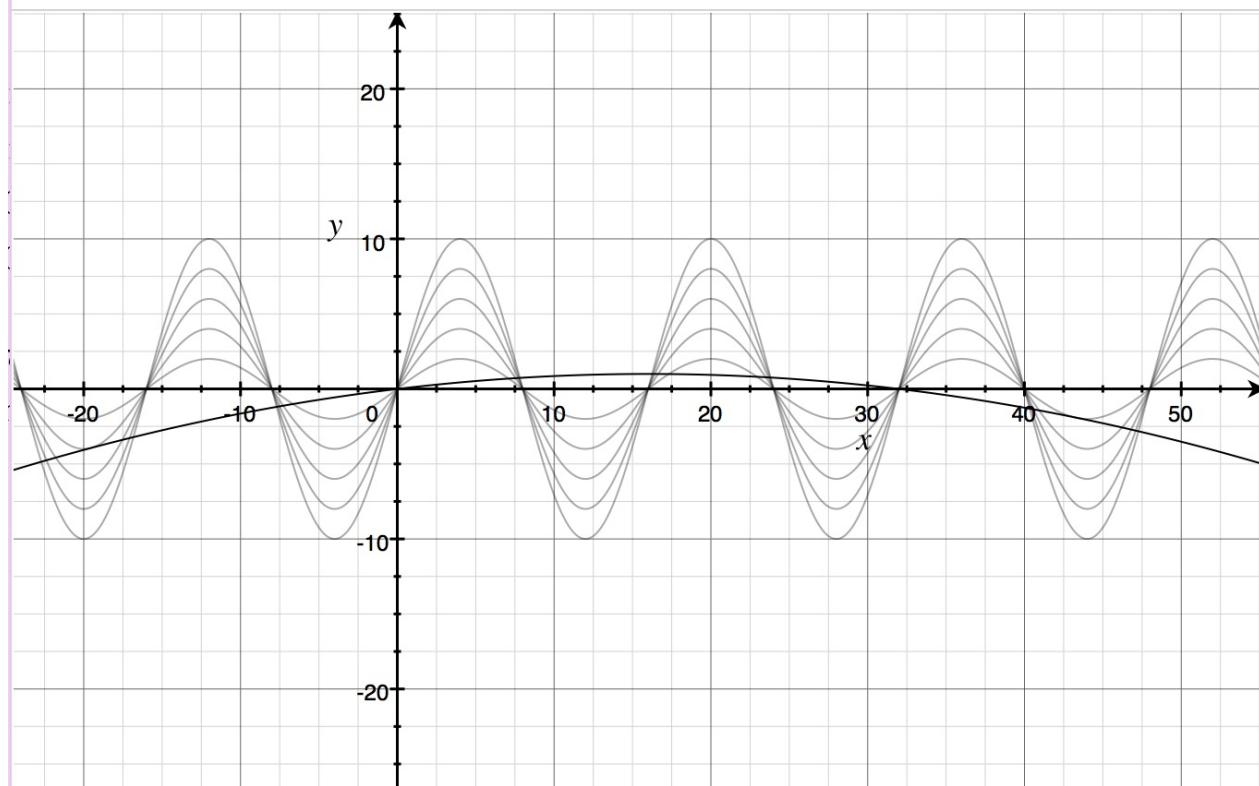
上面已经非常接近我们想要的效果了，但是还有一个比较重要的，就是最终出来的效果应该是越靠近屏幕中间的位置，波峰越大，靠近屏幕边缘的地方，无限接近于静止。

那么我们还需要一个参数（一元二次方程）来调整。

满足在 x 的范围内，值从 0 ~ 正数值变化。

$$y = -\left(\frac{x}{16} - 1\right)^2 + 1$$

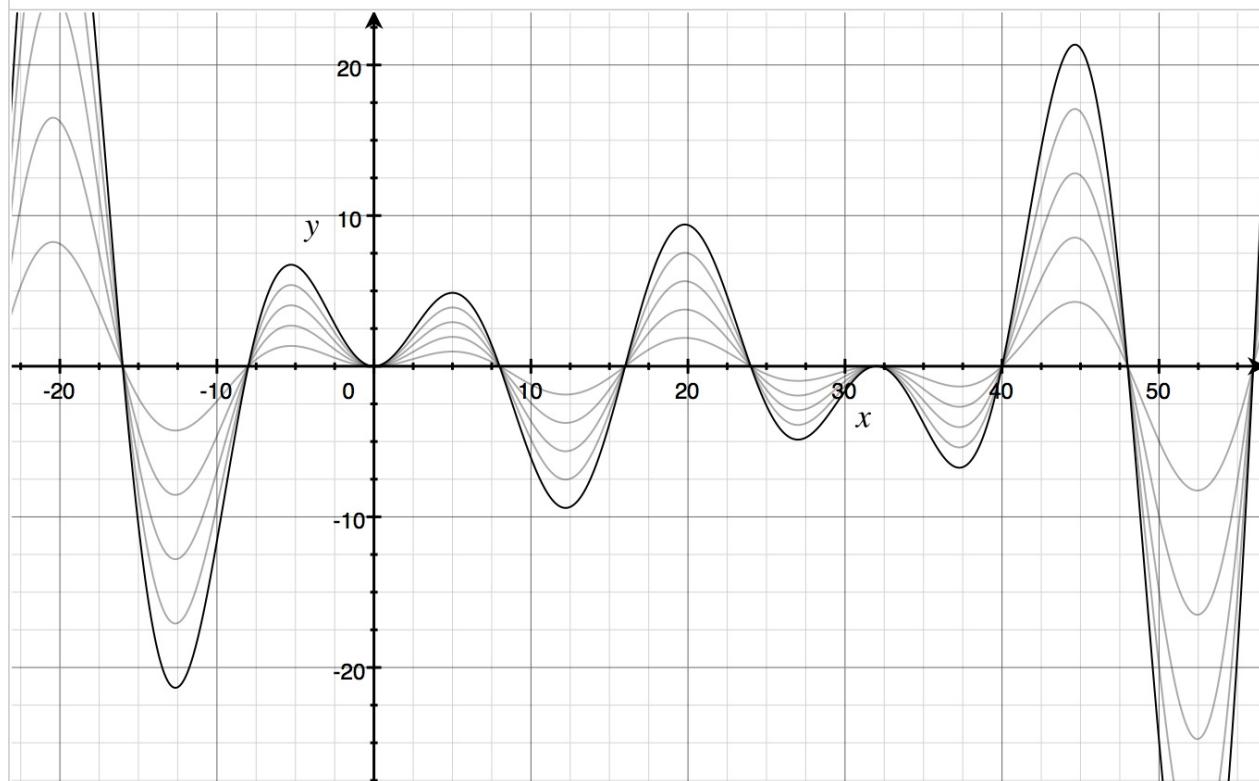
→  $\sum x^2$



那么这两个函数相乘的时候，就能实现我们想要的效果

$$y = \left(-\left(\frac{x}{16} - 1\right)^2 + 1\right) \cdot 10 \cdot \left(\frac{5}{3}\right) \sin\left(2\pi \cdot \frac{x}{32} \cdot 2\right)$$

→  $\sum x^2$



## Animate

---

如何让这个动起来呢？

满足以下两个条件

1. 一个用来调整波峰的参数

把声音的音量处理后作为参数传入，于函数相乘。

2. 循环进行  $x$  变化的参数

使用 CADisplayLink 作为循环器，声明一个位移量，每次循环的时候进行递增，然后传入我们的函数。

# Product Next 新篇章

---

以 小记 作为主体的第一本 Producter 到此结束，希望这本书可以帮助你开始产品人的生涯，期望可以在下一本中与你继续探讨新的产品。

最后，要感谢 Ray (@rayps) 为本书设计的封面和官网，或许你可能会讶异——“为什么你不自己做呢？”

独立完成一款好的产品是一种修养，让产品变得更完美是一种艺术。

Yep!

其实只是因为我懒。