

iOS9特性

支持机型：

iPhone：4S、5、5c、5s、6、6 Plus、6s
和 6s Plus

iPad：2、iPad Retina(即iPad3和iPad4)、全部iPad Air、iPad pro、
全部 iPad mini

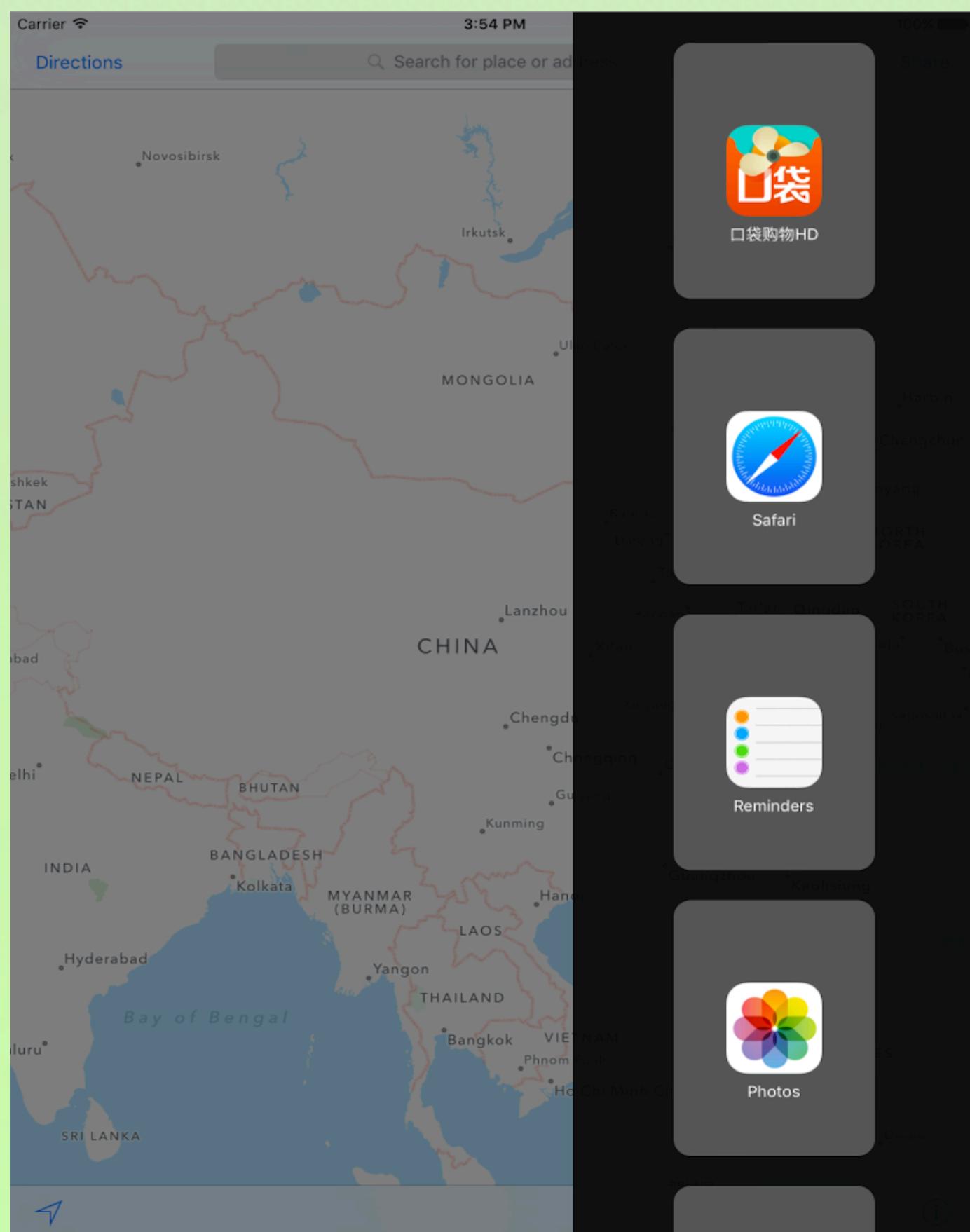
iPod touch：第五代和第六代

注意：

iOS9不仅为iOS9提供相关特性。也在为以后新元素做好了铺垫

SlideOver

iPad功能，类似从上侧往下的翻盖，用户也可以通过右侧边栏向左滑出临时出现和交互的滑动覆盖，从中选择app



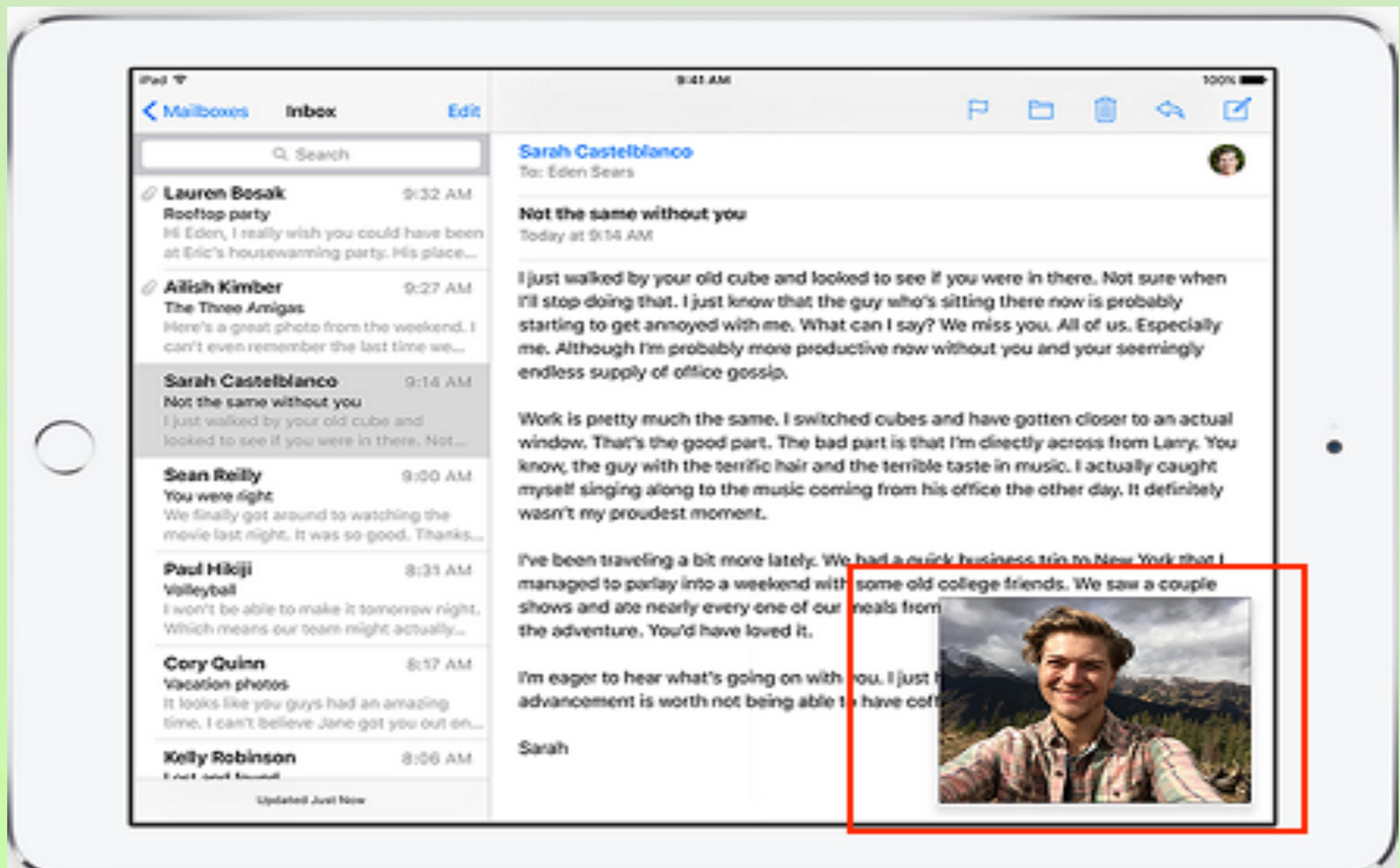
SplitView

iPad功能，分割视图模式，不离开当前app的同时就能利用侧滑操作打开第二个app，能同时操作两个 app。



Picture in Picture

iPad功能，在其他app里依然可以进行视频播放的画中画模式。用户在观看视频时如果想要看其他app，可以直接切换，而视频此时会以小窗口的形式浮动在邮件窗口上面，并且可以随意切换大小。



SlideOver、 SlideOver、 Picture in Picture业务相关

支持的设备：

- 1、 SlideOver与Picture in Picture：针对iOS系统在9.0及以后版本。硬件设备在iPad air以及iPad mini2之后的机型上都能使用。
- 2、 SlideOver：针对iOS系统在9.0及以后版本。但只有性能最强大的硬件设备 iPad Air 2代和之后的机型才支持分割视图模式。

优点：

- 1、能快速切换程序的功能，减少用户的操作成本
- 2、用户能同时使用2个软件，而且各自的功能不会有任何限制，iPad的大屏幕优势得以体现
- 3、其中Picture in Picture间接的有利咱们的app

业务相关：

能提高app的曝光率

交互方面

UI方面多样化

增强app与app之间互动性

SlideOver、 SlideOver技术相关

Size Classes:

iOS8提出，适应设备不同尺寸的新理念，不再根据设备屏幕的具体尺寸来进行区分，变成遵循尺寸的视觉感官来进行适配，将其分为普通(Regular) 和紧密(Compact) 两个种类(class)。开发者便可以无视具体的尺寸，而是对这两类和它们的组合进行适配。

LaunchScreen.storyboard:

一种新的闪屏，xib的升级，不再只是一张图片的效果，可以通过Auto Layout方式对闪屏进行布局，但不是代码

Auto Layout

从IOS 6开始苹果引入来取代Autoresizing的新的布局技术，该技术有三种设置方式：视图的大小（即视图的绝对大小）、位置（视图相对于父视图或者兄弟视图的位置）、对齐方式（相对于父视图或者相对于兄弟视图）。

AVKit 播放视频的框架

AVPlayerViewController 、 AVPlayerLayer将会自动适配Picture in Picture

Size Classes

UITraitCollection:

封装了像水平和竖直方向的 Size Class 等信息，从而得知当前的 Size Class，并进一步确定界面的布局

UITraitEnvironment:

UITraitCollection 发生变化时，比如转屏，来相应其相关的方法

Auto Layout

UILayoutGuide

Auto Layout 里面的一个内部类（实现了UILayoutSupport协议的一个UIView），其主要是用于辅助布局的。

比如：topLayoutGuide，如果你的导航栏是透明的，则y轴的0就是在状态栏下面，如果不是透明的，则在导航栏下面。

Auto Layout

NSLayoutAnchor

NSLayoutConstraint的升级版。让约束声明更加清晰明，而且还通过静态类型检查以确保您的约束保证能够正常工作。与frame的区别是，frame是限定UI区域，NSLayoutAnchor是约束UI区域

NSLayoutConstraint

约束UI的一种机制，iOS6引入的技术

VFL (Visual format language) 语句

例如：“H:|-10-[view]-10-|”表示view水平方向约束在左右距离父视图为10的区域内

Auto Layout

NSLayoutConstraint

```
/// ****ios8*****
{
    //view1的宽
    [view1 addConstraint:[NSLayoutConstraint constraintWithItem:view1 attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:nil attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:100]];

    //view1的高
    [view1 addConstraint:[NSLayoutConstraint constraintWithItem:view1 attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:nil attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:100]];

    //view1左边距离父视图左边50的距离
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view1 attribute:NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:self.view attribute:NSLayoutAttributeLeft multiplier:1 constant:50]];

    //view1等于父视图 Y值的中点
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view1 attribute:NSLayoutAttributeCenterY relatedBy:NSLayoutRelationEqual toItem:self.view attribute:NSLayoutAttributeCenterY multiplier:1 constant:0]];

    //一元约束
    //view2的宽
    //    [view2 addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeWidth
    //        relatedBy:NSLayoutRelationEqual toItem:nil attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:100]];
    //view2的高
    //    [view2 addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeHeight
    //        relatedBy:NSLayoutRelationEqual toItem:nil attribute:NSLayoutAttributeNotAnAttribute multiplier:2 constant:100]];

    //二元约束:
    //view2的宽
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:view1 attribute:NSLayoutAttributeWidth multiplier:1 constant:0]];

    //view2的高
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:view1 attribute:NSLayoutAttributeHeight multiplier:1 constant:0]];

    //view2的左边在view1右边且距离40的位置
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:view1 attribute:NSLayoutAttributeRight multiplier:1 constant:40]];

    //view2等于view1 Y值的中点
    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:view2 attribute:NSLayoutAttributeCenterY relatedBy:NSLayoutRelationEqual toItem:view1 attribute:NSLayoutAttributeCenterY multiplier:1 constant:0]];
}
```

NSLayoutAnchor

```
/// ****ios9*****
{
    //设置view1的宽
    [view1 addConstraint:[view1.widthAnchor constraintEqualToAnchor:nil multiplier:1 constant:100]];

    //设置view1的高
    [view1 addConstraint:[view1.heightAnchor constraintEqualToAnchor:nil multiplier:1 constant:100]];

    //view1左边距离父视图左边50的距离
    [self.view addConstraint:[view1.leftAnchor constraintEqualToAnchor:self.view.leftAnchor constant:50]];

    //view1等于父视图 Y值的中点
    [self.view addConstraint:[view1.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor constant:0]];

    //一元约束
    //view2的宽
    //    [view2 addConstraint:[view2.widthAnchor constraintEqualToAnchor:nil multiplier:1 constant:100]];
    //view2的高
    //    [view2 addConstraint:[view2.heightAnchor constraintEqualToAnchor:nil multiplier:2 constant:100]];

    //二元约束:
    //设置view2的宽
    [self.view addConstraint:[view2.widthAnchor constraintEqualToAnchor:view1.widthAnchor multiplier:1 constant:0]];

    //设置view2的高
    [self.view addConstraint:[view2.heightAnchor constraintEqualToAnchor:view1.heightAnchor multiplier:1 constant:0]];

    //view2的左边在view1右边且距离40的位置
    [self.view addConstraint:[view2.leftAnchor constraintEqualToAnchor:view1.rightAnchor constant:40]];

    //view2等于view1 Y值的中点
    [self.view addConstraint:[view2.centerYAnchor constraintEqualToAnchor:view1.centerYAnchor constant:0]];
}
```

Auto Layout

业务方面

- 1、快速的UI布局，提高效率
- 2、尤其VFL，建议看情况混合使用VFL代码模式，熟练后能够很大的提高开发效率

Swift 2

Swift 在Xcode7的改善和进步：

- 1、Cocoa API 中的 error 全数替换成throw，新增的 ErrorType 也很好地将错误描述进行了统一。
- 2、guard、defer控制流关键字,让 Swift 的书写更加简化。
- 3、加入了 available 块，以前我们需要自己去记忆 API 的可用性，为了解决在运行时的不同 SDK 的可用性的问题，并通过检查系统版本并进行对比来做这件事情。available会检查出可能出现版本不匹配的 API 调用，app 开发的安全性得到了进一步的保障。
- 4、为了让整个 SDK 更适合 Swift 的语法规习惯，Apple 终于在 Objective-C 中引入了泛型。这看似是 Objective-C 的加强，但是实际上却实实在在地是为 Swift 一统 Apple 开发开路。有了 Objective-C 泛型以后，用 Swift 访问 Cocoa API 基本不会再得到AnyObject类型了，这使得 Swift 的安全特性又上了一层台阶。

Swift 2

业务相关：

- 1、首先Swift是苹果重点推广语言之一，对各业务的关联影响比较长远
- 2、对后续研究swift速度起的积极的作用
- 3、异常机制减少错误查找所需要的时间，让swift开发的app更加安全

3D Touch

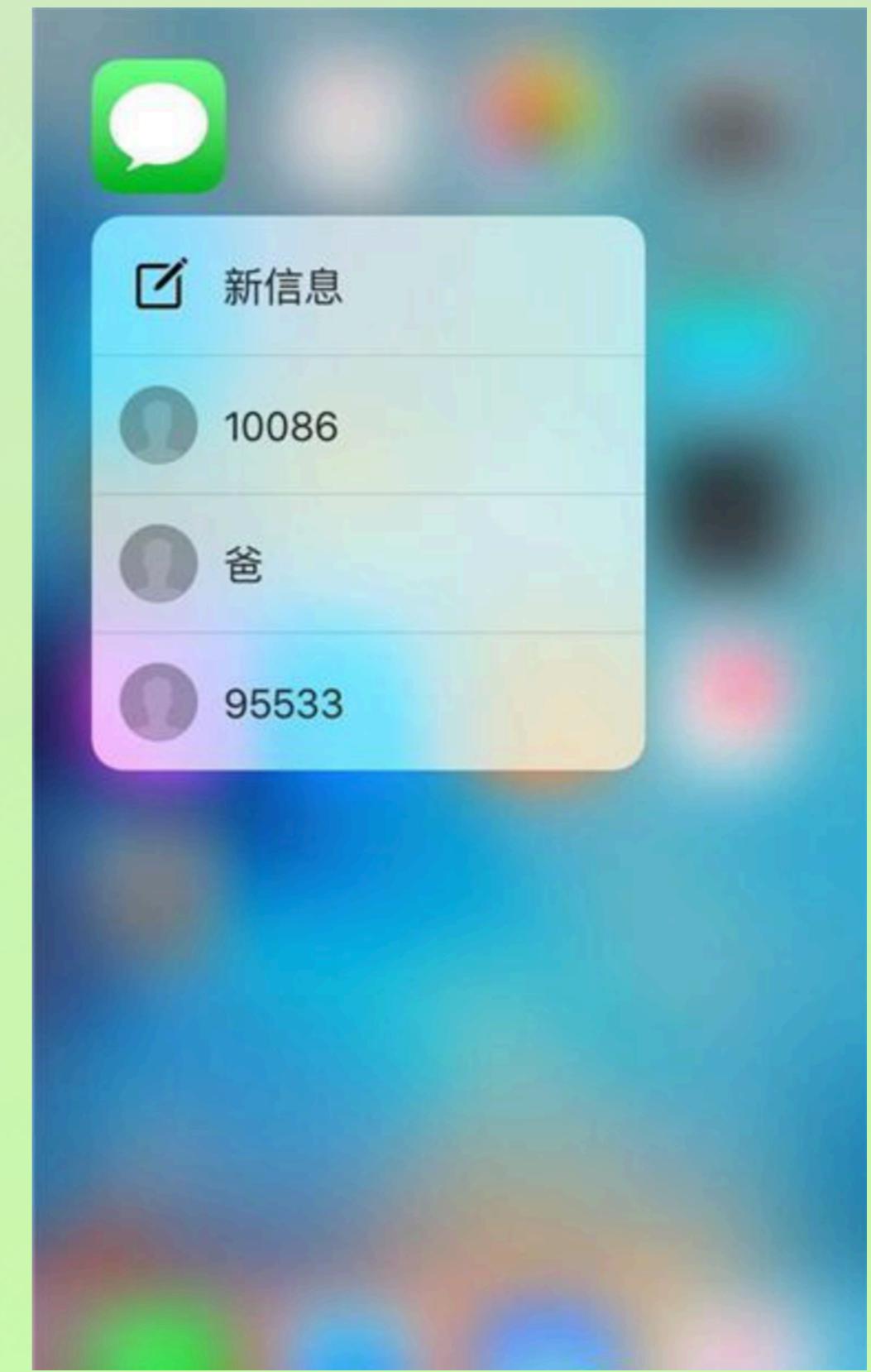
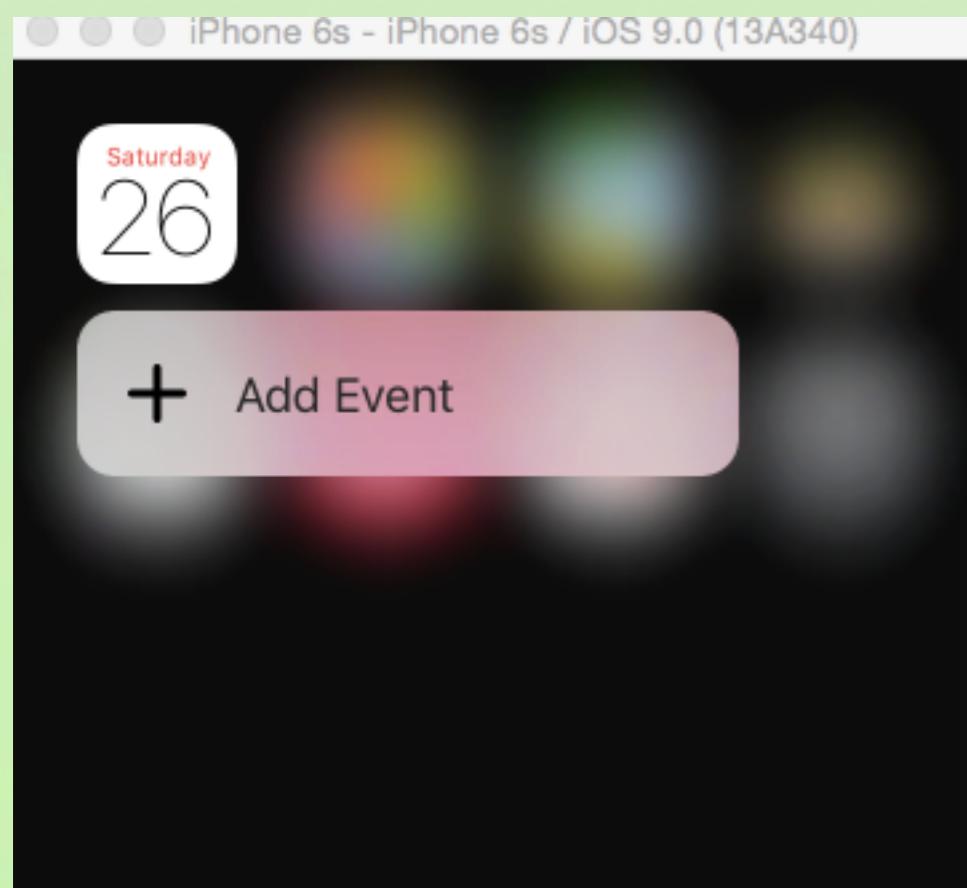
3D Touch

简单来说，就是在Touch的基础上添加了垂直维度的Touch事件。其原理上是增加了一个压力的感触，通过区分轻按和重按来进行不同的用户交互。

使用场景非常多

3D Touch

通过主屏幕的app图标，人们可以用3D Touch唤出一个菜单，快速的定位app内的相关功能



3D Touch

app内，人们可以使用这种按压力度与app更好的交互



3D Touch

业务相关：

交互方面多了一种操作方式

节省用户的访问时间

技术相关：

- 1、安装SBShortcutMenuSimulator能在模拟器模拟3D Touch效果
- 2、UIApplicationShortcutItem 创建3DTouch标签的类
- 3、UIMutableUIApplicationShortcutItem 创建可变的3DTouch标签的类
- 4、UIApplicationShortcutIcon 创建标签中图片Icon的类
- 5、ForceTouch 创建3DTouch力度方面的类

spotlightSearch

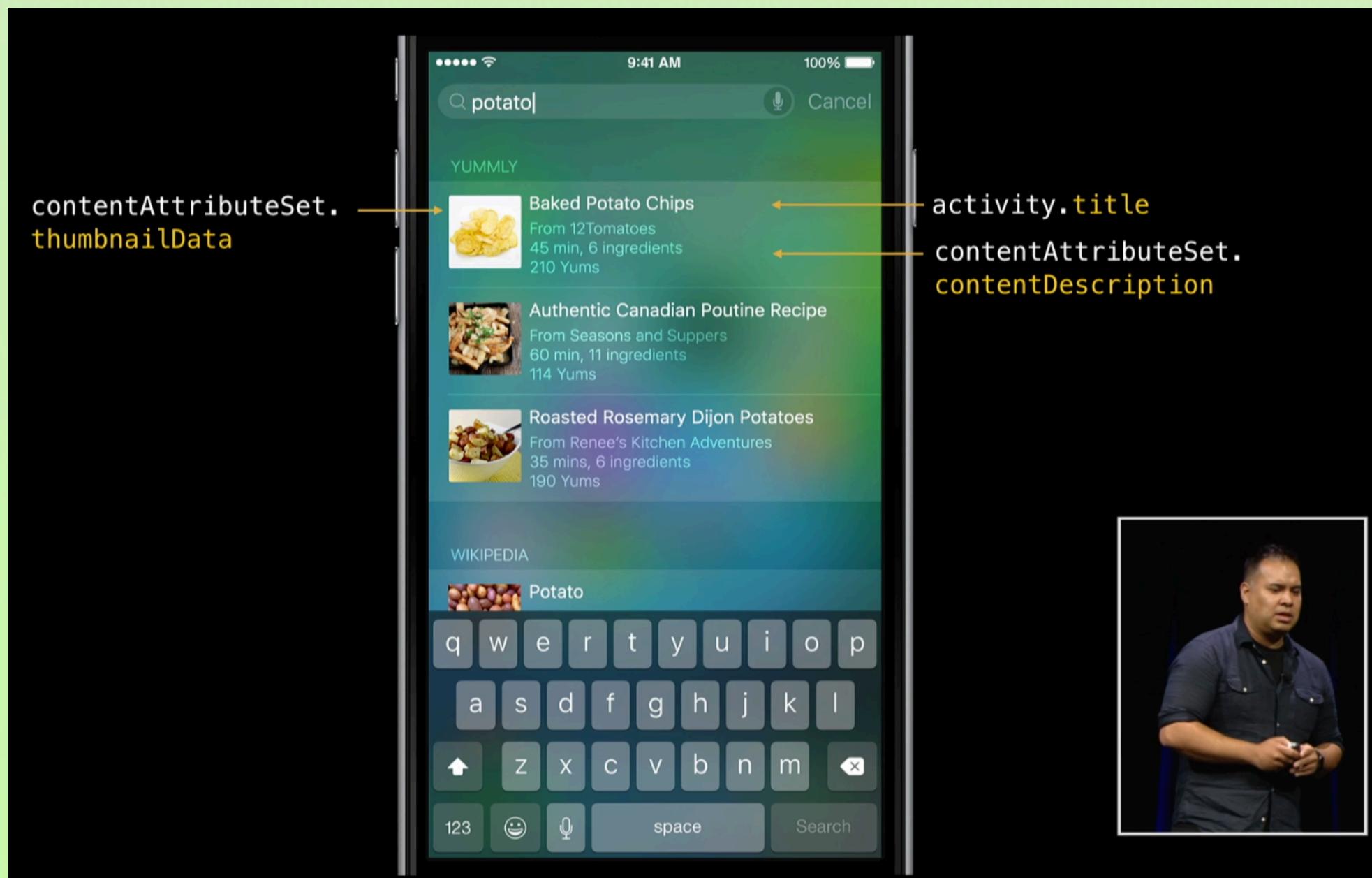
简单的理解为：手机内的搜索功能

按着屏幕中间往下拉那个搜索框

spotlight只是开放了部分api，可以搜索app及其中的内容。其实大家更想要的是Siri开放API。

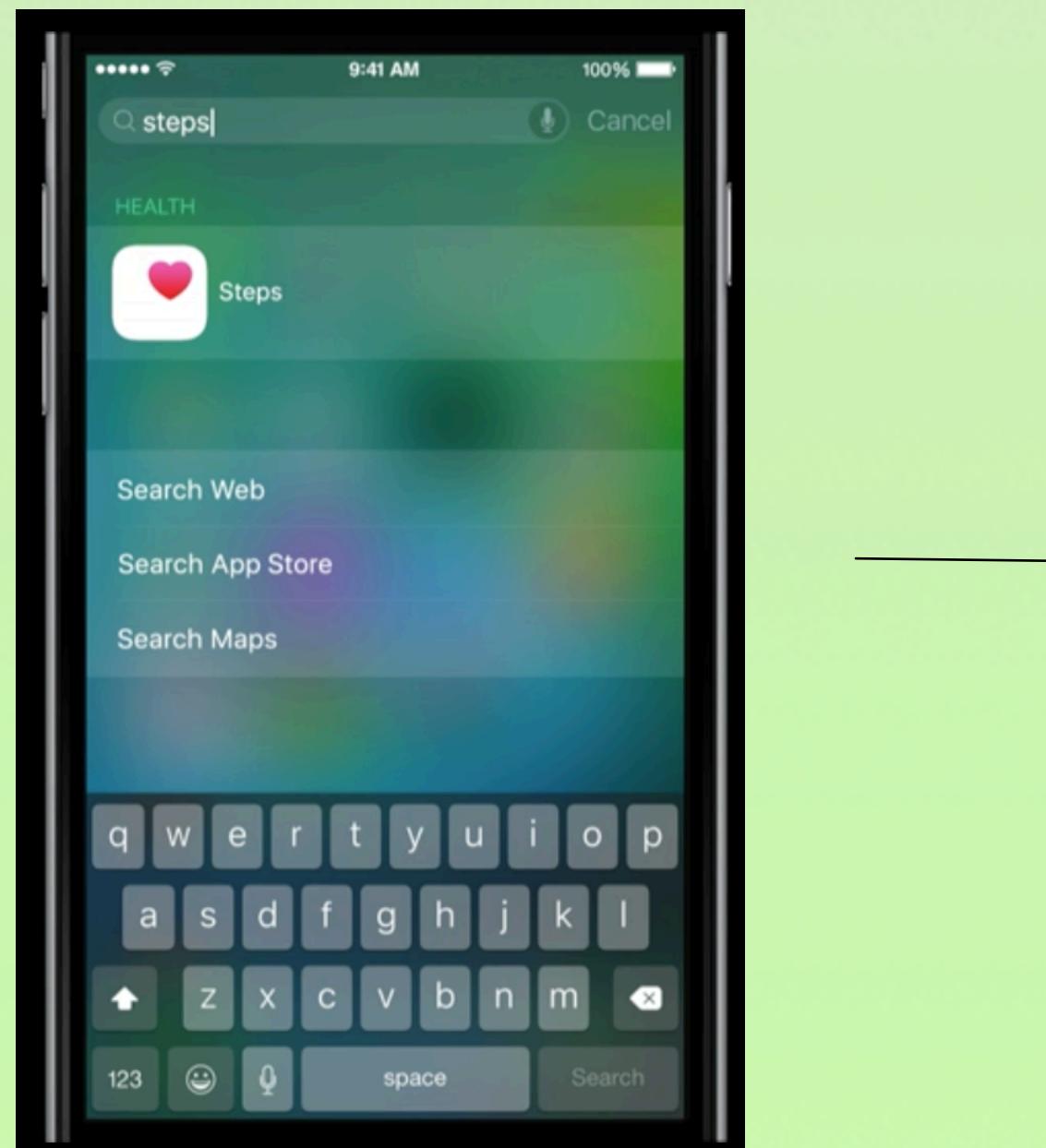
spotlightSearch

1、搜索手机当中安装的app都有哪些，向用户展示app的名称，让用户探索app中的内容



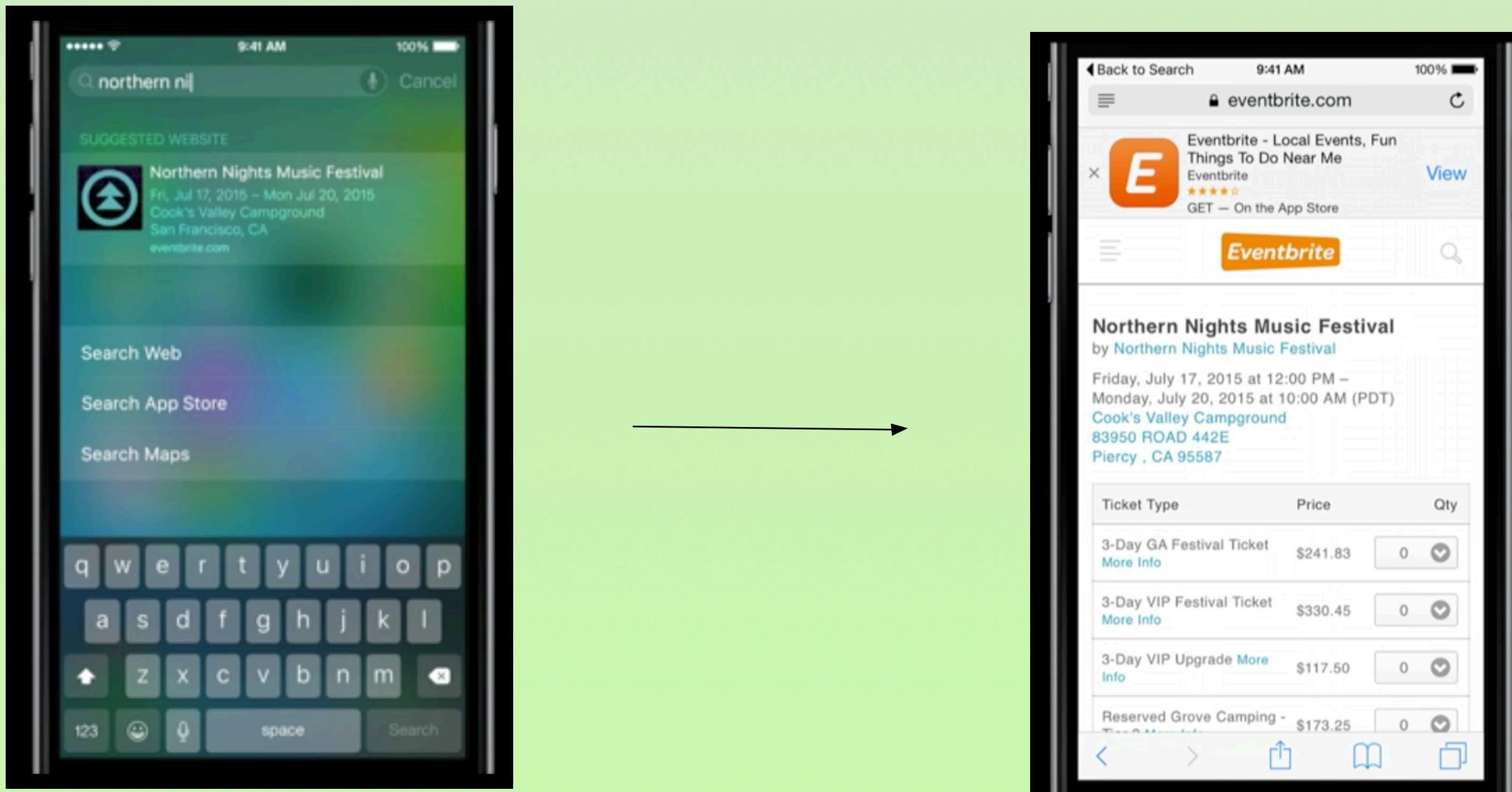
spotlightSearch

2、app里的内容能通过Spotlight搜索功能被发现和使用。直接跳到APP相对应的位置。



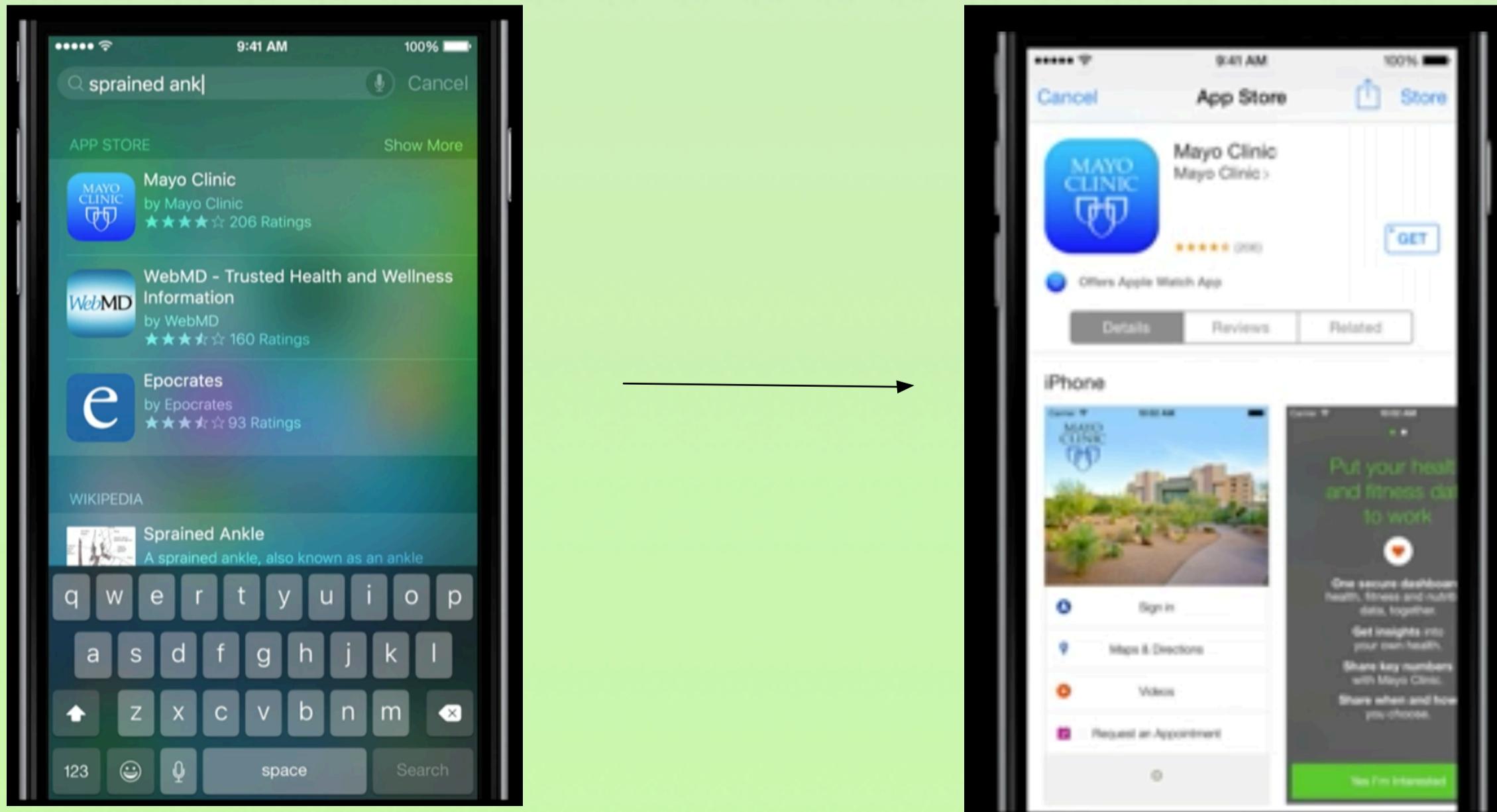
spotlightSearch

3、用户可以通过Safari看见你的app网页



spotlightSearch

4、没有安装app的用户，使用 spotlight 的时候，您的应用同样也会出现在建议列表当中。



spotlightSearch

业务相关：

- 1、让 app 多了一个可能的入口，提高打开的机会。
- 2、有效帮助应用展示和推销自己，所谓的免费营销。

注意：

搜索力度需要把控，因为用户可以在“设置-通用-Spotlight搜索”里将不想被找到的app关闭。

spotlightSearch

技术相关：

1、 NSUserActivity

包括新方法和属性,可以帮助您指项目等用户执行的 活动在你的应用程序访问一个导航点或创建和查看内容。几乎所有的应用程序可以利用 NSUserActivity api 来为用户提供有用的内容

2、 Core Spotlight框架

帮助您设备内置 app-specific 内容添加到索引和启用深度链接到您的应用程序, 类似数据库的设计并且能够给你提供更多的关于你想被搜索到的内容的信息

3、 Web markupWeb

标记允许您使您的相关网页内容搜索和帮助你丰富用户的搜索体验

MapKit

iOS9对地图做了不少优化

- 1、自定义标注、
- 2、自定义地图地图上显示交通、比例和指南针
- 3、找到与特定坐标相关的时区
- 4、支持Flyover（城市观光）



MapKit

业务方面：

LBS还是有一定帮助的

Contacts.framework

Contacts.framework 和 ContactsUI.framework 这2个新的面向对象的库取代了之前的Address Book 与 Address Book UI frameworks。如果你有从不同的数据源来的相同联系人数据，他们会自动合并，无需手动进行合并的操作。

业务方面：

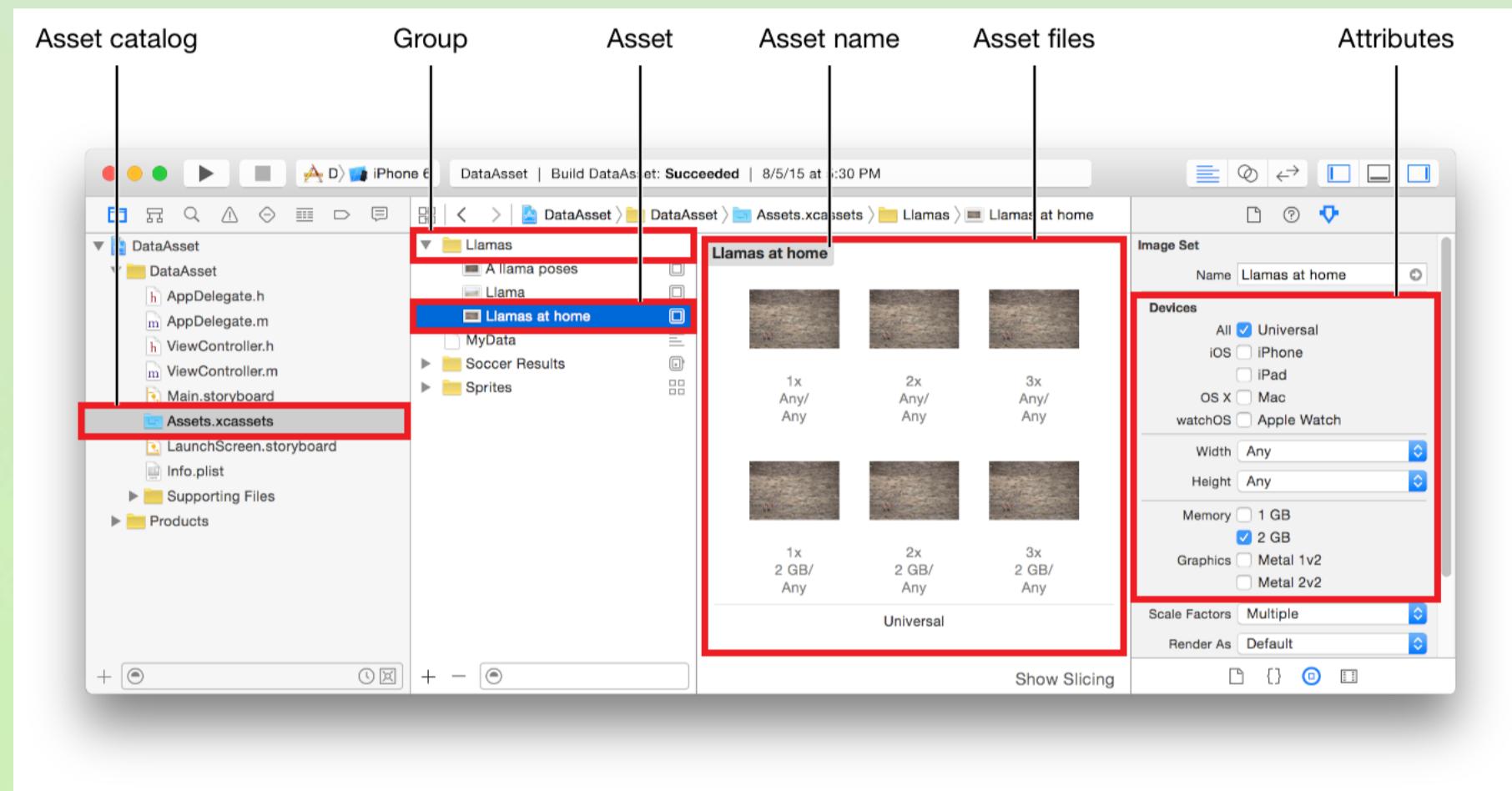
ContactsUI framework 选取联系人变简单和易用了
需要获取联系人的相关业务得关注

Thinning

瘦包，给包减肥。针对设备对apps做特定优化，以减少安装包大小，苹果表示 iOS 9的全新架构让应用程序在更新时，下载的数据包比那些采用旧系统版本的设备要小很多

Thinning

Asset Catalogs把资源放到一个叫做Assets.car这个文件里

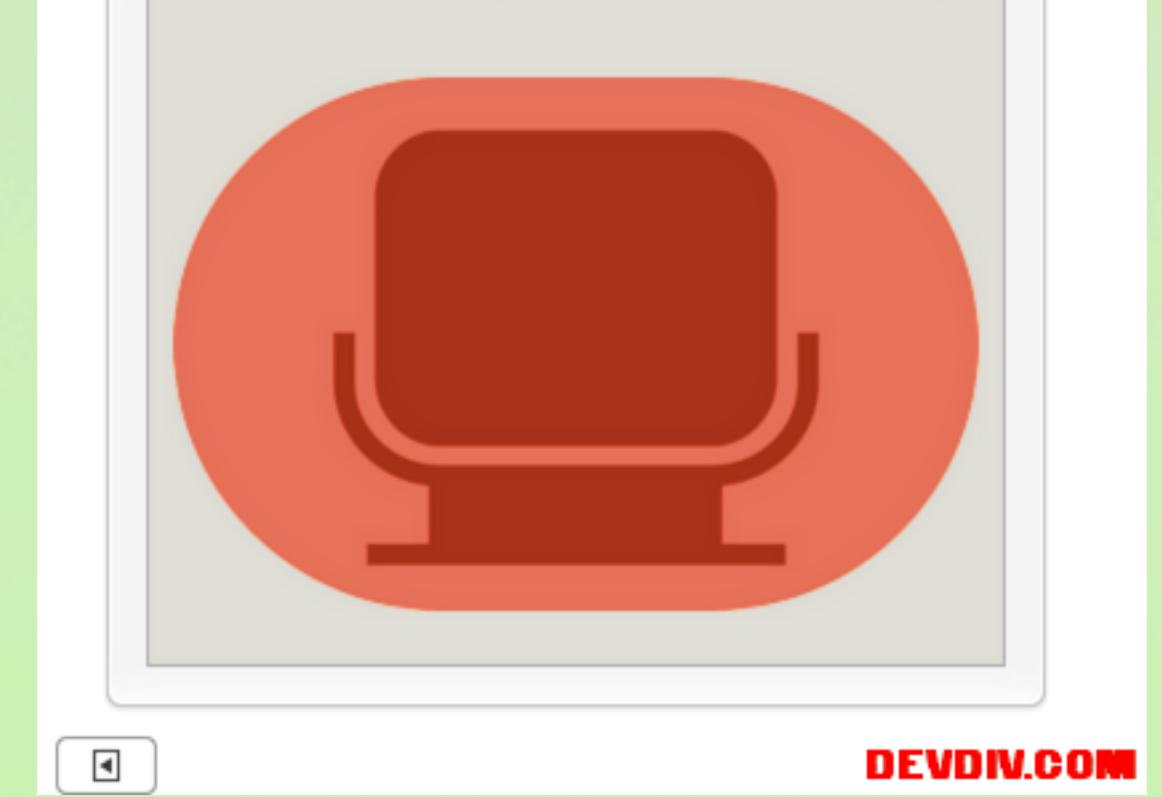
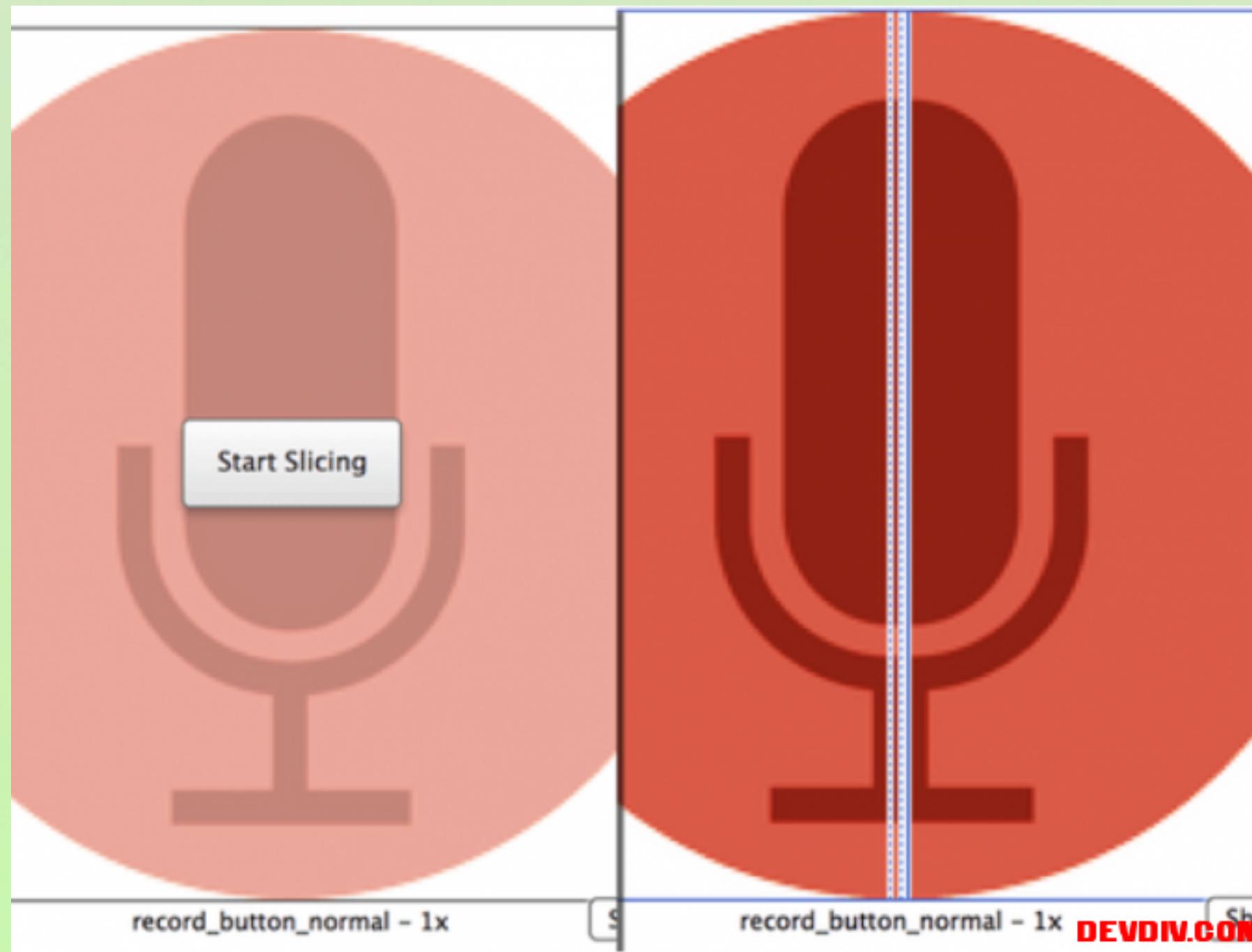


1、自定义 imagesets



Thinning

2、Image Slicing



DEVDIV.COM

Thinning

3、Bitcode

被编译程序的一种中间形式的代码，支持bitcode配置的程序将会在App store上被编译和链接。苹果在后期重新优化我们程序的二进制文件，而不需要我们重新提交一个新的版本到App store上。

4、On-demand

启用按需加载资源，部分资源放在云端或者AppStore，需要的时候才会请求下来，又是为了缩减App包体积。关于懒加载资源的好处，后面再列。大型游戏里这是很常见的优化方法

Thinning

优点：

- 1、减少应用下载的时间
- 2、为那些设备存储空间有限的用户带来了巨大的福音
- 3、安装速度更快

业务相关：

自身软件方面有很大的优势

NSDataAsset

快速获取Assets.xcassets里面的data。苹果公司建议我们把数据方面文件能放到Assets.xcassets里，方便开发员的使用，加速包的安装

业务相关
测试人员值得关注

字体

iOS9系统增加了31 种新字体， 默认字体为：细体“.SFUIText-Regular”、粗体“SFUIText-Semibold”，有点类似于一种word字体“幼圆”，字体有轻微的加粗效果，并且最关键的是字体间隙变大了！
iOS8中，字体是“Helvetica Neue”，中文的字体有点类似于“华文细黑”，渲染后，所以看上去可能比普通的华文细黑要美观。

业务方面：

给用户界面比较好的视觉效果，字体是其中一个因素

kCIAttributeFilterAvailable_iOS

对image滤镜，新增了41种滤镜

业务方面：

对图片的美化方面添加一些效果

AVSpeechSynthesisVoice

语音播报文字内容

业务方面：

输入语音输出文字，对我们搜索等是有帮助的

Content Blocker

iOS9对Safari内容拦截插件,目标就是将广告项拦截掉

业务方面:

防止干扰用户在Safari体验我们的业务

SafariServices

可以通过编程的方式将URL添加到用户的Safari阅读列表中
用户不需要跳出app外去打开Safari

业务相关

交互方面节省了用户的操作时间

UITest

Apple 给出了一个更加诱人的选项，那就是 Xcode 自带的 XCUI Test 的一系列工具。和大部分已有的 UI 测试工具类似，XCUI 使用 Accessibility 标记来确定 view，但因为是 Apple 自家的东西，它可以自动记录你的操作流程，所以你只需要书写最后的验证部分就可以了，比其他的 UI 测试工具方便很多。

业务相关
测试人员值得关注

UIFieldBehavior

behaviours都有一些属性可以用来设置不同的效果，并且可以简单的添加和使用。它有几个视图（一个椭圆和一个正方形）添加了一些碰撞逻辑和一些噪音的UIFieldBehavior

业务相关
动画方面有一定的作用

UIPress

实际物理按键监听的事件。比如：遥控器、手柄等

业务相关
多了一种交互体验

UIRegion

动力学形状的区域，如磁力或重力。

业务相关
多了一种交互体验

UIFocus

优化咱们动画多一个类，非常重要。能让app的动画效果更加生动

业务相关
动画的效果

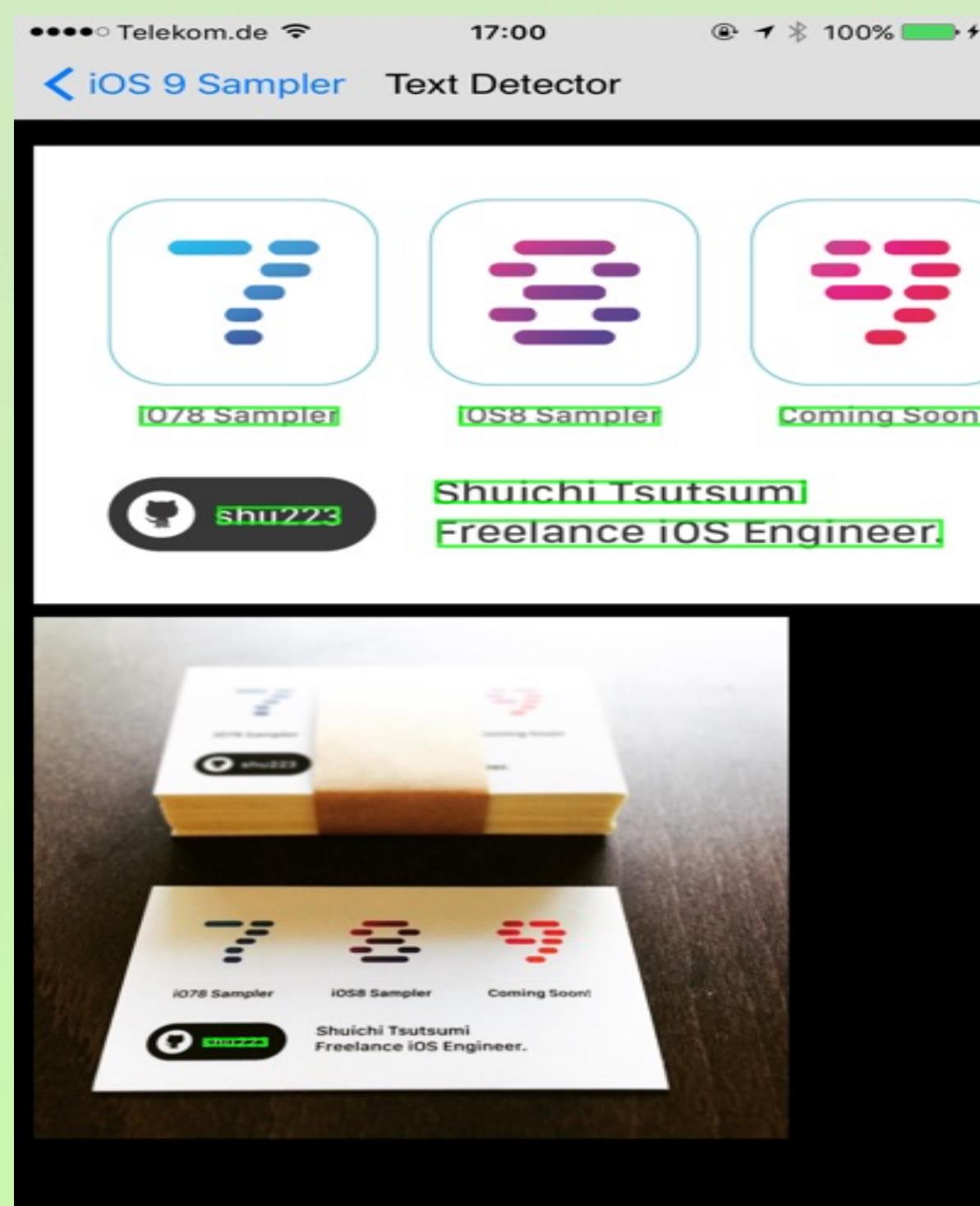
CASpringAnimation

它继承于CABaseAnimation，用于制作弹簧动画，先演示个例子：

业务相关
动画的效果

CIDetectorTypeText

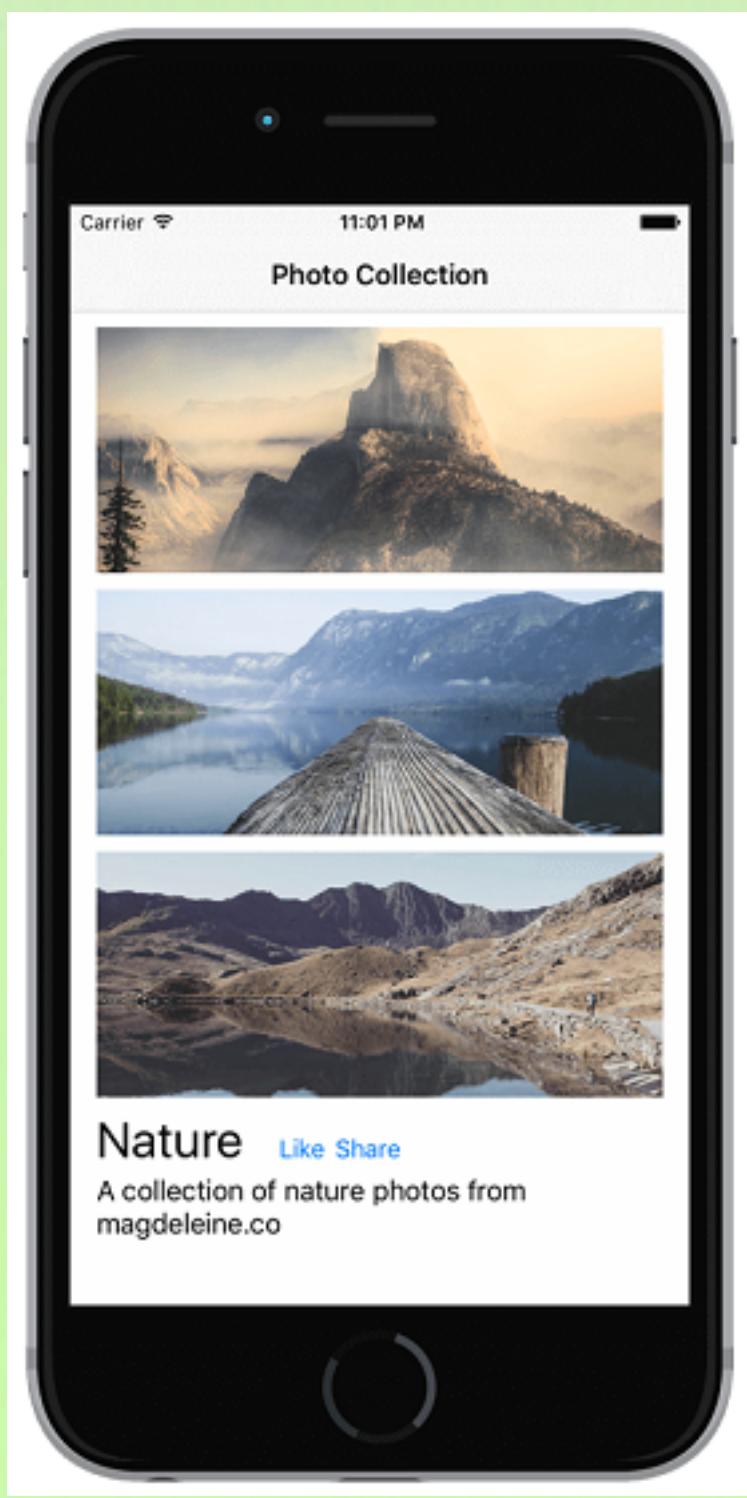
iOS9给识别器多了一种识别：文字识别



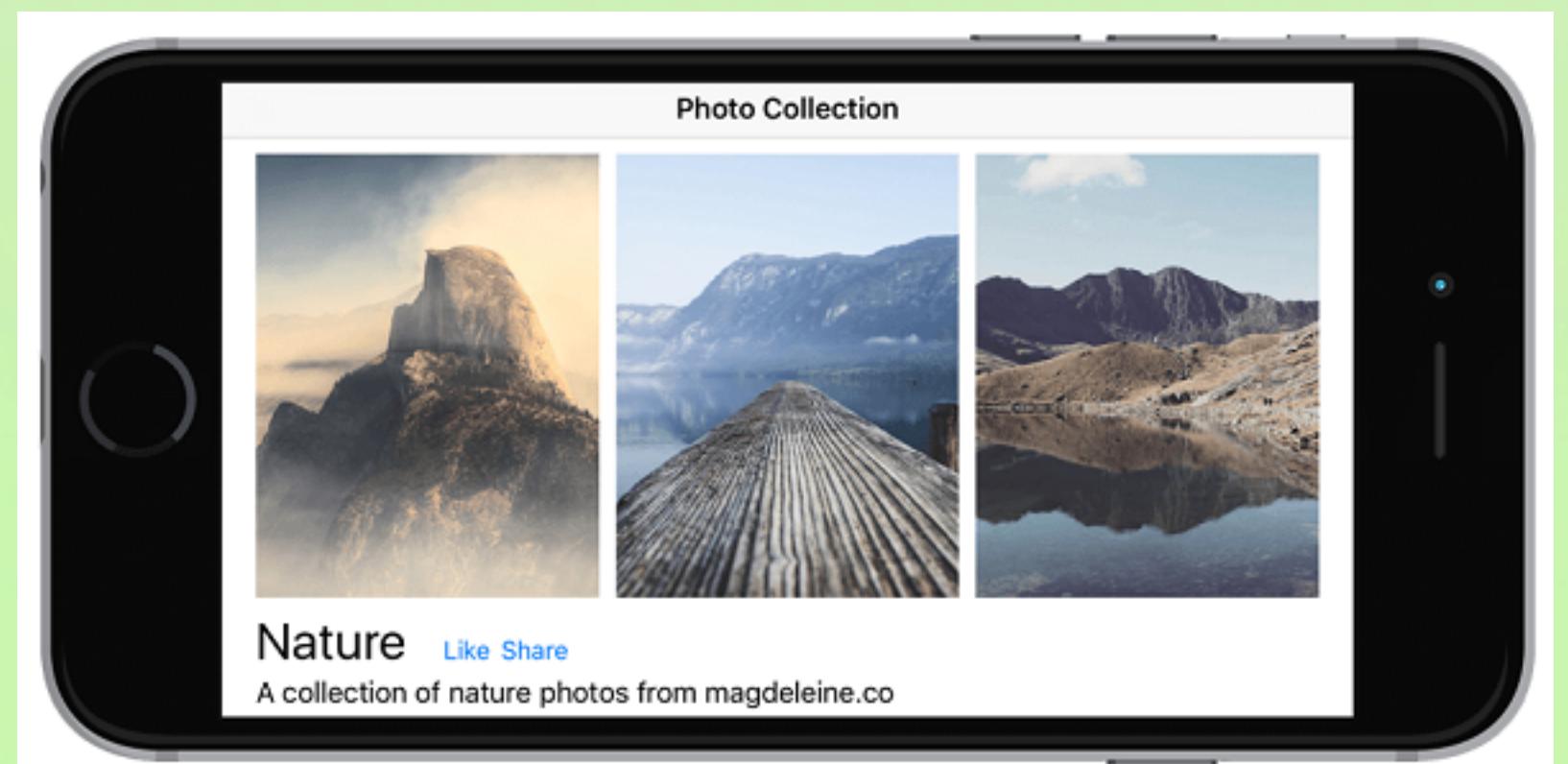
业务相关
对扫描文字方面非常有用

UIStackView

一种堆栈视图，在布局和自适应方面给开发者带来便利，对于嵌入到StackView的视图，你不用再添加自动布局的约束了。Stack View管理这些子视图的布局，并帮你自动布局约束



横屏后自动调整的样子



ARC与MRC

ARC(Automatic Reference Counting, 自动引用计数), iOS 5推出的, 值得旧事重提。但需要注意, 使用ARC后并不是就没有内存泄漏了, 只是“人工”引用计数到“自动”引用技术的演变, 并不是“智能”的。

相关业务:

ARC能增加开发者效率, 对各个业务有着非常积极的作用。