会写代码的猪

没谱青年，资深80后，大爱加菲猫，不炫技会死！

首页 | About | Contact

iOS应用2.0版怎么做

作者：gaosboy | 时间：March 16, 2013 | 分类：猪在写代码

首发于InfoQ中文站

移动互联网如火如荼，iOS 应用+ Android 应用+ 手机站似乎成了所有互联网公司的标配，你的网站要是还没有个iOS 应用，似乎都不好意思跟人打招呼。

iOS 应用诞生毕竟才只有不到5年的时间，各个方面还都处在起步阶段。不管是出于团队缺乏经验，还是那个“唯快不破”的铁律，往往这些iOS 应用的第一个版本都比较简陋，尤其在技术架构上。这篇文章，我想跟大家探讨的就是这个问题，当你的iOS应用已经有一个版本在线的情况下，如何去构建一款可以支持高效迭代，快速响应的2.0版。 首先，应用的架构要层次清晰。把不依赖版本更新变化的，以来版本更新变化的，不需要变化的，业务逻辑，视图逻辑，工具，等这些内容梳理清楚，单独处理。

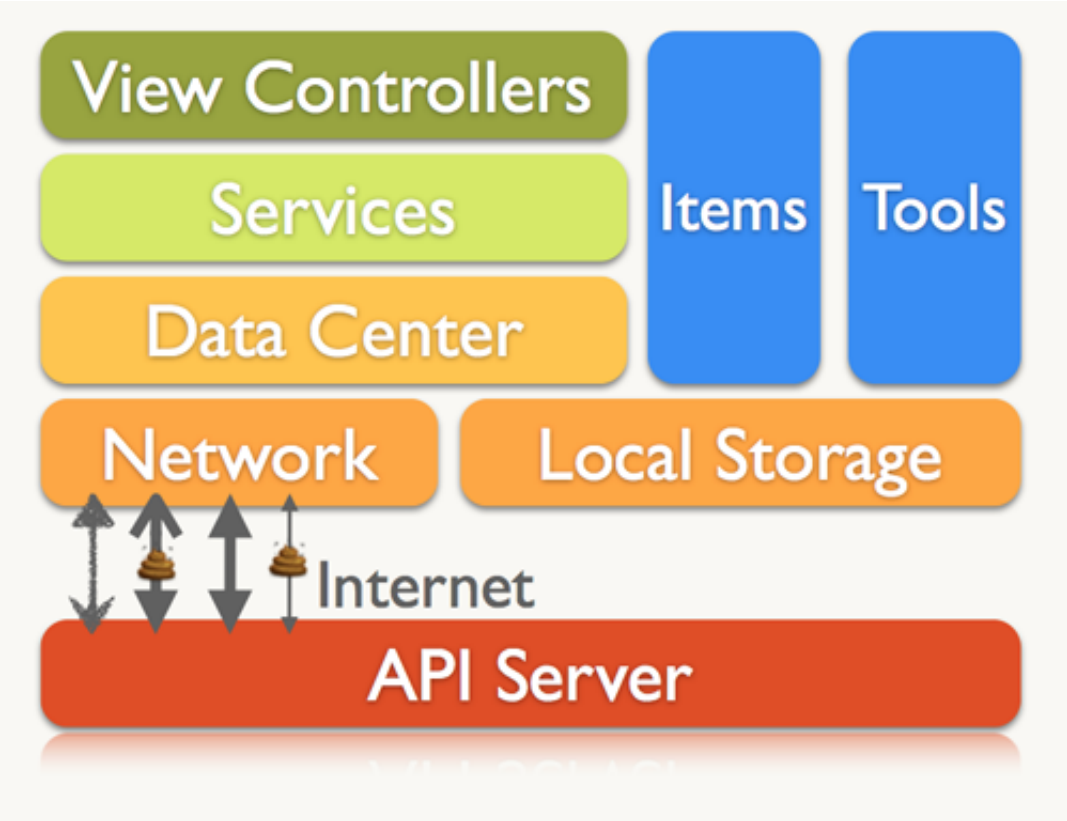

图1，应用整体架构

自下而上，最底层的是API Server ，也就是服务端，和经常踩到大便一般的网络，这篇文章着重讨论客户端架构，对这两层不多说。

Network 和Local Storage 是整个应用的数据来源。Local Storage 除了存储资源数据，还缓存来自Network 的数据。在这一层中，所有数据都以原始的未经格式化的结构存在，或是文本，或是文本的数组和字典。

Network 是客户端与服务端的沟通桥梁，一般建议使用开源控件（如：AFNetworking）实现。在大便一样的网络

情况下，请求会出现各种异常，因此这些控件中提供的请求控制方法会提供很多便利，去处理这些问题。

Local Storage 除以文件形式直接存储图片等资源文件外，信息类数据，如：用户信息，配置信息等，建议使用Cocoa 框架中提供的Core Data，对SQLite 的一个非常好的封装。直接使用文件存储要注意写入频率，高频率的I/O操作非常占用资源，一般来说把数据直接放在内存中，只有需要的时候，如：关闭应用或关键数据写入（用户密码）才进行回写。

Items 是整个应用中对数据对象的封装，某种程度上就是Value Object 的集合。 一个数据类型的Item 可以是最原始的数组或字典，也可以是封装成如Java 中的POJO 一般，或者更复杂的带有基本处理方法的数据对象。 Data Center 是Items 的控制中心，把网络请求，网络异常处理，缓存机制，等完全封装起来。Data Center 响应上层的数据请求，向Network 或Local Storage 索取原始数据，并拼装成Items 中定义的数据结构，返回给上层。

Services 是对业务逻辑的封装，Data Center 关注如何获取Items，Services 则关注如何组织Items，如：怎么构建搜索参数，如何翻页，列表排序等。Services 把复杂的业务逻辑封装成类名，方法名和参数，供上层调用。这一层的变化较上面提到的几层要频繁的多，因为Services 随着业务变化而变化，已经不是纯技术层面了。

View Controllers 顾名思义，就是Cocoa 框架中的ViewController，在这些ViewController 里只有展示逻辑。这一层并不关心返回的Items 是什么含义，为什么要如此组合，View Controllers 只关心什么Items 需要什么视图，采用是什么颜色，等。

Tools 是几乎贯穿整个应用的工具和扩展集合，工具如：字符串md5 加密，数组扁平化，等；扩展如：UIView 框架设置，URL 拼装，等。

在这个架构下，Items 和Tools 几乎是静态的，没有逻辑。Network、Data Center和Local Storage 从技术角度封装逻辑，在版本迭代过程中几乎不发生变化。Services 这一层是版本迭代中调整的重点，他的厚度取决于这款应用的业务复杂度。View Controllers 的内容改变更加频繁，甚至不依赖版本更新，需要通过API Server 的的信息修改界面展示，需要较高的可定制性。

在如上描述的框架下构建一个iOS 应用可以让整个工程的代码层次清晰起来，然而一个高效的互联网应用，仅有层次清晰还远远不够，还需要更多其他方法降低应用的开发和运维成本。 首先，恰当使用WebView 可以大大提升应用的灵活性，降低开发成本。使用WebView 要与创建Web 应用区别开，创建Web 应用是整个应用的视觉控件全部基于WebView 创建，少量的本地代码辅助交互；而在这里说的是在一个应用中，在不影响视觉和交互体验的前提下，使用WebView 完成一部分视觉元素。

在应用中使用WebView 分为两种，一种是整张视图使用WebView 创建，这种方式适用于非关键界面，为了降低成本，或是处在Beta 阶段的界面，仍然有许多细节需要调整。 这种使用方式WebView 的复杂度较高，需要在WebView 中实现Cookie 记录，HttpRequest 自定义，与本地代码交互等操作。

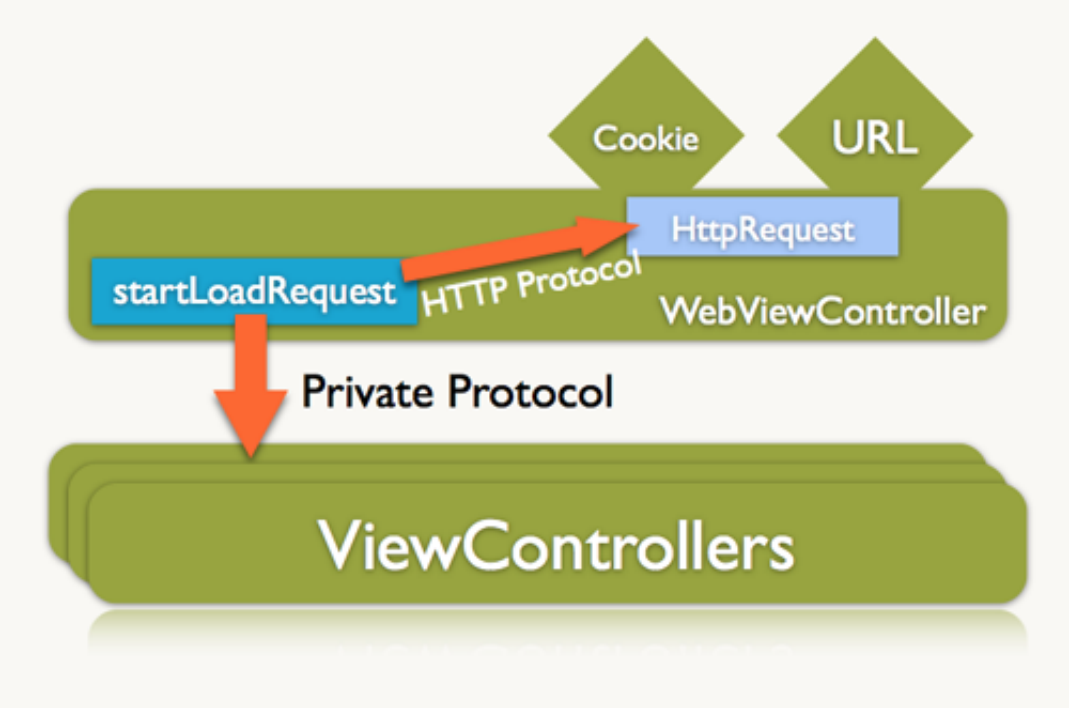

图2，复杂的WebView

例如，微信的FAQ 页面，这是一个非关键页面，但内容变化有比较频繁，所以使用WebView 来实现。微信的这个WebView 界面并未做任何伪装，就是以一个原生WebView 的样式示人，在实际应用中，如果对视觉要求较高，可以通过修改NavigationBar、ToolBar和WebView 中的一些特有手势进行伪装，使其看起来更像一个本地视图。

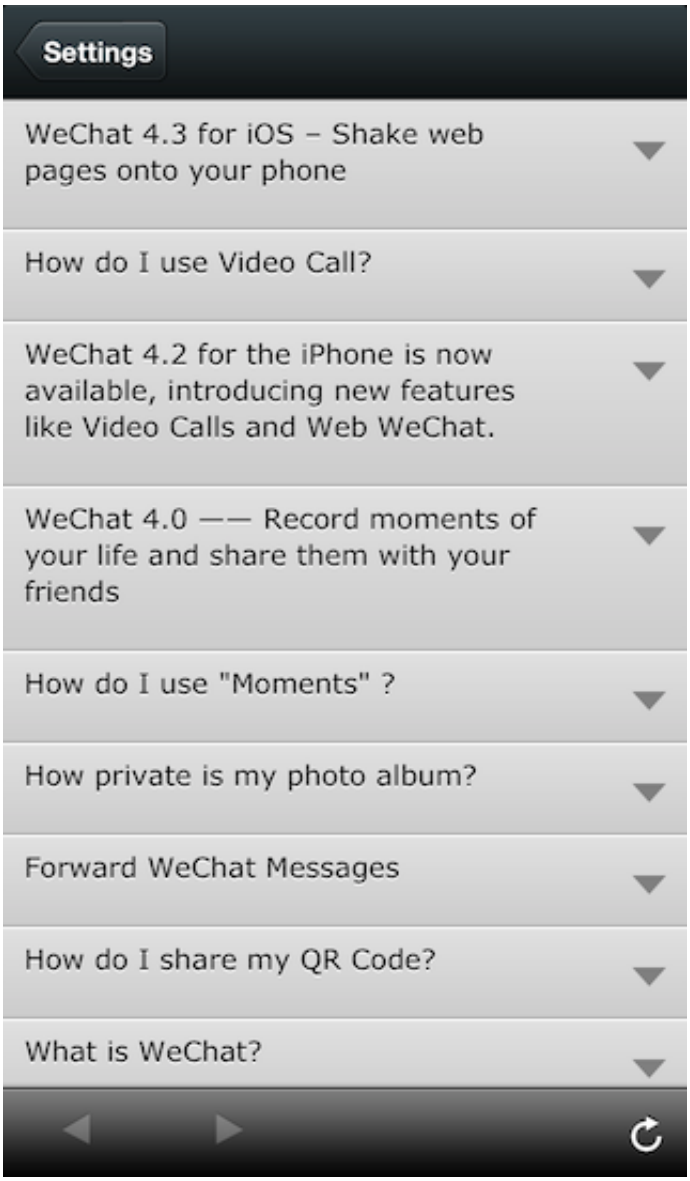

图3，微信FAQ

另一种是WebView 嵌入本地代码中，适合展示较复杂的视觉元素，既可以由WebView 直接从网络加载内容，也可单独加载HTML 代码，以本地形式渲染。这种WebView 相较上一种使用方式简单许多，只需要实现与本地代码交互的部分即可。

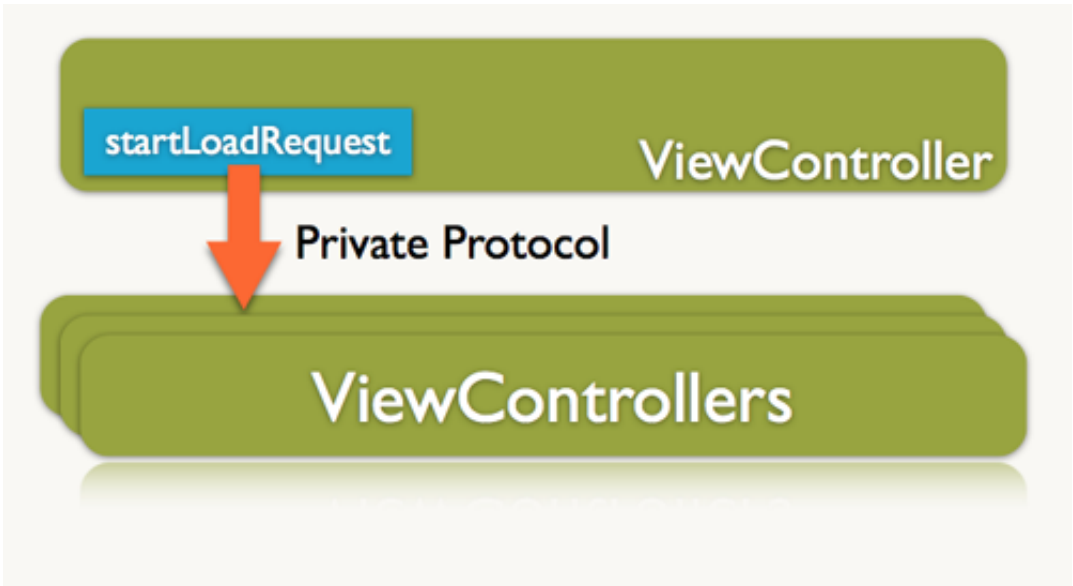

图4，简单的WebView

新浪微博的微博详情页就是以这种方式实现，微博正文和来源使用WebView实现，其他部分是本地代码。由于微博正文内的表情较多，因此图文混排的情况较复杂，所以使用局部WebView 实现是一个非常好的方法。

图5，微博详情页

在WebView 之外，大量的交互和视觉是由本地视觉控制器（ViewControllers ）管理的，传统的iOS 应用由UINavigationController 中的堆栈储存ViewControllers 对象，通过push 和pop 方法直接管理。这种方式管理的ViewControllers 耦合性非常强，之间存在依赖，不能单独存在。而强耦合在应对变化中总会非常麻烦，因此需要对这种强耦合的方式进行一些修改和封装，适应互联网应用的特点。

使用URL Scheme 管理ViewControllers 最初是Facebook 开发的Three20 框架实现（由于该框架体积庞大已被官方放弃，不建议使用，https://github.com/gaosboy/urlmanager 是一个轻量级的开源URL Scheme 管理控件）。URL Scheme 的特点在于 ViewController 中需要的信息全部通过参数的形式传递，通过一个URL 来标示、创建和管理ViewController。

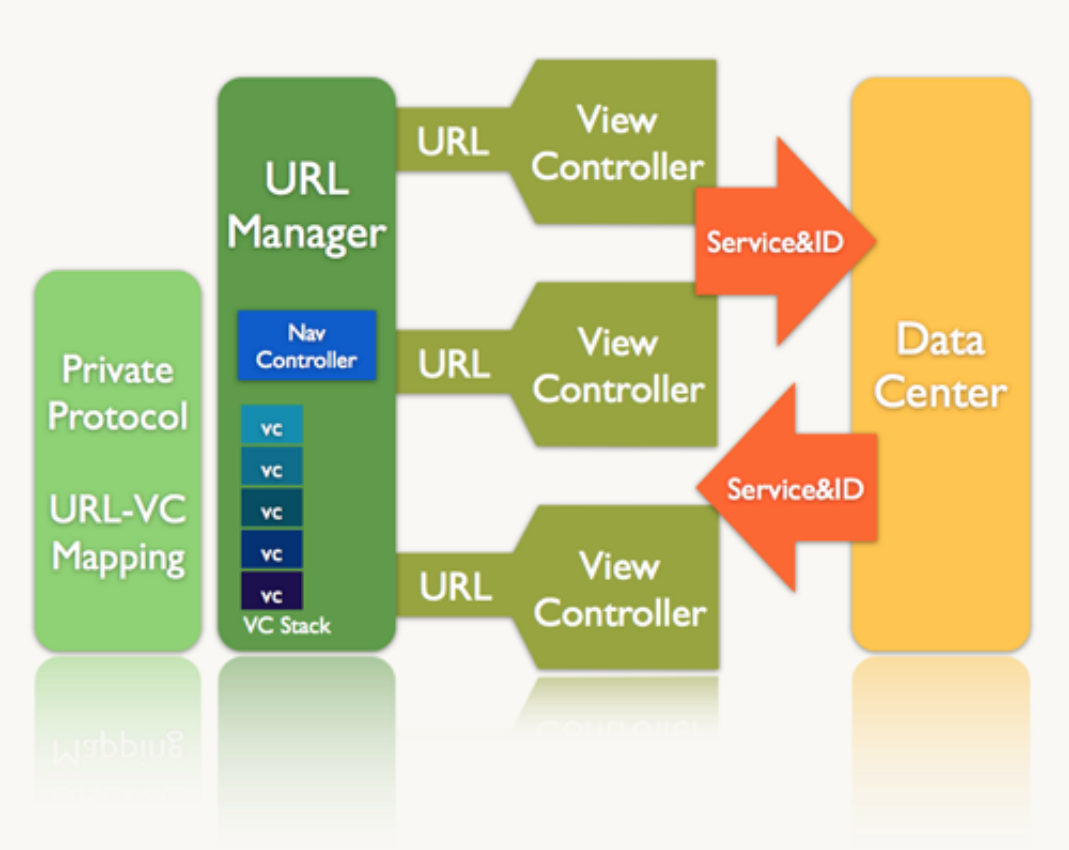

图6，URL Scheme 原理图

URL Scheme 的原理是维护一张URL - ViewController Class Name 的关系表，在这种关系表中开发者要定义一个私有协议，以区分HTTP 请求的URL，例如：sf:// 。ViewControllerA 通过需要调用ViewControllerB 时，首先要构造一个与B 相对的URL，传入URL 管理器。管理器通过查关系表找到相对的ViewController，并通过NSClassFromString 方法初始化活的对象实例。在该实例初始化过程中，要完成对其URL 的解析，从URL 的参数和路径中获取接下来试图渲染所需要的全部信息。

通过URL Scheme 管理ViewControllers 可以实现从任意ViewController 对任意ViewController 的调用，而且这些调用是完全动态的，只需要构造一个URL 加入相应参数即可，实现了ViewController 之间的解耦。

SegmentFault 官方应用是对以上所述的原则和架构的一次实践，你可以通过https://github.com/gaosboy/iOSSF 获取到该项目的代码。每一款iOS 应用都有其独一无二的特点，这是由应用本身的业务特点决定的，因此一套架构体系无法完全适应，针对应用自身特点，在以上提到的原则下对架构作出合理调整，建立一个使用自身应用的架构体系。在iOS 应用开发过程中，遇到任何问题都可以到http://segmentfault.com/t/ios 进行交流，让大家帮你构建一个合理的应用。

总之，在应用从1.0 向2.0 的过渡中，最重要的一点是应用架构趋于合理，为接下来的版本迭代做好技术储备，能够应对快速变化的需求，让版本迭代的节奏跟上需求变化的速度。

标签：ios

添加新评论

称呼 *

Email *

网站

http://

内容 *

提交评论