# 当前版本存在的问题

## 类的实例化

目前实例化采用配置文件，然后在页面加载的时候用一个迭代实例化配置文件中的类，但是子App的类在点击了之后才进行实例化，会有些许卡顿现象，但若一开始就实例化，则页面初次加载的时间会很长，而且当App变多时，全部实例化会导致内存飙升，可能会使得整个页面都变得卡顿。

## 页面布局

一开始的时候完全采用-webkit-box-flex布局，但是后来由于偷懒，就直接写了高度，屏幕大小换了之后肯定会出问题，但是很多时候光用-webkit-box布局又会增加很多div，对页面结构也不友好，-webkit-box只能平铺满。重构后应该要支持完整的自适应，所有东西能够按比例缩放。

## 静态属性继承

由于Ext不支持静态属性继承，有些地方父类的静态属性就没写，有些地方却写了，有些地方后来经过修改会造成不一致。

## 暴露的接口

使用Ext的面向对象模块，比较难实现私有方法，当初决定在私有方法前面加下划线来表示私有方法，但是由于后来偶尔没有注意，导致后面已经分不清楚私有和公有方法。需要找一个在Ext的面向对象模块下的实现私有的方法，或者标识如下：private的方法使用“\_\_”，而protected的方法采用“\_”标识，变量同理，同时舍弃Ext自带静态变量管理，使用自定义静态变量管理。

## 配置文件

目前配置文件有过去信息，并且冗余量较大，比较关键的一点，每个App的配置信息并没有在公用的配置文件里面，而是标识在了各自的App的文件中，到时，修改起来可能会非常麻烦。

## Css类名命名

目前css类名命名还是很不规范，有些地方很容易引起冲突以及歧义，很多类名也有冗余，包括CSS文件，冗余量过大，表示也不够清晰。

## MVC

之前觉得小程序还是不要用MVC了，麻烦得要死，现在才发现自己错了，东西多了之后确实不好管理，若把现在的control层和view层分开，一定能省下不少事情。多用MVC，对自己以后的工作也有好处。

## 数据

目前太关注于前台展示，而过于忽略数据。主要一个原因还是在于太心急了，重构时，一定要把数据一并加进去。

## 后台

考虑要不要采用后台来传输数据，这样可以方便管理数据，但是对环境的依赖性比较强，换了一个环境之后，配置环境比较麻烦。采用后台也可以学习一下nodeJS的知识。

## 数据存储方式

之前欣喜于HTML5的localStorage，但是才发现只能存储字符串，若要存储其他数据，则需要自己设计一套管理方案，先将数据转换成字符串，然后读取的时候再转换成相应的需要的数据，但是不知道到时候需要时候的数据类型，并且，转换成字符串之后也许会引起某些问题。

## 开发语言

最近看了coffeeScript蛮不错的，代码很精简，并且支持面向对象，但是也许适应需要一个过程，CoffeeScript语法类似ruby，可能自己还是比较喜欢c/java类型语法，但是学习新知识是保持一个积极向上的心的表现，也可以让自己不那么显老。

## 模块化

目前将所有的js全部写在HTML页面，页面庞大的同时，管理上也有很多不便之处，并且由于js文件加载的先后及以来关系，花费较多时间在处理这些东西。或许可以采用SeaJS进行按需加载，不过先后关系还是不能很好地处理，也许会出现一些未知错误，还有很关键的一点，页面需要非常严格的流畅性，动态加载JS文件或许会造成卡顿的现象，这是很不好的一种情况，也许需要找找别的解决办法。

## 开发浏览器环境

浏览器环境应该依然是会选择webkit，首先是对webkit的性能欣喜，并且有V8引擎驱动js，在safari的表现上面也丝毫不差。Firefox对HTML5的支持也不错，而且还是通过插件将web应用转换成桌面应用，这是一个比较好的功能，但是无奈firefox给我一种迟钝的感觉，性能表现上也没有chrome来得优秀。

## 各个表现层的初始化

目前没有一个统一的初始化方法或者初始化样式表，改变样式后，再将其还原花费了很多精力，需要制定一个初始化方法，能够根据需要初始化全部而部分组件。

## 严格模式

HTML5新增严格模式，对于语法不能再像以前那么松散，这对于后期管理是不错的。每个js文件都应该加入严格模式，让浏览器检查，好及时发现一些疏漏，养成良好的编码习惯。

## 整体架构

之前记得顶部状态栏是占位置的，然后也并没有考虑完善，导致后来在做iphoto的时候就出问题了，iphoto并不是全屏应用，但是他能够通过拖动使得主体部分显示在顶部栏下面，而由于拖动需要，将容器设置成了overflow：hidden；之后又通过修改结构样式，将顶部栏的空间占用去掉了，这样也许就破坏了当初的设计，或许会对之后的结构造成比较大的影响。

## BUG

目前才做了这么一些东西，就发现了很多BUG，包括在工作中的问题也是一样，一味只顾着进度，却忽视了质量，其实质量才是最重要的，心急还是一个主要的问题，要好好控制自己的心态。重构时，将各模块独立化，做好单元测试，尽量减少BUG。

## Card切换

记得上次在写card切换的时候，真的是绞尽脑汁，加起来差不多快有一个星期了，终于差不多写了一个大概，还是没有考虑到比较全面的地方，而且动画效果也基本只有滑动一种，若是以后想用别的动画效果的话，还是得修改原先的代码。可以考虑专门写一个动画类，包含各种类型的动画，适用于整个系统。

还有一个比较严重的问题，就是目前card的运动和各自card的bar的运动是不同步的，稍微有些时间偏差的话，就会造成运动看上去错位，目前因为把card的bar也放在app中实例化，所以不得不采用这种方法。

## 模块化

之前没有重视这个问题，用一个一个script标签引入js文件，导致现在html页面有50多个js文件，一排的script，着实让人望而生畏，而且，由于js文件的依赖关系，也曾静花费了1 2天的时间来处理，这都是没有必要的。今天早上完整得看了一下seajs的文档，之前以为seajs是等到require的时候才加载js文件，对于iOS这种对流畅度要求非常高的地方来讲，异步加载始终是噩梦，也就不了了之，今天看完了整个文档才发现，seajs会一次性地下载所有require的js文件，也可以加个参数实行真正的按需加载，那么自己就安心了，看来seajs会是一个不错的选择，正好工作中也有使用到，模块化就是用seajs了。

## 开机界面

由于现在总是将文件在本地打开，也就不存在一个加载的问题，如果放到某些服务上的话，打开的时候就会很恶心，所以，准备在这版本里面加入一个开机界面，也就是iPhone的开机白色苹果logo，等到文件完全下载完成之后，然后再显示解锁界面，这样第一次打开的时候正好也可以模拟iphone的开机界面，又可以让文件完全被下载，而且第二次打开的时候由于有缓存的关系，可以很快被加载。

## 动画类

目前动画的实现都是直接写在代码中，或者可以考虑将所有动画封装成一个类，减少冗余量。

## 图片缓存

之前想用seajs来缓存图片，结果发现seajs的require只能加载CMD规范的module，看来还是只能用image对象的方式去解决。

## 节点缓存

之前获取的节点都会缓存在相应实例里面，若是下次获取相同元素，则先从缓冲池里面寻找，若是没有找到，则再从DOM中寻找，这样的一个好处是提高重复元素选取的速度，但是也带来了一个不好的地方，若是一个元素只被选取过一次，类似某些按钮添加单击事件，这样，这个节点DOM对象依然是被永久保存在相应实例中，对内存有一些浪费，但是在开始的时候也不能预测以后是否会再次选取该节点对象，还需要有一个折中的考虑，或者在写的时候就进行人为的控制。