

# 移动应用开发期末项目报告 – 跨平台天气预报移动应用的设计与开发

---

小组成员

概述

背景

拓展实现

缺点和不足

兼容性

项目说明

功能模块

逻辑流程

天气页

搜索页

技术说明

渲染

数据

服务端

开发

代码架构

src/assets

src/components

src/config

src/layouts

src/models

应用效果

开发过程中遇到的问题及解决办法

问题 1：左右滑动如何停留在整数页？

问题 2：如何保证卡片组的高度与数量无关？

问题 3：Icon Font 引入后乱码？

问题 4: 绝对定位的渲染层级问题?

问题 5: 应用打包安装后内容没有变化?

问题 6: 页面元素如何位移?

思考及感想

参考文献

## 小组成员

姓名	学号	邮箱	负责内容	贡献量
郑志伟	17231239	work@imhele.com	设计、组件、国际化、数据流、服务端	60%
麻锦涛	16281262	3464984961@qq.com	原生桥接、Debug、Review、打包	40%

## 概述

本小组开发的 Tiny Weather 为一款天气预报移动应用, 拥有以下功能:

- 预报最近三天内白天与夜间的天气状况
- 支持国内外共 5937 个城市与地区的天气查询
- 支持添加最多 6 个城市, 城市保存在本地, 无需每次打开重新添加
- 打开应用自动更新天气, 也可手动下拉更新天气

## 背景

一开始本小组准备开发的是一款饮食推荐应用, 来源于其中一个组员的大创项目, 希望通过本堂课程的学习从零开始开发一款移动应用, 但由于项目并没有入选互联网+, 所以重新选择参考范例中的一类写了一个。虽然现在天气应用有很多, 但是我们希望重新设计和开发一款跨平台的天气预报应用, **探索极致用户体验与最佳工程实践**。

技术	灵感来源
中心化数据流	Redux <a href="https://redux.js.org/">https://redux.js.org/</a>

国际化文案	FormatJS <a href="https://formatjs.io/">https://formatjs.io/</a>
动效设计与实现	Ant Design <a href="https://ant.design/">https://ant.design/</a>
代码文件组织	Umi <a href="https://github.com/umijs/umi">https://github.com/umijs/umi</a>
函数式编程构建组件	React Hooks <a href="https://reactjs.org/docs/hooks-intro.html">https://reactjs.org/docs/hooks-intro.html</a>

## 拓展实现

Tiny Weather 能满足大部分用户日常最基本的天气预报需求，目前只给出三天预报是因为阿里云市场十五天预报的接口收费有点小贵，三天预报的数据由墨迹天气免费给出，中间隔了一层阿里云函数计算防止 Secret 和 Token 泄密。

## 缺点和不足

- 没有实现定位获取城市
- 没有完整的实现 Accessibility
- 无法无限量添加城市

最后一个其实也是 Tiny Weather 体验设计的一部分，限制最多添加 6 个城市，用户不会迷失在频繁的寻找城市操作内，我们需要的是用过合理的空间布局与交互响应，最大限度的提高用户删除、查找和添加城市操作的效率，将城市数量限制在 6 个以内，同时也考虑到布局渲染问题，过多的城市会导致在低配置设备上渲染掉帧，影响用户体验。

## 兼容性

Tiny Weather 同时兼容 Android@^5.0 和 iOS@^7.0，对不同比例屏幕的设备也有兼容，保证跨设备跨平台的应用渲染表现一致。

---

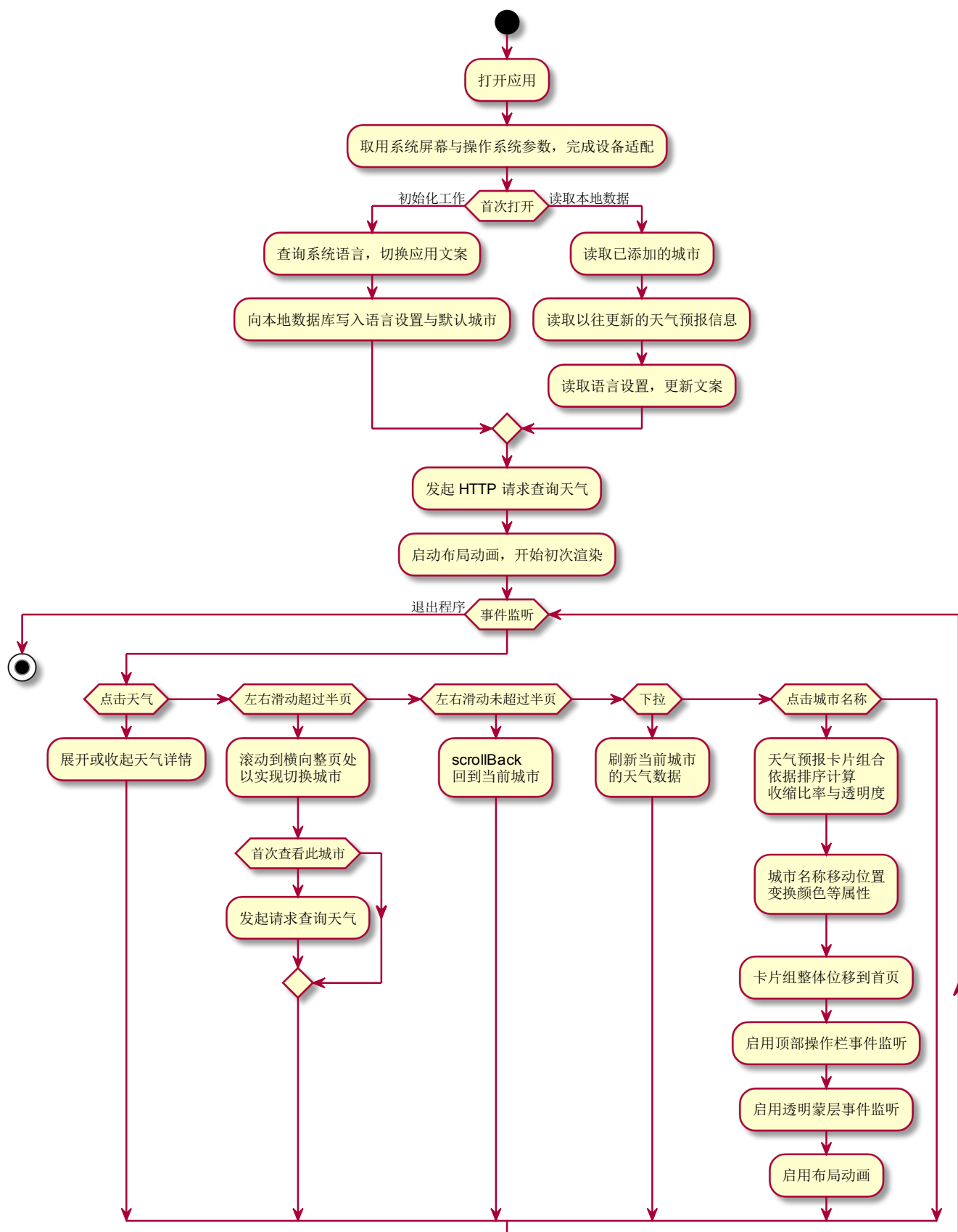
## 项目说明

### 功能模块

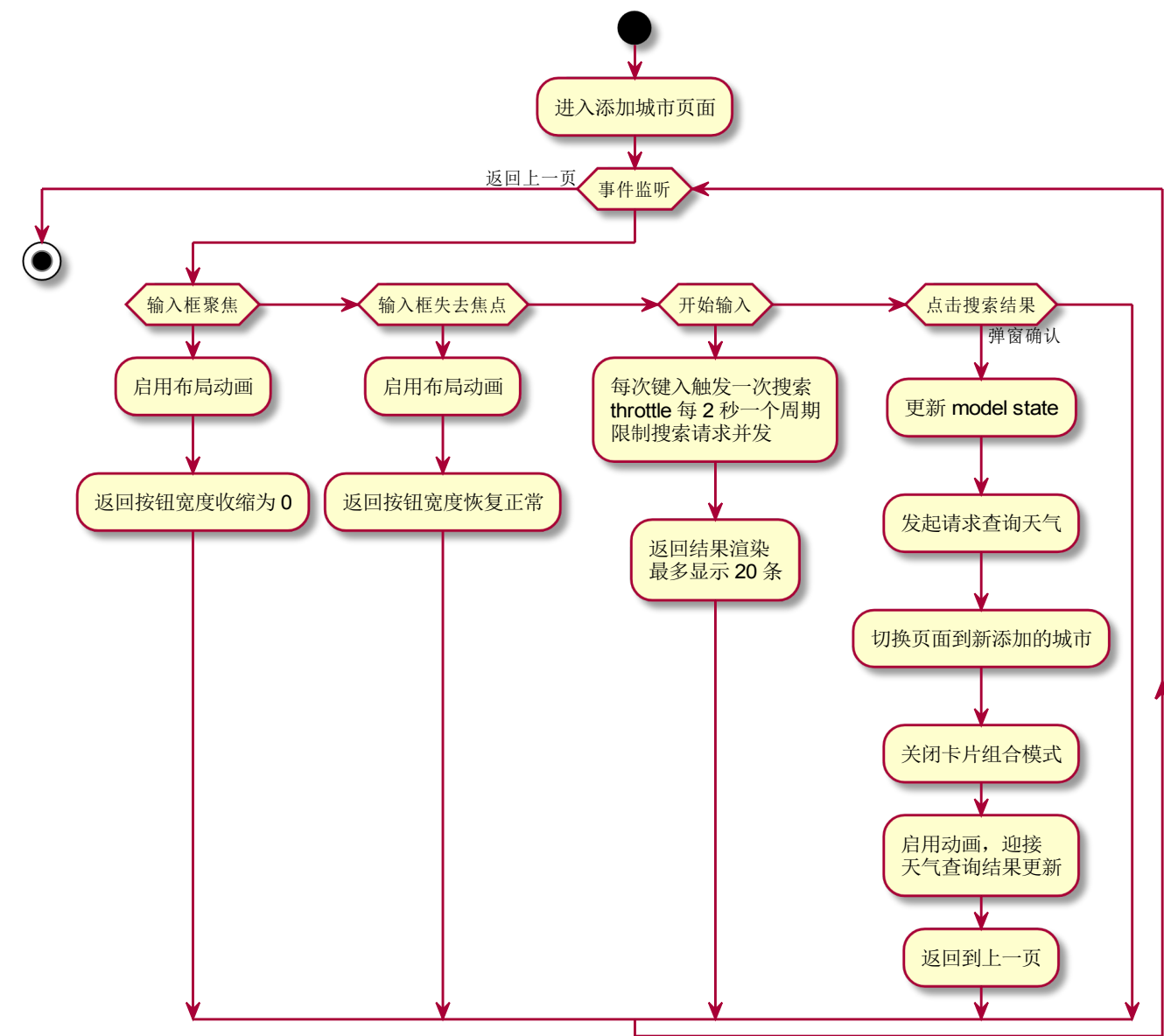


## 逻辑流程

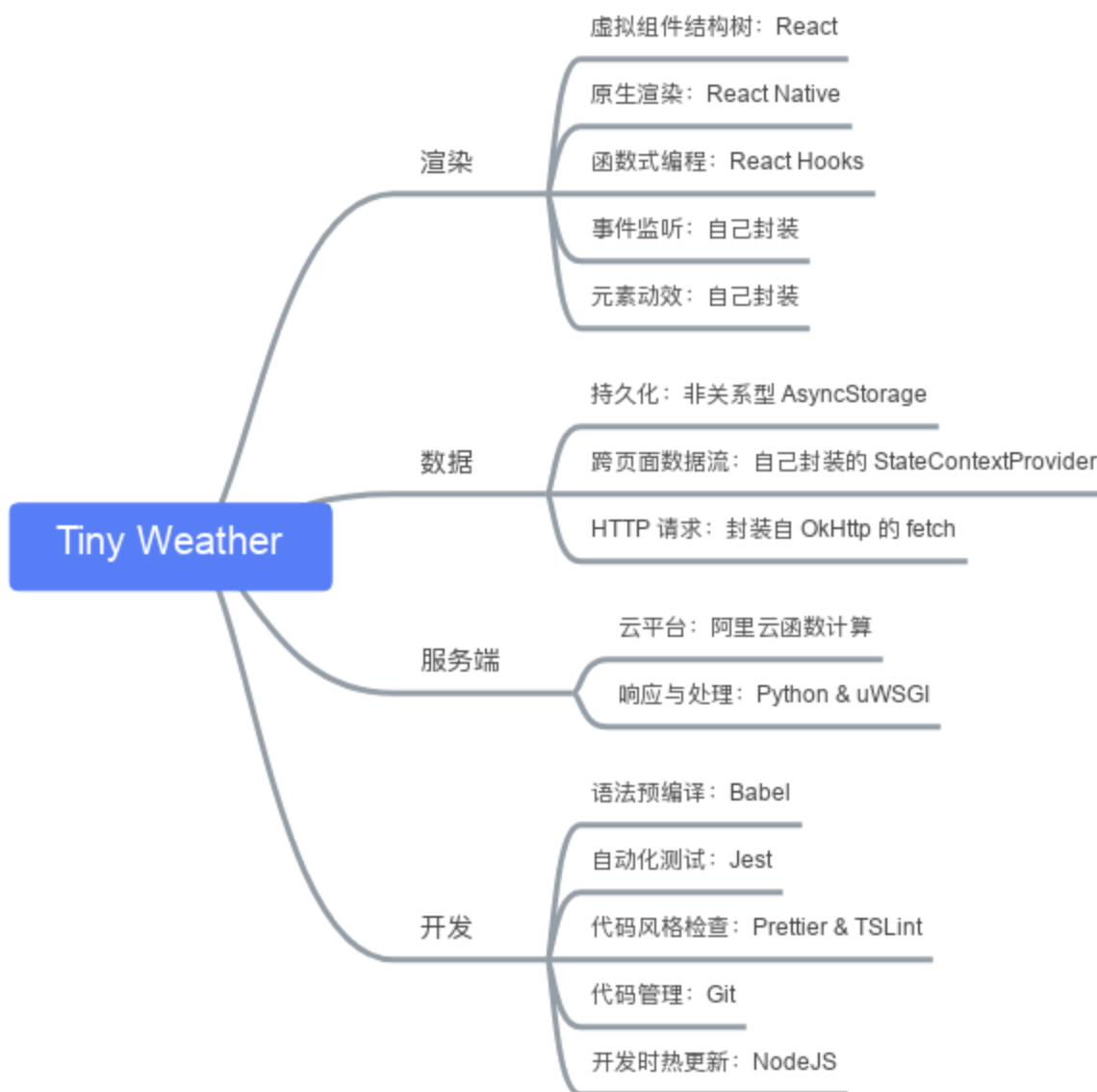
### 天气页



搜索页





技术说明



## 渲染

有一位组员在开源社区的 React 生态做了半年，出于开发效率考虑，我们在底层渲染中选择了 React Native，通过另一位组件在原生端代码桥接以利用 React 的 Virtual DOM 以及过去半年累积的开发经验。但实际上选择仍然处于先行版的 React Native 之后，开发并没有想象中那么顺利，周边生态极其糟糕，封装的组件库少得可怜也不怎么好看，没有开源的 RN 数据流方案，开发遇到的各种问题也搜索不到解决方案，只能翻来覆去看 React Native 的源码然后一遍一遍 debug，**但这是当时为了实现跨平台的最佳方案**，就当给开源社区造造轮子。还有一个重要的原因是 React Hooks，React 团队创造性的提出 Hook 概念，把函数变得不纯，以在保留 Function 的相对高性能的特性下实现 new Class 的功能。

## 数据

数据流方案灵感来自 `redux` 的架构，但 `redux` 没有原生应用上合适的实现方案，所以手写了一个简单的，没有做太多的渲染节流措施，但由于应用的数据流非常简单，简单的计时节流已经够用了，没有具体写 `diff` 算法。中心化数据流是我们用来实现跨页面跨组件数据通信的解决方案，同时我们按经验做了 `reducer`，**将异步操作从界面渲染逻辑中独立出来，防止异步操作阻塞页面渲染**，顺便监听了异步操作的执行状态，对外提供 `$loading` 变量，用以显示页面加载态。对于数据持久化，由于使用的地方比较少没有做封装，直接在异步操作中手动调用 `AsyncStorage` 的接口将数据保存在本地。数据流的 TypeScript 类型解决方案是其中一位组员在以往项目中总结的，通过 `Ant Design Pro`  `Star`  `19198` [<https://pro.ant.design>] 对外开源。

## 服务端

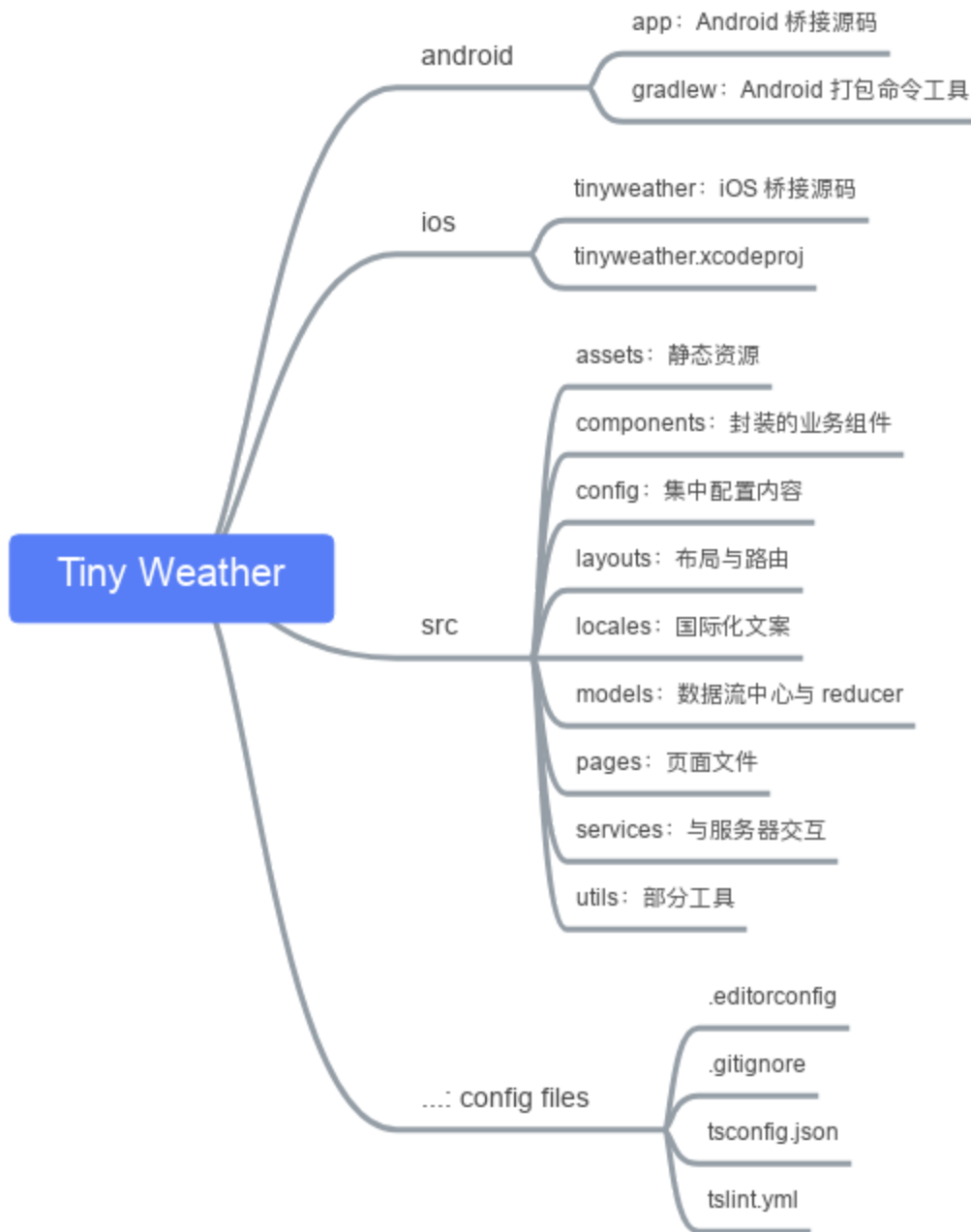
采用了阿里云的函数计算，没有用服务器是因为其中一个组员还在没学完 `docker`，而且 `Tiny Weather` 请求的功能比较简单，只有查询城市和查询天气，直接在函数计算做比较简单。

## 开发

这里按照以往的开发习惯初始化项目，项目根目录下的零碎文件基本都是用来提升开发效率的工具配置文件，用好开发相关的工具可以有效提高开发效率和写出来的代码质量。自动化测试没有做 `E2E` 冒烟测试，没有适配 `Web` 平台，也没有相关的文档介绍，`Chromium` 内核下载挂梯子也太慢，促使我们放弃了冒烟测试，项目部不大直接肉眼 `Debug`。

## 代码架构





React Native 将 src 文件夹下的具体实现内容编译打包到 android 与 ios 两个目录，构建最后的安装包则需要分别进入两个项目使用命令行工具或者 IDE 进行打包。以 Android 为例，桥接需要在 MainApplication.java 中额外引入 ReactApplication、ReactNativeHost、ReactPackage 并进行初始化，在 AndroidManifest.xml 中引入 `DevSettingsActivity`，再修改 build.gradle 打包相关配置，一般跟着教程走没有太大问题。

## src/assets

在这里的主要工作是引入天气图标，其中一位组员在自行摸索引入图标之后写过一篇博客《[React Native 引入 Icon Font](#)》，最终在 `src/components` 目录中封装为 `Icon` 组件，拥有完善的 IDE 自动补全支持：

```
55  /**
56   * *****
57   * `User` tab - stack navigator
58   * *****
59   */
60  const usertabConfig: NavigationScreenConfig<NavigationScreenOptions> = ({ navigationOptions }) => ({
61    tabBarIcon: <Icon type="us" />,
62    title: intl.upper('我的'),
63    ...navigationOptions,
64  });
65
66  const usertab = createStackN
67  {
68    ...CommonPages,
69    mine,
70  },
```

## src/components

一般应用开发都会遇到大量重复的元素，我们通常会将它们封装成组件放在这里。这里简要介绍一下我们封装的几个组件：

- Animation：封装动效组件，可配置动画效果、时长、延迟、动效曲线、启用原生线程绘制。
  - HoverScale：响应用户按压，阻止原生的按压响应效果，重新渲染为缩放与透明度变化。
  - PageContainer：提供 `ScrollView` 的功能，对外暴露下拉刷新事件监听的接口。
  - ScaleView：通过控制 `visible` 参数实现被包裹元素的缩放入场与离场，例如 `Tiny Weather` 中的“编辑”、“删除”、“添加”按钮以及搜索结果项都使用了这个组件避免闪现。
- Hooks：在应用的各个生命周期与事件监听的过程中会涉及到大量的回调，我们将所有回调统一封装在此，使用 `registerHook` 和 `registerHooks` 注册钩子，使用 `Hooks.onXXX()` 向回调函数队列推入新的回调函数，使用 `callHook` 批量调用回调函数队列中的函数。详情参见组员的博文《[轻量级 Hooks 方案](#)》。
- Icon：图标。
- intl：国际化文案，支持语言跟随系统、语言设置持久化、动态修改语言、`lodash template` 格式化，并对 `template` 格式化有做缓存，例如 添加 `<%= cityName %>` 到列表 文案，可以通过 `intl(textId, { cityName: '北京' })` 以输出格式化后的文案 添加 北京 到列表 。
- Wrapper：统一的外层容器包裹器，对于中心化数据流等功能，需要在每一个页面的最外层提供一个 `ContextProvider` 以提供数据穿透，在这里统一对外暴露一个组件 `Wrapper` 与一个方法 `wrap`，在页面外包裹一层 `Wrapper`，在其他代码文件里面调用 `wrap` 方法来添加包裹层。

## src/config

这里集中了应用的大部分配置，包括主题颜色配置、字号配置、设备与设计稿分辨率适配、API 请求地址前缀、国际化默认语言等内容。

## src/layouts

这里实现路由，封装了 `push()`、`back()`、`navigate()`、`replace()` 等路由方法，并统一对各个页面做初始化工作。

## src/models

数据流中心，包含以下内容：

- `merge()`：递归合并 previous state 与 next state
- `$STATE`：所有 model 的数据都保存在这里
- `StateContext`：上下文提供方，包裹在页面最外层，在子组件中通过 `connect` 装饰器取用
- `setState`：更新一部分数据，并启动新一个周期的渲染
- `wrapReducers()`：用于包裹各个 reducer 方法，监听异步 reducer 的运行状态，并将 reducer 的返回结果 merge 进 `$STATE`
- `dispatch`：暴露给外界调用 reducer 的地方，例如需要在页面中刷新天气则需要调用 `dispatch.weather.fetchWeather()`，`dispatch` 内的所有函数调用后都将返回一个 Promise 对象而不会阻塞当前函数
- `connect(mapFunc)(Component)`：组件装饰器，传入一个函数用于取用 `$STATE` 中的部分内容，返回一个函数用于包裹组件

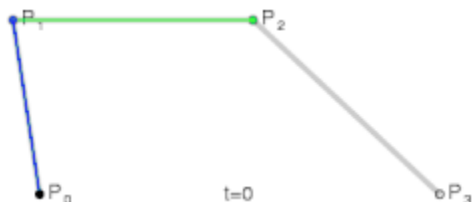
## 应用效果

为了帮助用户快速获取有效信息，我们做出了如下努力：

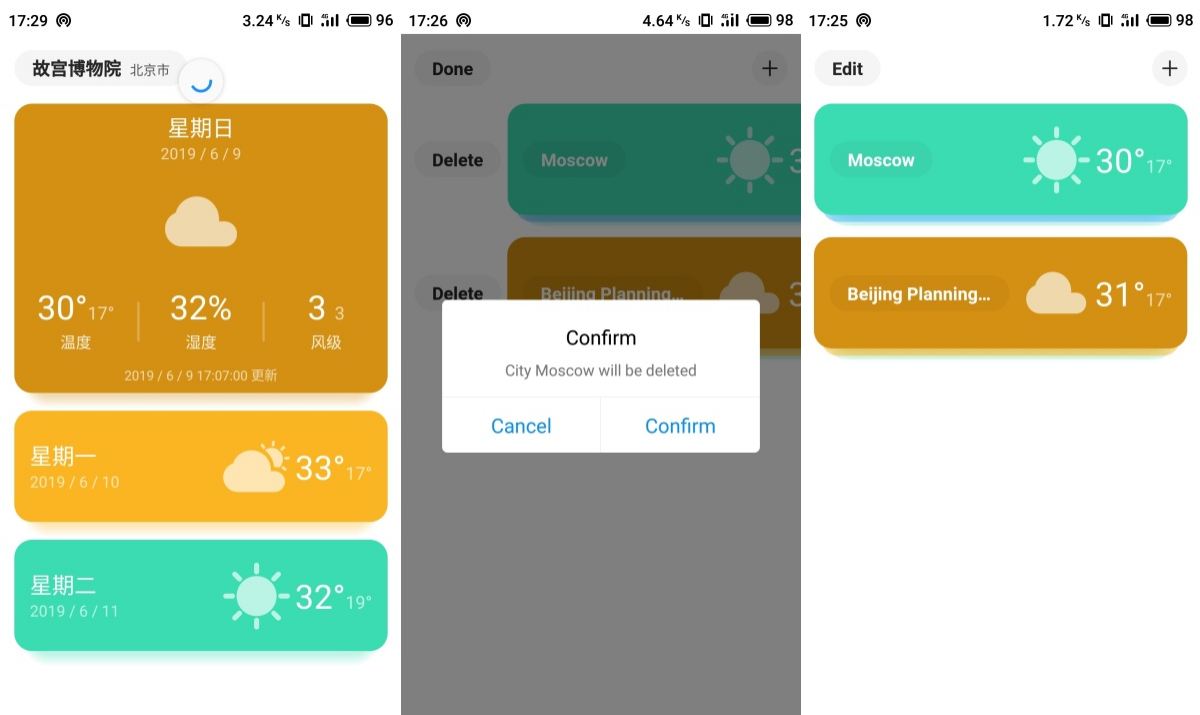
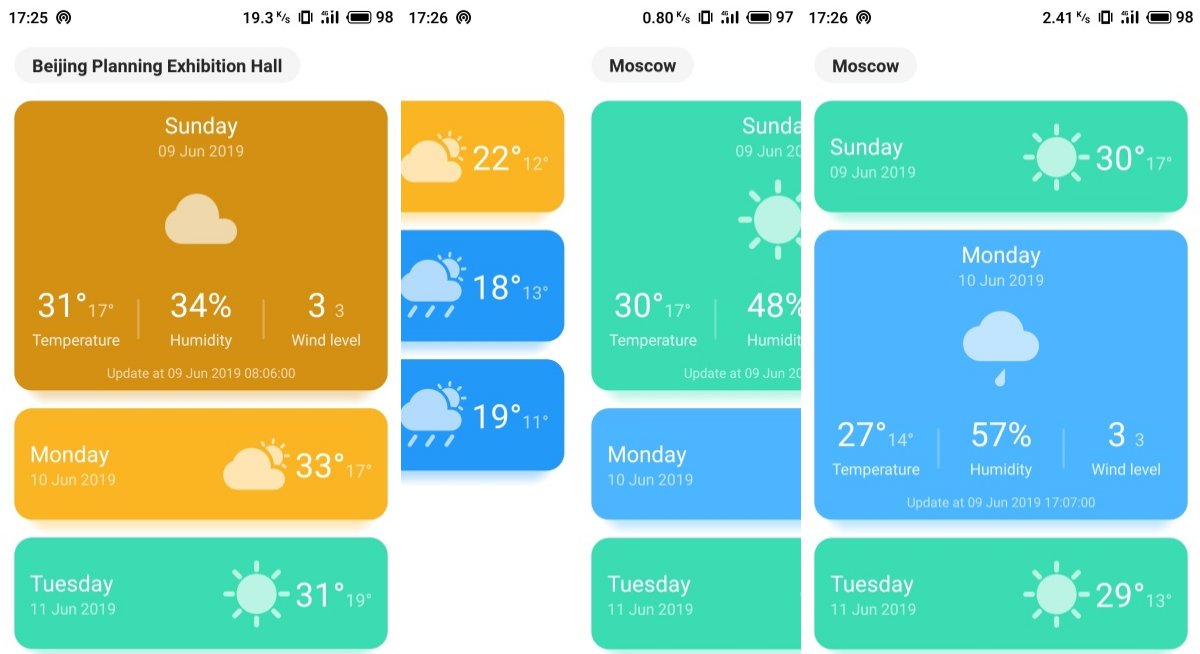
- 精心调配的背景颜色，精密计算的主题颜色与黑白灰颜色层级，保证饱和与对比的平衡以提供舒适的视觉体验



- 精心调制的贝塞尔曲线参数，还原自然运动速率，让元素内容与位置的改变不显得突兀



- 同样的天气，白天与夜间的背景颜色与图标不同，白天优先展示最高温度与白天风级，夜间与此相反
- 白天与夜间的判定界限根据城市所在的时区所确定
- 为所有的布局移动添加动效过渡，帮助用户感知数据的来源与去处
- 及时响应用户按压、拖拽等交互操作，给出操作暗示的同时向外界证明应用并没有卡死
- 卡片式布局聚合有效信息，帮助用户快速聚焦，重要文字加大加粗显示，附加信息降低透明度





## 开发过程中遇到的问题及解决办法

### 问题 1：左右滑动如何停留在整数页？

对于开源社区的所有 swiper 封装我们都不满意，原因是所有的封装对 swiper item 超出边界的内容都会隐藏，而我们管理城市的功能就是需要通过将各个 swiper item 的元素相对位移到第一页来展现，所有我们自己封装了一个 Swiper，在用户滑动超过半页边界时切换 swiper 页面，未超过半页则滚动回上页。在管理城市模式时，由于各个 swiper item 内容时相对位移出来的，原先的宽度仍然存在，需要主动滑回首页，并禁用滚动，这里的封装也暴露了 scrollTo 方法来实现。

### 问题 2：如何保证卡片组的高度与数量无关？

我们构思了一个简单的公式  $16 * (1 - 1 / (index * index + 1))$  来保证卡片之间的距离逐层递减，并始终不超过 16，透明度的变化同理。

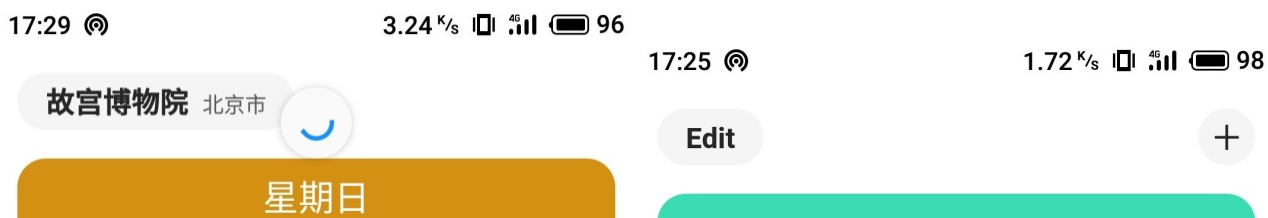


### 问题 3: Icon Font 引入后乱码?

一开始是以为命令行工具在打包 src 到 android 的时候忘记拷贝 iconfont.ttf 文件了, 然后试着看两个文件的 hash 发现并没有区别, 手动拷贝过去也有问题, 然后在页面其他位置使用也没问题, 只有在添加按钮上的“+”有问题, 最后覆盖重写“+”外界继承的所有属性, 发现是 iconfont.ttf 并不支持 font-weight。

### 问题 4: 绝对定位的渲染层级问题?

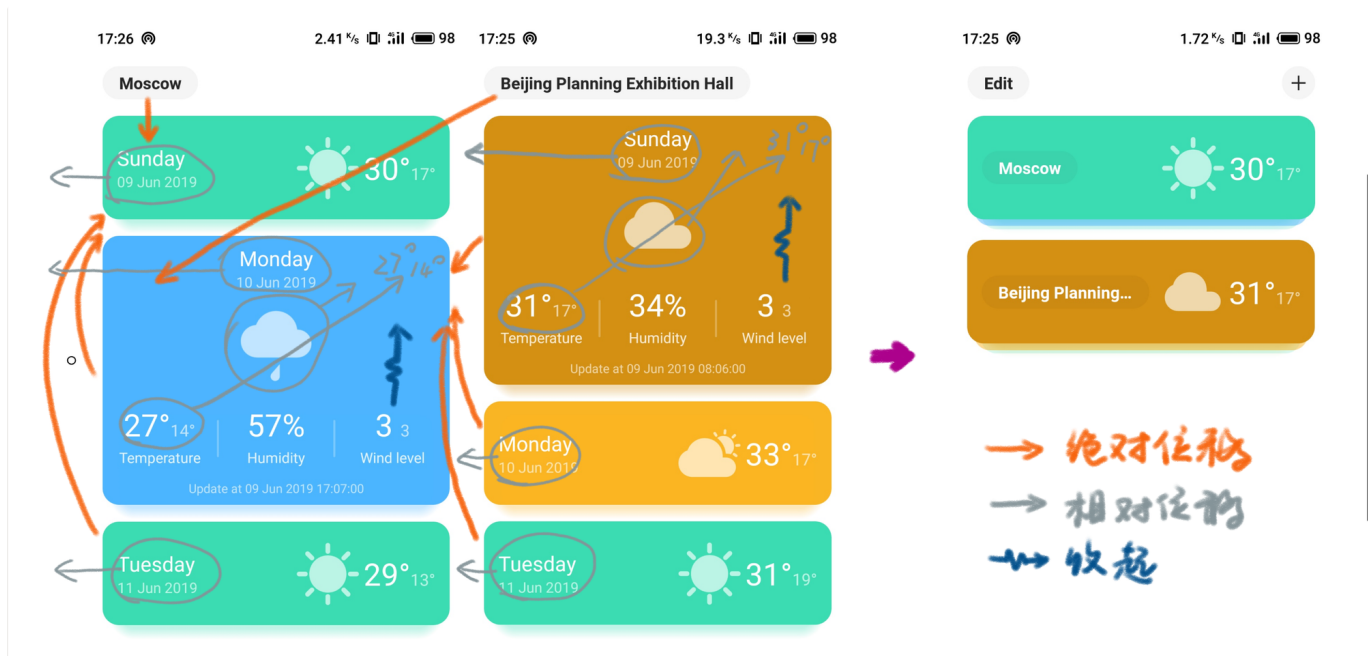
一般来说, 同层级后渲染的组件总是在先渲染的组件之上展示, 这就有一个问题, 天气预报的第三天和第二天日期在后, 但是三天卡片收起时需要第一天在最上层, 这个好解决, 倒过来渲染就行, 布局位置公式重新写一个。但还有另一个问题, “编辑”按钮的位置恰好是城市名称所在的位置, 两者的事件监听位置重复, 渲染层级切换并没有用, 然后卡片组也是相对位移过来的, 无法在相对位移的位置监听到触摸事件, 处于上述各种问题, 在管理城市的模式下创建了一个顶层透明蒙层, 统一监听和处理触摸事件, 通过触摸位置计算应该由哪个元素响应。



### 问题 5: 应用打包安装后内容没有变化?

打包的时候忘记更新版本号了, 打包的内容没有变化。

### 问题 6: 页面元素如何位移?



## 思考及感想

首先是学到了移动应用开发的基础概念，包括 Android 的 Activity、控件、权限、网络等等，这些内容也在开发实践中得到了进一步的认识和理解。除此之外，由于我们希望实现跨平台，在开发过程中又通过自行探索、查看底层源码等方式进一步了解了原生应用的渲染和运行机制，以及 Android 和 iOS 两个平台原生应用开发的区别与联系。我们也将本次课程的作业开源到 GitHub 上，有机会也可以成为他人学习参考的范例。

在做 Tiny Weather 的时候，发现移动端开发在交互上最大的一个特点就是需要牢记“心中的拇指”，用户的绝大部分交互行为都通过手指来完成，所有在展现每一个元素时，都需要想象这个元素如何响应用户的触摸、如果元素现在还不想响应用户的触摸该如何告诉用户、用户不小心误触了怎么办等等。实际上，移动应用的开发重点就在绘制与交互上，大型手游等应用需要考虑高效绘制，而无论简单或复杂的应用都需要充分考虑交互，在开发 Tiny Weather 时我们也参考了 Human Interface Guidelines、Ant Design、Material Design 等一些经典的设计体系来构思一套良好的应用交互方式，然后再花大量的功夫去用代码实现这些交互。

由于时间关系我们展示的 App 有一些 bug，在撰写报告期间修复并重新打包了 2.1.0 版本。在开发中我们也遇到了很多很多很多的问题，最后还是被一个一个一个解决了，实际去开发收获总是很大的。

## 参考文献

1. React Native: <https://facebook.github.io/react-native/>

2. Use Transition: <https://ant.design/docs/spec/transition>
3. React Immediately: <https://ant.design/docs/spec/reaction>
4. Colors: <https://ant.design/docs/spec/colors>
5. Hooks: <https://reactjs.org/docs/hooks-intro.html>
6. Human Interface Guidelines: <https://developer.apple.com/design/human-interface-guidelines>
7. Material Design: <https://material.io>
8. 函数开发指南: [https://help.aliyun.com/document\\_detail/75152.html](https://help.aliyun.com/document_detail/75152.html)
9. 墨迹天气免费版: <https://market.aliyun.com/products/57096001/cmapi023656.html>
10. Ant Design Pro: <https://github.com/ant-design/ant-design-pro>