# Chosyeat 基础框架

计科1702班 17231239 郑志伟

## Chosyeat 是什么？

在物质生活逐渐丰富的中国，越来越多的人开始重视自己的身体健康状况，但如何准确把握自身的身体状况、合理搭配每天都需要摄入的食物是个难题，由此诞生了 Chosyeat —— 你的营养搭配专家。

## 从0开始搭建工程

### React Native

本人使用React框架半年，封装过一些组件库与工具，TypeScript用的比Java熟，所以选择支持热加载的React Native进行开发，这里只针对iOS与Android两个平台进行适配。

使用react-native-cli初始化项目之后，按我的个人习惯将工程改造成如下目录：

.

├── \_\_tests\_\_ // 单元测试文件

├── .idea // Android Studio编辑器配置文件

├── .vscode // 个人习惯VS Code编辑器

├── android //

├── ios //

├── src

├── assets // 静态资源文件

├── components // 项目中抽离出的组件

├── config // 全局配置项

├── layouts // 页面布局与路由

├── locales // 国际化文案目录

├── models // 数据流

├── pages // 页面UI文件

├── services // 服务端API交互专用

└── utils // 小工具

├── .editorconfig // <http://editorconfig.org>

├── .prettierrc // Prettier配置文件

├── app.json // 应用基础配置

├── babel.config.js // Babel编译配置

├── index.js // 入口文件

├── package.json // 项目信息

├── README.md

├── rn-cli.onfig.js // react-native-cli配置

├── tsconfig.json // TypeScript编译配置

├── tslint.yml

└── typings.d.ts

为什么会有这样的目录结构？详情可查看[前17条GitHub Commit记录](https://github.com/imhele/chosyeat-client/commits/511215bbdabd12468e9b6dcd65fca73fd7ccf373)，从0开始搭建src目录，当然这并不是必须的，在以后的React Native项目中可以重复使用这套框架。

### 准备工作

按照以往的开发习惯，先做如下准备工作：

1. 添加代码风格检查，新增配置文件tslint.yml，从自己以往的项目中复制累积的配置规则，保证多人协作看彼此的代码更习惯；
2. 添加自动排版工具prettier，使代码排版统一；
3. 添加git pre-commit自动检查，自动修正部分错误，检查排版是否有误差；
4. 锁定依赖包版本；
5. 配置文件夹别名，这样移动文件时不需要更新import语句；
6. 配置TypeScript、gitignore、Babel等；
7. 新增./src/utils/types.ts文件，补充常用类型定义。

### 路由

一个应用一般是多页面的，这里选择官方推荐的react-navagation进行自定义包装：首先构建顶层容器AppContainer，在这里监听全局渲染生命周期，在初次渲染完成之后暴露navigator导航API；然后封装常用的导航methods；最后编写页面之间的Stack与Tab等导航逻辑与配置项。此部分文件在./src/layouts/Routes目录下。

### 引入Icon Font

这里额外写了篇博客，详见附件，或在线阅读[《React Native 引入 Icon Font》](https://www.yuque.com/hele/react/am5cdd)。

### 生命周期钩子方案

在开发中遇到了大量的跨文件、跨组件、跨生命周期的回调，因此构思了一个轻量级的Hooks方案，专用于注册钩子、储存回调函数。详见附件的博客，或在线阅读[《轻量级 Hooks 方案》](https://www.yuque.com/hele/react/bg2sbb)。

### 全局配置

做过五年平面设计和一年交互，对页面原子设计的实现有极其严苛的要求。作业开始做的有点晚，所以简单做了一下：首先构建了一套层级主题色、透明度与色彩体系，从./src/config导出为Color以供页面绘制UI使用；其次构建了一套字体体系，这是去年做出来的，详情参阅本人的博客[《Design Principle – Text》](https://www.yuque.com/hele/design/bjtu-text);React Native 默认使用dp单位，但一般设计稿使用px，Web开发也更熟悉px，这里集中封装了Pixel，用于根据屏幕dpi与系统font scale配置转换单位。

### 国际化

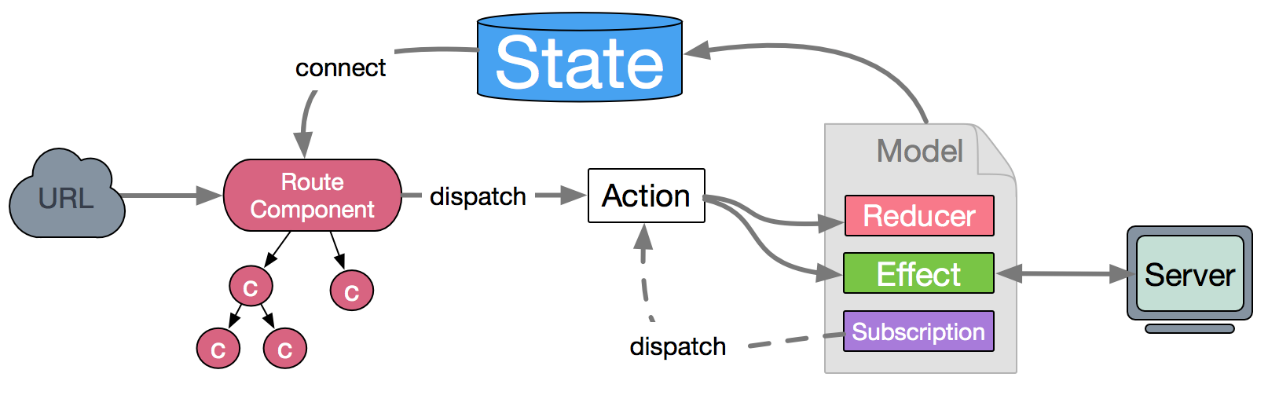
在社区给UmiJS做了一段时间的国际化方案，顺手在React Native实现了一套轻量级的国际化文案方案，拥有如下特性：

* 支持[lodash/template](https://lodash.com/docs/4.17.11" \l "template" \t "_blank) 处理文案；
* 支持语言设置持久化；
* 支持跟随系统配置变更语言；
* 支持TypeScript文案ID自动补全；
* 翻译国际化文案时也支持TypeScript自动补全；
* 支持部分语言首字母大写或全大写，基于toUpperCase。

这里又写了一篇博客，详见附件，或在线阅读[《React Native 轻量级可配置国际化方案 - with TypeScript》](https://www.yuque.com/hele/react/ghh52d)。

### 数据流

React Native 并没有合适的专用数据流解决方案，那就自己实现一套：



在./src/models/index.ts中集中储存state，并包裹各个model的reducer，深层嵌套组件可以通过dispatch来修改其他组件的状态与渲染状态，或者触发请求服务端数据，集中监听reducer函数是否正在执行（以便在页面中展现加载态），集中catch错误与页面rerender重渲染节流。《100 行内的轻量级数据流》还没写完，可能后面的报告见。

### 动效

Native端的动画实现比CSS声明式难得多，不过性能也更好。目前只封装了一个HoverScale组件，在用户按住组件并未抬起时，组件将展现缩放状态，可配置缩放程度与是否透明，默认启用原生线程驱动UI动画，可在演示视频中看到效果。

## 附件

* 轻量级 Hooks 方案
* React Native 引入 Icon Font
* React Native 轻量级可配置国际化方案 - with TypeScript
* chosyeat-client-20190520.mp4
* chosyeat-client@1.0.2.apk