## react 最佳实践

* 一个文件一个组件。
* 尽量使用无状态（Stateless）组件，也就是如果只是写一个单纯展示的组件，不需要组件保存自己的状态，不需要生命周期方法或者 refs 来操作 DOM 的组件则优先使用无状态组件，采用函数的形式。以项目 Tips 组件示例:

**import** React **from** "react";

**import** PropTypes **from** "prop-types";

**import** styles **from** "./tips.scss";

**export** **default** **function** **Tips**({ title, content }) {

**return** (

<div className={styles.tips}>

<p className={styles.title}>{title}</p>

<p className={styles.content}>{content}</p>

</div>

);

}

Tips.propTypes = {

title: PropTypes.string.isRequired,

content: PropTypes.string.isRequired

};

* 和上面相反，如果你需要组件生命周期方法优化组件性能（典型应用，重写 shouldComponentUpdate 方法），需要组件保存自己的状态，或者用 refs 操作 DOM，你就需要一个有状态组件，采用 es6 class 继承 React.Component 的写法。组件示例：

**import** React **from** "react";

**import** PropTypes **from** "prop-types";

**import** classnames **from** "classnames";

**import** styles **from** "./cell.scss";

**import** { isObjEqual } **from** "../../utils/helpers";

**export** **default** **class** **Cell** **extends** **React**.**Component** {

**static** propTypes = {

value: PropTypes.number.isRequired

};

shouldComponentUpdate(nextProps, nextState) {

**return** (

!isObjEqual(nextProps, **this**.props) || !isObjEqual(nextState, **this**.state)

);

}

render() {

**const** { props: { value } } = **this**;

**const** color = `color-${value}`;

**return** (

<td>

<div

className={classnames([styles.cell, { [styles[color]]: !!value }])}

>

<div className={styles.number}>{value || null}</div>

</div>

</td>

);

}

}

* 事件绑定 this 方法。在构造函数里面绑定一次 this 之后后面就可以正常使用。以 [ControlPanel](https://github.com/devrsi0n/React-2048-game/blob/e6812e8b89bb38109387e7f6495fcd5d70c11f26/src/containers/ControlPanel/index.js) 组件部分代码示例：

**constructor**(...args) {

**super**(...args);

**this**.handleMoveUp = **this**.handleMoveUp.bind(**this**);

**this**.handleMoveDown = **this**.handleMoveDown.bind(**this**);

**this**.handleMoveLeft = **this**.handleMoveLeft.bind(**this**);

**this**.handleMoveRight = **this**.handleMoveRight.bind(**this**);

**this**.handleKeyUp = **this**.handleKeyUp.bind(**this**);

**this**.handleSpeakerClick = **this**.handleSpeakerClick.bind(**this**);

**this**.handleUndo = **this**.handleUndo.bind(**this**);

}

* 使用 [propTypes](https://reactjs.org/docs/typechecking-with-proptypes.html) 属性进行传入 prop 的校验。可以校验 prop 的类型和是否必需，非必需的 prop 还必需填写 defaultProps 默认值。以无状态组件 [Button](https://github.com/devrsi0n/React-2048-game/blob/e6812e8b89bb38109387e7f6495fcd5d70c11f26/src/components/Button/index.js) 的部分代码示例：

Button.propTypes = {

children: PropTypes.oneOfType([PropTypes.node]),

onClick: PropTypes.func,

size: PropTypes.oneOf(["lg", "md", "sm", "xs"]),

type: PropTypes.oneOf([

"default",

"primary",

"warn",

"danger",

"success",

"royal"

]).isRequired

};

Button.defaultProps = {

children: "",

onClick() {},

size: "md",

};

* 使用 [HOC(Higher-Order Components)](http://huziketang.com/books/react/lesson28) 代替 mixin。mixin 官方已经不推荐使用了，redux 的 connect 方法就是 HOC 的应用。
* 为了提高应用性能，避免不必要的视图重绘，在需要的组件使用 shouldComponentUpdate 方法；以组件 [Row](https://github.com/devrsi0n/React-2048-game/blob/e6812e8b89bb38109387e7f6495fcd5d70c11f26/src/components/Row/index.js) 示例：

// 如果该行没有格子需要刷新也没有组件自己的状态刷新，

// 则该组件不执行 render 方法，

// 避免每次别的行数据刷新也跟着重新渲染。

shouldComponentUpdate(nextProps, nextState) {

**return** (

!isObjEqual(nextProps, **this**.props) || !isObjEqual(nextState, **this**.state)

);

}

## 项目结构

本项目是基于 Facebook 官方出品的 [create-react-app](https://github.com/facebookincubator/create-react-app) 脚手架搭建的，reject 后做了适当修改以适配项目需求。

### 调整如下

* [webpack](https://github.com/webpack/webpack) 添加 [scss](http://sass-lang.com/guide) 支持。之所以没有用 [CssInJS](https://medium.com/@gajus/stop-using-css-in-javascript-for-web-development-fa32fb873dcc) 的方案是因为这些方案普遍不完美，也考虑到要遵循样式和结构分离的原则，scss 是目前比较成熟的 css 预处理器，社区轮子也比较多，开发起来很方便。推荐学习 scss/sass [教程](http://www.sassshop.com/#/1/2)。添加 sass-loader 到 scss 规则下面最下面即可。[配置代码](https://github.com/devrsi0n/React-2048-game/blob/149d75e117c048a44704315a6122e0e28c256a97/config/webpack.config.dev.js#L190)
* 开启 [css module](https://github.com/css-modules/css-modules) 支持。在大型项目里面组件之间需要尽量解耦，但是 css 类名的全局特性很容易导致意料之外的错误。开启 css module 之后，所有的类名最终都会被一小段 hash 值填充，所以类名也就有一定的唯一性，不容易污染全局的代码。[配置代码](https://github.com/devrsi0n/React-2048-game/blob/149d75e117c048a44704315a6122e0e28c256a97/config/webpack.config.dev.js#L170)
* 添加 [stylelint](https://github.com/stylelint/stylelint) 支持。js 代码已经有 [eslint](https://github.com/eslint/eslint) (但采用了更流行，校验更严格的 [airbnb](https://github.com/airbnb/javascript/) 规则) 来检查代码，但是样式代码也需要保持代码风格统一，同时校验规则一般有社区的最佳实践。[配置代码](https://github.com/devrsi0n/React-2048-game/blob/149d75e117c048a44704315a6122e0e28c256a97/config/webpack.config.dev.js#L251)
* 添加静态资源 cdn 支持。由于项目部署在 [github page](https://pages.github.com/) 在国内访问速度不是很理想，所以在可能的情况下尽量减小 js 包的大小对页面加载速度至关重要。像 ReactDOM 这类较大的 npm 包从打包文件剥离出去采用 CDN 来加载，可显著减小打包文件的大小。（PS：之所以 CDN 加载比较快，是因为 CDN 提供商在全国各地都建立了缓存服务器，资源就近获取比自己从 github 获取快得多，而且一般 CDN 的带宽也比较充裕）把 React 和 ReactDOM 剥离出去只需要在 html 文件添加 CDN 的 [script 标签]()，同时在 webpack 添加 [externals](https://github.com/devrsi0n/React-2048-game/blob/e6812e8b89bb38109387e7f6495fcd5d70c11f26/config/webpack.config.prod.js#L77) 属性，该属性指定代码 import 该包时直接从全局变量获取。剥离后打包的 js 文件大小从 278kb 减小到 164 kb。
* 添加 webpack [代码压缩](https://github.com/webpack-contrib/compression-webpack-plugin)插件。默认的 webpack 配置直接输出原始的 js，css 代码，但添加压缩过后，文件显著减小（js 文件从 164kb 到 49kb），对于移动浏览器来说打开速度得到明显提升。[配置代码](https://github.com/devrsi0n/React-2048-game/blob/25099b82afe7b32d060b0957862e4d1d397fc539/config/webpack.config.prod.js#L329)
* 添加 [webpack-bundle-analyzer](https://github.com/webpack-contrib/webpack-bundle-analyzer) 插件，通过各模块包所占打包文件后的比重来分析项目代码，借此优化代码。比如，React 和 ReactDOM 的剥离就是因为分析后发现这两个包所占比重较大。

### 文件结构

* src， 项目源代码大部分都在这里，主要是 react 组件 js 代码 和 scss 样式代码。次级目录包含了 [jest](https://github.com/facebook/jest) 单元测试代码，测试代码尽量和源代码挨着，以方便编写。
  + assets，主要存放一些全局样式代码，icon svg 文件，游戏音效 mp3 文件，图片等等；
  + components，存放 [react dumb 组件](http://huziketang.com/books/react/lesson43), 每个组件包含在采用首字母大写的目录的 index.js 里面，同时该目录包含该组件用到样式的 scss 文件，尽量一个目录包含该组件所需的所有代码避免污染其他代码，提高组件复用性。
  + containers，存放 [react smart 组件](http://huziketang.com/books/react/lesson43)，该目录结构和 components 类似，但因为是 smart 组件，所以这里的组件可以操作 redux 的数据，不用太考虑复用性。
  + reducers，这是 redux 包含的是无副作用的纯函数式计算状态操作的函数。
  + utils，包括评论组件初始化，i18n 多语言文件，移动浏览器滑动检测和注册 [ServiceWorker](https://developer.mozilla.org/zh-CN/docs/Web/API/Service_Worker_API) 等等。
  + index.js，项目入口文件，主要把 react 根组件 渲染到指定 DOM 节点，并且注册 [ServiceWorker](https://developer.mozilla.org/zh-CN/docs/Web/API/Service_Worker_API)。
  + store.js，redux store 初始化，同时 store.subscribe 订阅应用状态更新，序列化状态存到 localStorage。
* public，包括项目的 html 文件，网站 icon favicon 和 [PWA manifest](https://developer.mozilla.org/zh-CN/docs/Web/Manifest) 文件。
* config，主要包括 webpack 的各种配置文件。
* scripts，npm 的启动脚本，启动开发模式，项目打包，运行 jest 单元测试等等。
* build，项目打包后的输出目录。
* screenshots，README 各种图片的原图，为了国内用户访问方便实际上 README 的图片来自新浪微博的图床。
* [.editorconfig](http://editorconfig.org/)，通用的编辑器配置，统一不同编辑器 / IDE 的代码格式。
* .eslintignore，需要 eslint 忽略的文件或者目录，规则类似 .gitignore
* .travis.yml, 持续集成脚本，每次提交代码到 github 之后，测试服务器都会自动运行该脚本执行测试用例，并输出代码覆盖率，最后自动部署到 [github page](https://pages.github.com/)。所有状态都在项目中 README 的徽章中可见。
* package.json，项目基本信息和部分配置都存在这里。常见的内容包括项目的各类依赖包，各种启动脚本，项目 homepage 等等；为了减少根项目的文件数目，jest，babel，eslint，stylelint 的配置也写在这里。值得注意的是，项目中引入 [husky](https://github.com/typicode/husky)，在每次代码 commit 之前都会执行 [lint-staged](https://github.com/okonet/lint-staged)，以自动执行 [prettier](https://github.com/prettier/prettier) 来美化代码格式。每次代码推送 到 github 之前也会执行所有单元测试用例，全部通过才可以继续推送。
* yarn.locl，yarn 首次安装依赖包之后生成的 lock 文件。通过 yarn 来安装依赖包时，yarn 自动把项目的依赖包（包括依赖包依赖的父级包）固定在指定的版本（包括依赖包安装的 url 和 hash 值），这样所有开发环境都使用 yarn 来管理项目，不同的机器不同的系统安装出来包都是一样的，这样就避免了之前 npm 的缺陷（版本要求太松或者父级包版本更新等等导致每次安装出来的依赖版本不一样）。

## 技术栈

* [react](https://github.com/facebook/react)，组件式构建 UI
* [redux](https://github.com/reactjs/redux)，管理应用状态
* [babel](https://github.com/babel/babel)，把 es2017+ 语法转成 es5 兼容语法
* [webpack](https://github.com/webpack/webpack)，代码热加载，scss 样式文件处理，组件编译打包等等
* [scss](https://github.com/sass/sass)，成熟的 css 预处理器（之所以没有用 CssInJS 的方案是因为这些方案普遍不完美，也考虑到要遵循样式和结构分离的原则）
* [eslint](https://github.com/eslint/eslint)，使用流行的 airbnb 的代码规范严格约束代码风格
* [stylelint](https://github.com/stylelint/stylelint)，scss 代码风格检查
* [jest](https://github.com/facebook/jest)，fb 出品的代码测试框架，snapshot 功能对测试 react 组件 UI 十分方便
* [Prettier](https://github.com/prettier/prettier)，js 和 scss 代码格式美化工具
* [PWA](https://zhuanlan.zhihu.com/p/25167289)(Progressive Web Apps)，借助浏览器 service worker 能力，使 web 应用在移动平台有接近原生应用的能力，可离线使用，接收通知消息等等

## 运行 & 测试 & 打包

因为配置文件用了 es6+ 语法所以要求 node 的版本大于 6.10，同时建议使用 [yarn](https://yarnpkg.com/zh-Hans) 来管理依赖包。fork 项目之后可以按如下命令操作。

npm i -g yarn # 安装 yarn

git clone git@github.com:<你的名字>/React-2048-game.git

cd React-2048-game

yarn # 安装依赖包

yarn start # 开启调试模式，启动后自动打开浏览器 http://localhost:3000

yarn test # 自动测试

yarn build # 打包代码

## 踩坑记录

* 在调烟花动画的时候发现没效果，仔细对比了下 webpack 编译后的 css 文件发现所有的 @keyframes 的名字都加了 hash 值（也就是当成普通的局部 css 类名），解决办法就是在 @keyframes 的名字前面和整个 scss 文件添加伪类 :global，可以参考烟花的 scss 文件，这不是完美的解决办法(css 类名不再有局部特性)，后续再深挖一下。
* css module 用到的 :global 这个不是标准的伪类，所以 stylelint 需要添加配置以忽略这个错误。参见 package.json 的 stylelint.rules。