**React+Redux架构的CMSweb后台管理系统**

一、CMS管理系统功能

**CMS**是**C***ontent***M***anagement***S***ystem*的缩写，意为"内容管理系统"。

CMS都有可能包括些什么？

隐藏在[内容管理系统](https://link.jianshu.com?t=http:/baike.baidu.com/view/857578.htm)(CMS)之后的基本思想是分离内容的管理和设计。页面设计存储在模板里，而内容存储在数据库或独立的文件中。 当一个用户请求页面时，各部分联合生成一个标准的[HTML](https://link.jianshu.com?t=http:/baike.baidu.com/view/692.htm)（[标准通用标记语言](https://link.jianshu.com?t=http:/baike.baidu.com/view/5286041.htm)下的一个应用）页面。

一个内容管理系统通常有如下要素：

文档模板

脚本语言或标记语言

与数据库集成

内容管理系统也简化了网站的内容供给和内容管理的责任委托。很多内容管理系统允许对网站的不同层面人员赋予不同等级的访问权限， 这使得他们不必研究操作系统级的权限设置，只需用浏览器接口即可完成。

[内容管理系统](https://link.jianshu.com?t=http:/baike.baidu.com/view/857578.htm)被分离成以下几个层面：各个层面优先考虑的需求不同

1，后台业务子系统管理（管理优先：内容管理）：新闻录入系统，[全文检索](https://link.jianshu.com?t=http:/baike.baidu.com/view/440795.htm" \t "_blank)子系统等，针对不同系统的方便管理者的内容录入：所见即所得的编辑管理界面等，清晰的业务逻辑：各种子系统的权限控制机制等；

2，前台发布（效率优先：发布管理）：面向最终用户的缓存发布

可以通过WEB实现一套完整的CMS管理系统，用于对PC网站和移动端浏览内容的增、删、改、查等操作，通过对模板内容的修改即可改变网页展示内容，方便了网站管理人员的日常管理和操作。

二、One Page One Application

1.定义

One Page, One Application(后面缩写为OPOA，或者1P1A)， 含义很简单：一个页面就是一个应用。不再使用iframe, 页面提交不能再使用submit方式。网页中发生的操作和交互都在当 前页面进行。

在

众多的基于Web的MIS系统中，没有人关心页面的组织形式；大多数稍微复杂的MIS系统，都采用分祯(Frame)的方式来组织页面，这样，在进行业务

操作的时候，url的变化表现在一个框架页面内，从浏览器的地址看起来，只有一个地址；更有甚者，一些应用干脆弹出一个去掉了浏览器菜单、工具条、地址

栏、状态栏的窗口（比如招商银行、民生银行的网上银行系统），连地址都看不见。因此，一个页面就是一个应用，从用户的角度来说，对于操作型系统，是一种非

常自然的体现。用户无需了解每一个具体的操作对应的地址是什么。

这种设计背后的含义实际是：是希

望由程序来控制用户的行为，还是反过来。在操作型系统中，每一步的操作往往被业务含义严格定义，无论是应用的设计者，还是其使用者，都希望在一种受控的状

况下来进行操作。例如，一个审批动作，用户更希望是通过一个按钮来触发，而不是访问类似于/approve.action?itemid=123的方式。

这样的好处是：很多东西，例如：JS，CSS，HEAD等整个系统都只需加载一次。加快响应速度。客户体验也有所提高，不再弹出窗口，不再整个页面进行刷新。

2.场景（内容管理系统更倾向明确的URL定位页面）

显然，OPOA的设计只能针对那些对URL不敏感的系统，或者说操作型系统。绝大多数MIS系统都属于这一范畴，Email系统也是这一范

畴，其他领域，如监控系统，聊天室等都可以采用这种思路。反面的例子是，对于内容型系统，如新闻系统，Blog系统，论坛系统，用户更希望能够通过一个明

确的URL来定位页面内容，搜索引擎也喜欢这种地址。这种应用需要的是一个合理，易懂，明确的地址。

3.设计与实现

注意到上述的OPOA地实现只是对用户而言，看起来好像是一个页面一样，但实际上还是有众多的action， page在后面工作。

三、react的技术准备

**1.react的起源**

React 起源于 Facebook 的内部项目，意在解决随时间数据不断变化的大规模应用程序开发，react可以表现出应用程序在任何时间点的样子，底层数据改变时，react的虚拟DOM机制会自动重新渲染，更新界面。

**2.对react的认识**

React不是一个完整的MVC框架，最多可以认为是MVC中的V（View），甚至React并不非常认可MVC开发模式；

React的服务器端Render能力只能算是一个锦上添花的功能，并不是其核心出发点，事实上React官方站点几乎没有提及其在服务器端的应用；

React的虚拟DOM原理：在Web开发中，我们总需要将变化的数据实时反应到UI上，这时就需要对DOM进行操作。而**复杂或频繁的DOM操作通常是性能瓶颈产生的原因**（如何进行高性能的复杂DOM操作通常是衡量一个前端开发人员技能的重要指标）。React为此引入了**虚拟DOM（Virtual DOM）**的 机制：在浏览器端用Javascript实现了一套DOM API。基于React进行开发时所有的DOM构造都是通过虚拟DOM进行，每当数据变化时，React都会重新构建整个DOM树，然后React将当前 整个DOM树和上一次的DOM树进行对比，得到DOM结构的区别，然后仅仅将需要变化的部分进行实际的浏览器DOM更新。而且React能够批处理虚拟 DOM的刷新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如你连续的先将节点内容从A变成B，然后又从B变成A，React会认为UI不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟DOM树，但是因为虚拟DOM是内存数据，性能是极高的，而对实际DOM进行操作的仅仅是 Diff部分，因而能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的DOM元素，而只需要 关心在任意一个数据状态下，整个界面是如何Render的。

**jsx语法：**HTML 语言直接写在 JavaScript 语言之中，不加任何引号，这就是 JSX 的语法，它允许 HTML 与 JavaScript 的混写。React不是一个新的模板语言，JSX只是一个表象，没有JSX的React也能工作。jsx语法与javascript并不兼容，需要通过babel-loader来解析。

**组件化：构建可组合的组件（组件：对数据和方法的简单封装，**封装起来的具有独立功能的UI部件**），是代码复用、测试和关注分离。**React推荐以组件的方式去重新思考UI构成，将UI上每一个功能相对独立的模块定义 成组件，然后将小的组件通过组合或者嵌套的方式构成大的组件，最终完成整体UI的构建。MVC的思想让你做到视图-数据-控制器的分离，那么组件化的思考方式则是带来了UI功能模块之间的分离。

React认为一个组件应该具有如下特征：

**（1）可组合（Composeable）**：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件；

**（2）可重用（Reusable）**：每个组件都是具有独立功能的，它可以被使用在多个UI场景；

**（3）可维护（Maintainable）**：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；

3.**ReactJS组件**

**组件属性**

前面说了，ReactJS是基于组件化的开发，下面我们开始来学习

ReactJS里面的组件，React 允许将代码封装成组件（component），然后像插入普通 HTML

标签一样，在网页中插入这个组件。React.createClass 方法就用于生成一个组件类。

1）获取属性的值用的是this.props.属性名

2）创建的组件名称首字母必须大写。

3）为元素添加css的class时，要用className。

4）组件的style属性的设置方式也值得注意，要写成style={{width: this.state.witdh}}。

**组件状态**

组 件免不了要与用户互动，React 的一大创新，就是将组件看成是一个状态机，一开始有一个初始状态，然后用户互动，导致状态变化，从而触发重新渲染 UI 。**当用户点击组件，导致状态变化，this.setState 方法就修改状态值，每次修改以后，自动调用 this.render 方法，再次渲染组件。**

**组件的生命周期**

组件的生命周期分成三个状态：

Mounting：已插入真实 DOM

Updating：正在被重新渲染

Unmounting：已移出真实 DOM

React 为每个状态都提供了两种处理函数，will 函数在进入状态之前调用，did 函数在进入状态之后调用，三种状态共计五种处理函数。

componentWillMount()

componentDidMount()

componentWillUpdate(object nextProps, object nextState)

componentDidUpdate(object prevProps, object prevState)

componentWillUnmount()

**组件的嵌套**

React是基于组件化的开发，那么组件化开发最大的优点是什么？毫无疑问，当然是复用，下面我们来看看React中到底是如何实现组件的复用的，这里我们还写一个例子来说吧，代码如下：

这里我们创建了一个Search组件，然后又创建了一个Page组件，然后我们在Page组件中调用Search组件，并且调用了两次，这里我们通过属性searchType传入值。

**4.ReactJS小结**

关于ReactJS今天就先学习到这里了，下面来总结一下，主要有以下几点：

1、ReactJs是基于组件化的开发，所以最终你的页面应该是由若干个小组件组成的大组件。

2、可以通过属性，将值传递到组件内部，同理也可以通过属性将内部的结果传递到父级组件(留给大家研究)；要对某些值的变化做DOM操作的，要把这些值放到state中。

3、

为组件添加外部css样式时，类名应该写成className而不是class;添加内部样式时，应该是style={{opacity:

this.state.opacity}}而不是style="opacity:{this.state.opacity};"。

4、组件名称首字母必须大写。

5、变量名用{}包裹，且不能加双引号。

react相关资源：[https://github.com/simongfxu/simongfxu.github.com/issues/21](https://link.jianshu.com?t=https:/github.com/simongfxu/simongfxu.github.com/issues/21" \t "_blank)

**5.ReactJS+Flux开发模式**

react不是一个完整的jsMVC开发框架，仅仅可以充当View层。所有的react组件输出的只是HTML，不能单独使用react创建一个完整的动态应用。react不包含：

事件系统（除了原生DOM事件）

Ajax功能

数据层

这种开发模式下需要的几个要素：

Flux:用于管理数据流，描述单向数据流，只有从模型到视图的数据流动，定义组件之间，组件和数据模型之间如何通信。且包含某些特定的事件和监听器，dispacher和某个js事件库。

dispacher本质是一个事件系统，全局的分发器，负责广播和注册callback，有且只有一个全局的dispacher。

Store:可以存在多个Store存储不同数据类型，用于响应dispacher事件，并且进行响应的数据处理，是应用中唯一知道如何更新数据的地方，只有store可以注册callback。View永远不应该调用App.Dispatcher.register，Dispatcher的存在就是为了将消息从View传递到Store，然后Store触发change事件，通过第三方js事件传递到View。

参考资料

[https://zhuanlan.zhihu.com/p/19900243](https://link.jianshu.com?t=https:/zhuanlan.zhihu.com/p/19900243)

**6.ReactJS+Redux开发模式**

Redux 是 JavaScript 状态容器，提供可预测化的状态管理。 可以让你构建一致化的应用，运行于不同的环境（客户端、服务器、原生应用），并且易于测试。Redux 除了和[React](https://link.jianshu.com?t=https:/facebook.github.io/react/" \t "_blank)一起用外，还支持其它界面库。Redux 由[Flux](https://link.jianshu.com?t=http:/facebook.github.io/flux/" \t "_blank)演变而来，但受[Elm](https://link.jianshu.com?t=http:/elm-lang.org/guide/architecture" \t "_blank)的启发，避开了 Flux 的复杂性。

**要点**

应用中所有的 state 都以一个对象树的形式储存在一个单一的*store*中。

惟一改变 state 的办法是触发*action*，一个描述发生什么的对象。

为了描述 action 如何改变 state 树，需要编写*reducers*。

action：应该把要做的修改变成一个普通对象，这个对象被叫做*action*，而不是直接修改 state，action中定义action纯函数，所有的逻辑和数据处理在action函数中进行，包括数据接口请求，export特定的action事件名称，action对象中，type为约定的事件名，其他参数可以自定义。

*reducer：*编写专门的函数来决定每个 action 如何改变应用的 state，这个函数被叫做*reducer*。*import约定好的action事件名称，根据不同的action.type,*然后返回不同的新state对象，数据均来自action对象。

相比

Flux，只需要注意一个重要的区别。Redux 没有 Dispatcher 且不支持多个 store。相反，只有一个单一的 store

和一个根级的 reduce 函数（reducer）。随着应用不断变大，你应该把根级的 reducer 拆成多个小的

reducers，分别独立地操作 state 树的不同部分，而不是添加新的 stores。这就像一个 React

应用只有一个根级的组件，这个根组件又由很多小组件构成。

三大原则

Redux 可以用这三个基本原则来描述：

单一数据源

**整个应用的[state](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/Glossary.html" \l "state" \t "_blank)被储存在一棵 object tree 中，并且这个 object tree 只存在于唯一一个[store](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/Glossary.html" \l "store" \t "_blank)中。**

State 是只读的

**惟一改变 state 的方法就是触发[action](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/Glossary.html" \l "action" \t "_blank)，action 是一个用于描述已发生事件的普通对象。使用纯函数来执行修改**

**为了描述 action 如何改变 state tree ，你需要编写[reducers](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/Glossary.html" \l "reducer" \t "_blank)。**

Reducer

只是一些纯函数，它接收先前的 state 和 action，并返回新的 state。刚开始你可以只有一个

reducer，随着应用变大，你可以把它拆成多个小的 reducers，分别独立地操作 state tree 的不同部分，因为 reducer

只是函数，你可以控制它们被调用的顺序，传入附加数据，甚至编写可复用的 reducer 来处理一些通用任务

与Flux比较

Redux

的灵感来源于 Flux 的几个重要特性。和 Flux 一样，Redux 规定，将模型的更新逻辑全部集中于一个特定的层（Flux 里的

store，Redux 里的 reducer）。Flux 和 Redux 都不允许程序直接修改数据，而是用一个叫作 “action”

的普通对象来对更改进行描述。

而不同于 Flux ，**Redux 并没有 dispatcher 的概念**。原因是它依赖纯函数来替代事件处理器。纯函数构建简单，也不需额外的实体来管理它们。你可以将这点看作这两个框架的差异或细节实现，取决于你怎么看 Flux。Flux 常常[被表述为(state, action) => state](https://link.jianshu.com?t=https:/speakerdeck.com/jmorrell/jsconf-uy-flux-those-who-forget-the-past-dot-dot-dot" \t "_blank)。从这个意义上说，Redux 无疑是 Flux 架构的实现，且得益于纯函数而更为简单。

和 Flux 的另一个重要区别，是**Redux 设想你永远不会变动你的数据**。你可以很好地使用普通对象和数组来管理 state ，而不是在多个 reducer 里变动数据。正确且简便的方式是，你应该在 reducer 中返回一个新对象来更新 state， 同时配合[object spread 运算符提案](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/recipes/UsingObjectSpreadOperator.html)或一些库，如[Immutable](https://link.jianshu.com?t=https:/facebook.github.io/immutable-js)。

虽然出于性能方面的考虑，[写不纯的 reducer](https://link.jianshu.com?t=https:/github.com/gaearon/redux/issues/328#issuecomment-125035516)来变动数据在技术上是**可行**的，但我们并不鼓励这么做。不纯的 reducer 会使一些开发特性，如时间旅行、记录/回放或热加载不可实现。此外，在大部分实际应用中，这种数据不可变动的特性并不会带来性能问题，就像[Om](https://link.jianshu.com?t=https:/github.com/omcljs/om" \t "_blank)所表现的，即使对象分配失败，仍可以防止昂贵的重渲染和重计算。而得益于 reducer 的纯度，应用内的变化更是一目了然。

Action

首先，让我们来给 action 下个定义。

**Action**是把数据从应用（译者注：这里之所以不叫 view 是因为这些数据有可能是服务器响应，用户输入或其它非 view 的数据 ）传到 store 的有效载荷。它是 store 数据的**唯一**来源。一般来说你会通过[store.dispatch()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/Store.html#dispatch)将 action 传到 store。

Action 本质上是 JavaScript 普通对象。我们约定，action 内必须使用一个字符串类型的type字段来表示将要执行的动作。多数情况下，type会被定义成字符串常量。当应用规模越来越大时，建议使用单独的模块或文件来存放 action。

除了type字段外，action 对象的结构完全由你自己决定。参照[Flux 标准 Action](https://link.jianshu.com?t=https:/github.com/acdlite/flux-standard-action" \t "_blank)获取关于如何构造 action 的建议。

这时，我们还需要再添加一个 action type 来表示用户完成任务的动作。因为数据是存放在数组中的，所以我们通过下标index来引用特定的任务。而实际项目中一般会在新建数据的时候生成唯一的 ID 作为数据的引用标识。

**我们应该尽量减少在 action 中传递的数据**。比如上面的例子，传递index就比把整个任务对象传过去要好。

Action 创建函数

**Action 创建函数**就是生成 action 的方法。“action” 和 “action 创建函数” 这两个概念很容易混在一起，使用时最好注意区分。这样做将使 action 创建函数更容易被移植和测试。

Reducer

[Action](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Actions.html)只是描述了**有事情发生了**这一事实，并没有指明应用如何更新 state。而这正是 reducer 要做的事情。

设计 State 结构

在 Redux 应用中，所有的 state 都被保存在一个单一对象中。建议在写代码前先想一下这个对象的结构。如何才能以最简的形式把应用的 state 用对象描述出来？

以 todo 应用为例，需要保存两种不同的数据：

当前选中的任务过滤条件；

完整的任务列表。

通常，这个 state 树还需要存放其它一些数据，以及一些 UI 相关的 state。这样做没问题，但尽量把这些数据与 UI 相关的 state 分开。

Action 处理

现在我们已经确定了 state 对象的结构，就可以开始开发 reducer。reducer 就是一个纯函数，接收旧的 state 和 action，返回新的 state。

之所以称作 reducer 是因为它将被传递给[Array.prototype.reduce(reducer, ?initialValue)](https://link.jianshu.com?t=https:/developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce" \t "_blank)方法。保持 reducer 纯净非常重要。**永远不要**在 reducer 里做这些操作：

修改传入参数；

执行有副作用的操作，如 API 请求和路由跳转；

调用非纯函数，如Date.now()或Math.random()。

现在只需要谨记 reducer 一定要保持纯净。**只要传入参数相同，返回计算得到的下一个 state 就一定相同。没有特殊情况、没有副作用，没有 API 请求、没有变量修改，单纯执行计算。**

明白了这些之后，就可以开始编写 reducer，并让它来处理之前定义过的[action](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Actions.html" \t "_blank)。

我们将以指定 state 的初始状态作为开始。Redux 首次执行时，state 为undefined，此时我们可借机设置并返回应用的初始 state。

注意:

**不要修改state。**使用[Object.assign()](https://link.jianshu.com?t=https:/developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)新建了一个副本。不能这样使用Object.assign(state, { visibilityFilter: action.filter })，因为它会改变第一个参数的值。你**必须**把第一个参数设置为空对象。你也可以开启对ES7提案[对象展开运算符](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/recipes/UsingObjectSpreadOperator.html" \t "_blank)的支持, 从而使用{ ...state, ...newState }达到相同的目的。

**在default情况下返回旧的state。**遇到未知的 action 时，一定要返回旧的state。

Object.assign须知

[Object.assign()](https://link.jianshu.com?t=https:/developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)是 ES6 特性，但多数浏览器并不支持。你要么使用 polyfill，[Babel 插件](https://link.jianshu.com?t=https:/github.com/babel-plugins/babel-plugin-object-assign" \t "_blank)，或者使用其它库如[\_.assign()](https://link.jianshu.com?t=https:/lodash.com/docs#assign)提供的帮助方法。

switch和样板代码须知

switch语句并不是严格意义上的样板代码。Flux 中真实的样板代码是概念性的：更新必须要发送、Store 必须要注册到 Dispatcher、Store 必须是对象（开发同构应用时变得非常复杂）。为了解决这些问题，Redux 放弃了 event emitters（事件发送器），转而使用纯 reducer。

很不幸到现在为止，还有很多人存在一个误区：根据文档中是否使用switch来决定是否使用它。如果你不喜欢switch，完全可以自定义一个createReducer函数来接收一个事件处理函数列表，参照["减少样板代码"](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/recipes/ReducingBoilerplate.html" \l "reducers" \t "_blank)。

**注意每个 reducer 只负责管理全局 state 中它负责的一部分。每个 reducer 的state参数都不同，分别对应它管理的那部分 state 数据。**

现在看过起来好多了！随着应用的膨胀，我们还可以将拆分后的 reducer 放到不同的文件中, 以保持其独立性并用于专门处理不同的数据域。

Redux 提供了[combineReducers()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/combineReducers.html" \t "_blank)工具类，[combineReducers()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/combineReducers.html" \t "_blank)所做的只是生成一个函数，这个函数来调用你的一系列 reducer，每个 reducer**根据它们的 key 来筛选出 state 中的一部分数据并处理**，然后这个生成的函数再将所有 reducer 的结果合并成一个大的对象。

Store

在前面的章节中，我们学会了使用[action](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Actions.html" \t "_blank)来描述“发生了什么”，和使用[reducers](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Reducers.html" \t "_blank)来根据 action 更新 state 的用法。

**Store**就是把它们联系到一起的对象。Store 有以下职责：

维持应用的 state；

提供[getState()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/Store.html#getState)方法获取 state；

提供[dispatch(action)](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/Store.html#dispatch)方法更新 state；

通过[subscribe(listener)](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/Store.html#subscribe)注册监听器;

通过[subscribe(listener)](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/Store.html" \l "subscribe" \t "_blank)返回的函数注销监听器。

再次强调一下**Redux 应用只有一个单一的 store**。当需要拆分数据处理逻辑时，你应该使用[reducer 组合](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Reducers.html" \l "splitting-reducers" \t "_blank)而不是创建多个 store。

根据已有的 reducer 来创建 store 是非常容易的。在[前一个章节](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/basics/Reducers.html" \t "_blank)中，我们使用[combineReducers()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/combineReducers.html)将多个 reducer 合并成为一个。现在我们将其导入，并传递[createStore()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/createStore.html" \t "_blank)。

[createStore()](https://link.jianshu.com?t=http:/cn.redux.js.org/docs/api/createStore.html)的第二个参数是可选的, 用于设置 state 初始状态。这对开发同构应用时非常有用，服务器端 redux 应用的 state 结构可以与客户端保持一致, 那么客户端可以将从网络接收到的服务端 state 直接用于本地数据初始化。

**7.Webpack使用**

**什么是Webpack？**

事实上它是一个打包工具，而不是像RequireJS或SeaJS这样的模块加载器，通过使用Webpack，能够像Node.js一样处理依赖关系，然后解析出模块之间的依赖，将代码打包

[http://webpack.github.io/](https://link.jianshu.com?t=http:/webpack.github.io/)

[https://fakefish.github.io/react-webpack-cookbook/Getting-started.html](https://link.jianshu.com?t=https:/fakefish.github.io/react-webpack-cookbook/Getting-started.html)

[https://segmentfault.com/a/1190000002767365](https://link.jianshu.com?t=https:/segmentfault.com/a/1190000002767365)

参考资料：

ReactJS官网地址：[http://facebook.github.io/react/](https://link.jianshu.com?t=http:/facebook.github.io/react/" \t "_blank)

Github地址：[https://github.com/facebook/react](https://link.jianshu.com?t=https:/github.com/facebook/react)

[http://www.ruanyifeng.com/blog/2015/03/react.html](https://link.jianshu.com?t=http:/www.ruanyifeng.com/blog/2015/03/react.html)

[https://segmentfault.com/a/1190000002767365](https://link.jianshu.com?t=https:/segmentfault.com/a/1190000002767365)

[https://fakefish.github.io/react-webpack-cookbook/Getting-started.html](https://link.jianshu.com?t=https:/fakefish.github.io/react-webpack-cookbook/Getting-started.html)