



谢成

## 第一节

# ReactJs简介

## 1-1 ReactJs的起源

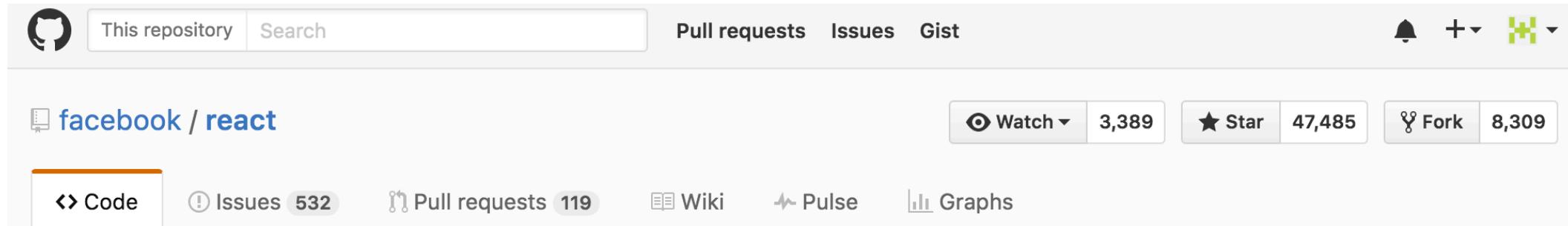


## 1-1 ReactJs的起源

- React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己写一套，用来架设 Instagram 的网站。做出来以后，发现这套东西很好用，就在 2013 年 5 月开源了。由于 React 的设计思想极其独特，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来 Web 开发的主流工具。这个项目本身也越滚越大，从最早的 UI 引擎变成了一整套前后端通吃的 Web App 解决方案。衍生的 React Native 项目，目标更是宏伟，希望用写 Web App 的方式去写 Native App。如果能够实现，整个互联网行业都会被颠覆，因为同一组人只需要写一次 UI，就能同时运行在服务器、浏览器和手机
- ReactJS 官网地址：<http://facebook.github.io/react/>
- Github 地址：<https://github.com/facebook/react>
- ReactJS 中文文档：<http://reactjs.cn/>

## 1-2 ReactJs的现状

- 国外使用ReactJS的公司：Facebook, Flipboard, Airbnb, BBC, GitHub, Instagram, Reddit, Uber, WhatsApp, Yahoo
- 国内使用ReactJS的公司：支付宝，淘宝，大搜车，Teambition，豆瓣，豌豆荚



A declarative, efficient, and flexible JavaScript library for building user interfaces. <https://facebook.github.io/react/>

## 1-3 ReactJs

- 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实现在服务器端的应用；
- 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。



React

[文档](#) [中文社区](#) [下载](#) [GITHUB](#) [关于我们](#)

# React



用于构建用户界面的JAVASCRIPT库

[快速入门](#)[下载 React v0.14.7](#)

## 仅仅是UI

许多人使用React作为MVC架构的V层。尽管React并没有假设过你的其余技术栈，但它仍可以作为一个小特征轻易地在已有项目中使用

## 虚拟DOM

React为了更高超的性能而使用虚拟DOM作为其不同的实现。它同时也可由服务端Node.js渲染 - 而不需要过重的浏览器DOM支持

## 数据流

React实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

## 1-4 ReactJs原理

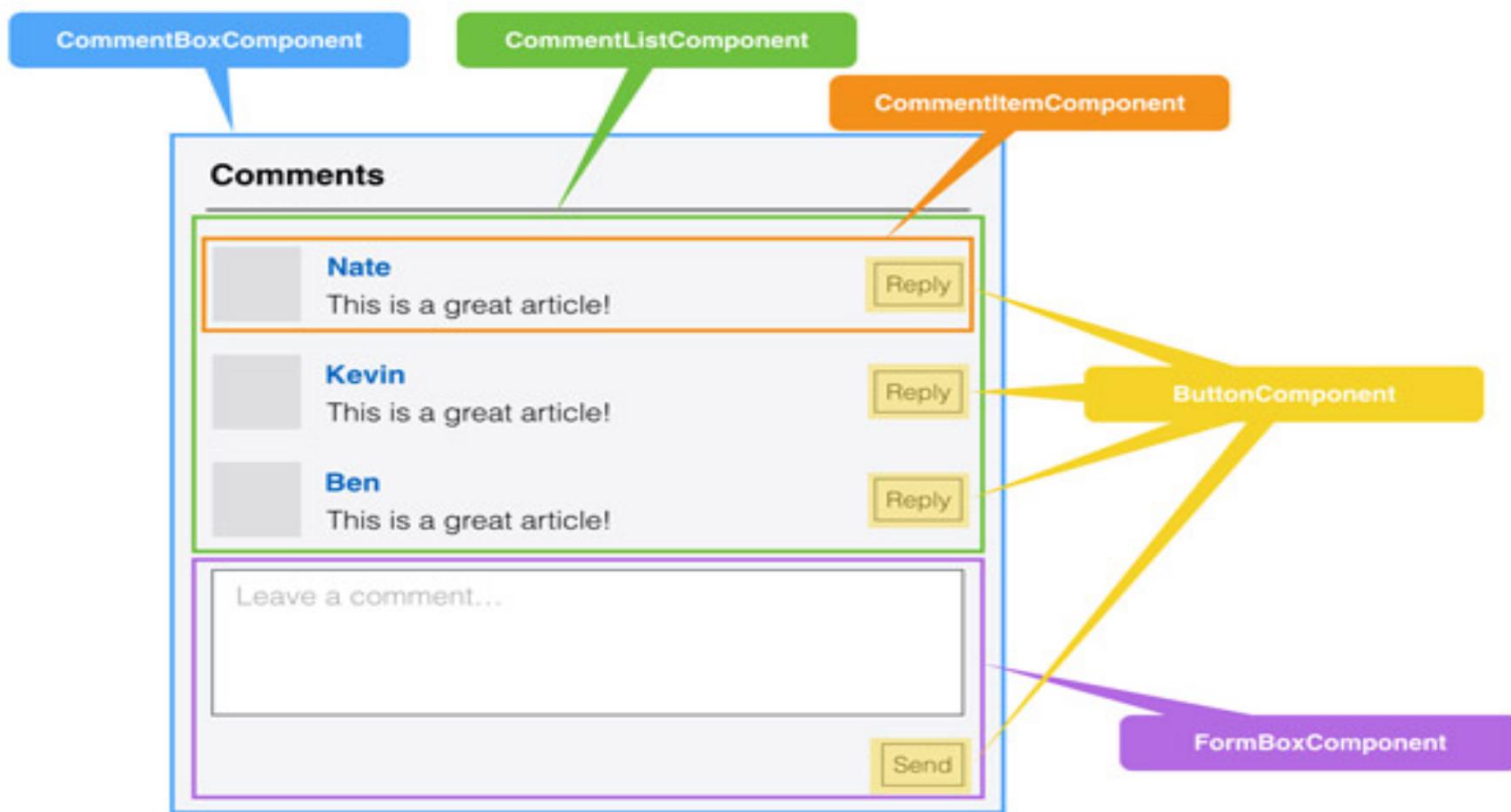
在Web开发中，我们总需要将变化的数据实时反应到UI上，这时就需要对DOM进行操作。而复杂或频繁的DOM操作通常是性能瓶颈产生的原因（如何进行高性能的复杂DOM操作通常是衡量一个前端开发人员技能的重要指标）。React为此引入了**虚拟DOM（Virtual DOM）**的机制。基于React进行开发时所有的DOM构造都是通过虚拟DOM进行，每当数据变化时，React都会重新构建整个DOM树，然后React将当前整个DOM树和上一次的DOM树进行对比，得到DOM结构的区别，然后仅仅将需要变化的部分进行实际的浏览器DOM更新。而且React能够批处理虚拟DOM的刷新，在一个事件循环（Event Loop）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。

## 1-5 组件化

- 虚拟DOM(virtual-dom)不仅带来了简单的UI开发逻辑，同时也带来了组件化开发的思想，所谓组件，即封装起来的具有独立功能的UI部件。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。例如，Facebook的instagram.com整站都采用了React来开发，整个页面就是一个大的组件，其中包含了嵌套的大量其它组件，大家有兴趣可以看下它背后的代码。
- 如果说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。我们通过一个典型的Blog评论界面来看MVC和组件化开发思路的区别。
- 对于MVC开发模式来说，开发者将三者定义成不同的类，实现了表现，数据，控制的分离。开发者更多的是从技术的角度来对UI进行拆分，实现松耦合。
- 对于React而言，则完全是一个新的思路，开发者从功能的角度出发，将UI分成不同的组件，每个组件都独立封装。

## 1-5 组件化

- ▶ 在React中，你按照界面模块自然划分的方式来组织和编写你的代码，对于评论界面而言，整个UI是一个通过小组件构成的大组件，每个组件只关心自己部分的逻辑，彼此独立。



## 1-6 组件的特征和优点

- 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；

- 优点

- 提高代码复用率：组件将数据和逻辑封装，类似面向对象中类；
- 降低测试难度：组件高内聚低耦合，很容易对单个组件进行测试；
- 降低代码复杂度：直观的语法提高代码可读性。

## 1-7 angularJs 的组件

```
38
39 app.directive('messageBox',function(){
40     return {
41         restrict: 'E',
42         template:'<div> \
43             <h1 ng-click="alert()">你好世界! </h1> \
44             <p>每次都是hello world, 你丫烦不烦? </p> \
45             </div>',
46         link:function($scope){
47             $scope.alert = function(){
48                 alert('没事干嘛点我? ');
49             }
50         }
51     }
52 })
```

## 第二节

# ReactJs入门

- 2-1 Hello World
- 2-2 JSX: 是一种类 XML 语言，全称是 JavaScript XML 。 React 可以不使用 JSX 来编写组件，但是使用 JSX 可以让代码可读性更高、语义更清晰、对 React 元素进行抽象等等。
- 2-3 组件嵌套
- 2-4 state: 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
- 2-5 props: 由于 `this.props` 和 `this.state` 都用于描述组件的特性，可能会产生混淆。一个简单的区分方法是， `this.props` 表示那些一旦定义，就不再改变的特性，而 `this.state` 是会随着用户互动而产生变化的特性。
- 2-6 event

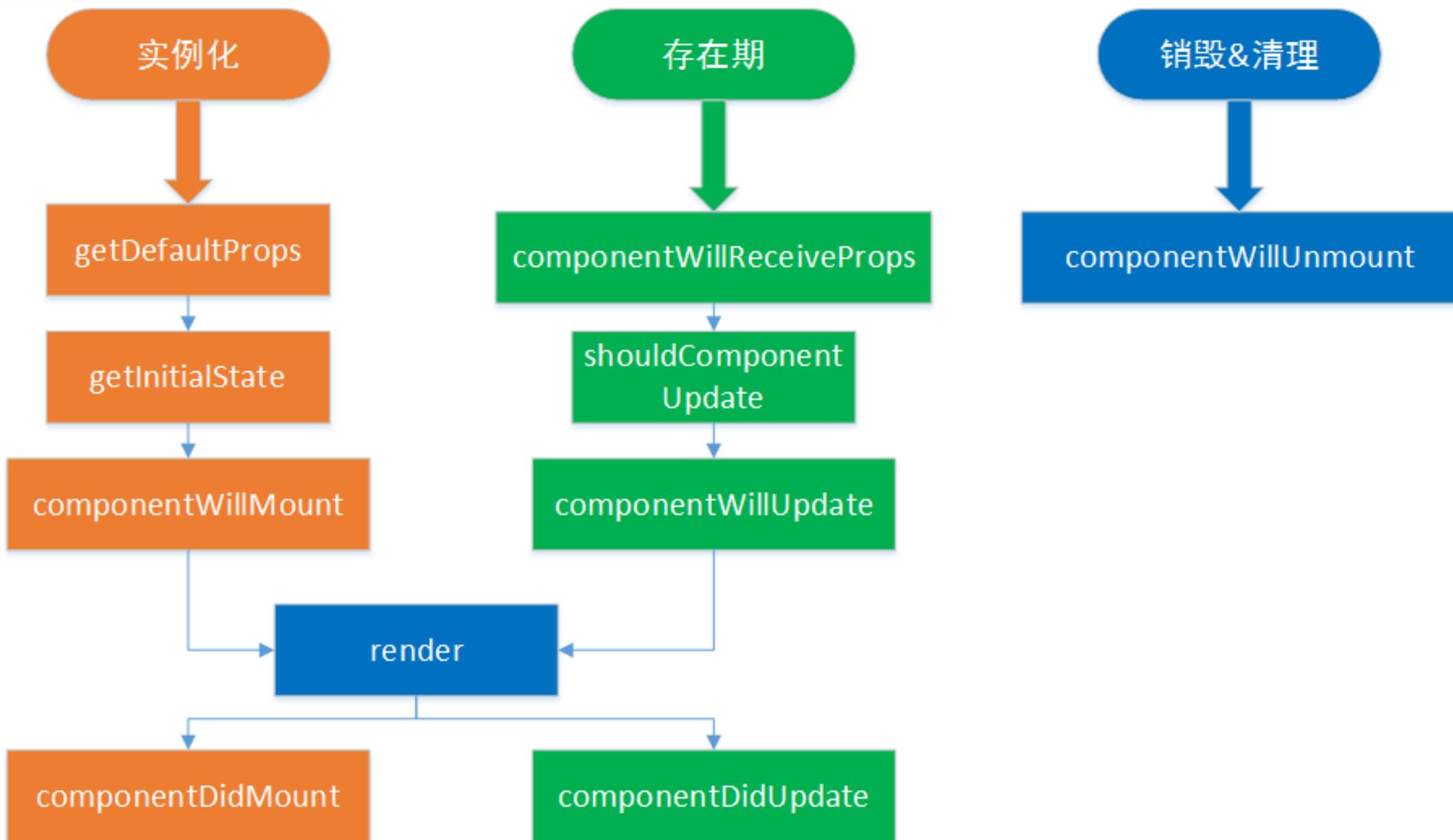
## 2-8 ref

- 组件并不是真实的 DOM 节点，而是存在于内存之中的一种数据结构，叫做虚拟 DOM（virtual DOM）。只有当它插入文档以后，才会变成真实的 DOM。根据 React 的设计，所有的 DOM 变动，都先在虚拟 DOM 上发生，然后再将实际发生变动的部分，反映在真实 DOM 上，这种算法叫做 DOM diff，它可以极大提高网页的性能表现。
- 但是，有时需要从组件获取真实 DOM 的节点，这时就要用到 ref 属性。
- 需要注意的是，由于 `this.refs.[refName]` 属性获取的是真实 DOM，所以必须等到虚拟 DOM 插入文档以后，才能使用这个属性，否则会报错。上面代码中，通过为组件指定 Click 事件的回调函数，确保了只有等到真实 DOM 发生 Click 事件之后，才会读取 `this.refs.[refName]` 属性。

## 2-9 组件生命周期

- 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()
- 此外，React 还提供两种特殊状态的处理函数。
  - componentWillReceiveProps(object nextProps): 已加载组件收到新的参数时调用
  - shouldComponentUpdate(object nextProps, object nextState): 组件判断是否重新渲染时调用
  - ReactDOM.unmountComponentAtNode(document.getElementById('app'));

## 2-9 组件生命周期



## 第三节

# ReactJs例子

## 第四节

# 常见问题FAQ

- reactjs定义组件使用什么方法?
- 什么是JSX?
- reactjs和vue有什么区别?
- state和props有什么区别?
- 组件之间如何传值?
- 组件的生命周期?

React 这样做:

```
<!DOCTYPE html>
<html>
  <head>
    <title>React Hello World</title>
    <script src="https://fb.me/react-15.0.0.js"></script>
    <script src="https://fb.me/react-dom-15.0.0.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
  </head>
  <body>
    <div id="greeting"></div>
    <script type="text/babel">
      var Greeting = React.createClass({
        render: function() {
          return (
            <p>Hello from React</p>
          )
        }
      });
      ReactDOM.render(
        <Greeting/>,
        document.getElementById('greeting')
      );
    </script>
  </body>
</html>
```

Vue 这样做:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Vue Hello World</title>
    <script src="https://vuejs.org/js/vue.min.js"></script>
  </head>
  <body>
    <div id="app">
      {{ message }}
    </div>
    <script>
      new Vue({
        el: '#app',
        data: {
          message: 'Hello from Vue'
        }
      });
    </script>
  </body>
</html>
```

## 双向数据绑定

Vue 这样做:

React 这样做:

```
// React.js (jsx)
```

```
var Message = React.createClass({
  getInitialState() {
    return {
      message: ''
    }
  },
  handleMessageChange(e) {
    this.setState({message: e.target.value});
  },
  render() {
    return (
      <div>
        <input type="text" onChange={this.handleMessageChange} />
        <span>{this.state.message}</span>
      </div>
    )
  }
});
```

```
// Vue.js (js)
```

```
var Message = new Vue({
  el: '#message',
  data: {
    message: ''
  }
});
```

```
<div id="message">
  <input type="text" v-model="message" />
  <span>{{message}}</span>
</div>
```

Vue.js 中的双向数据绑定在你使用了 v-model 时就会相当的简单。而在 React 中, 过程就比较漫长了。

## 迭代

## Vue这样做:

React这样做:

```
// React.js (.jsx)

var Iteration = React.createClass({
  getInitialState() {
    return {
      array: [1,2,3,4]
    }
  },
  render() {
    this.state.array.map( function(value) {
      return (
        <span>{value}</span>
      )
    });
  }
});

ReactDOM.render(<Iteration />, document.getElementById('array'));
```

// Vue.js (js)

```
var Iteration = new Vue({
  el: '#array',
  data: {
    array: [1,2,3,4]
  }
});
```

<!-- Vue.js (html) -->

```
<div id="array">
  <span v-for="value in array">{{value}}</span>
</div>
```



Thank you

谢

谢

观

看