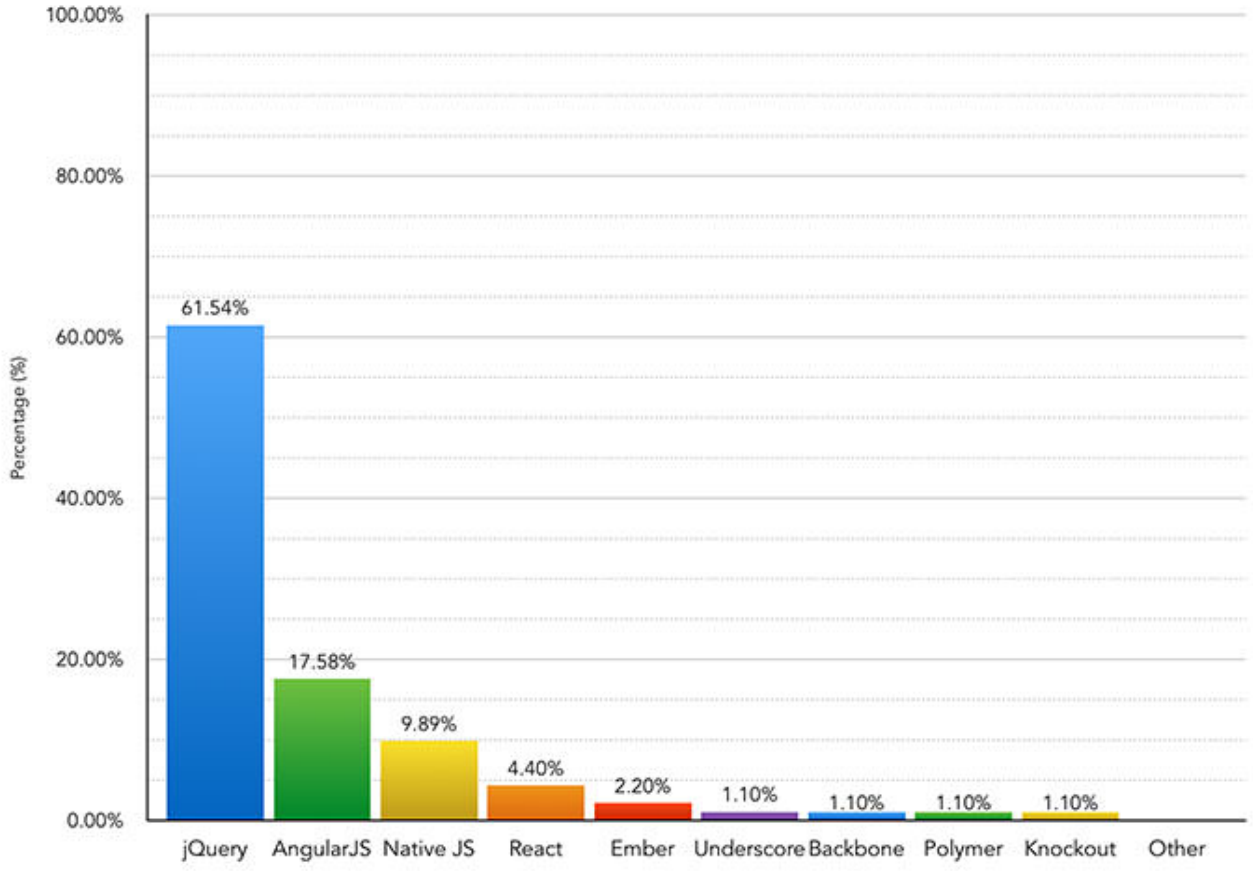
React技术学习--入门篇（一）

先看一张图



自2013年5月份facebook将React开源以来，截至目前已经在github上收获了超过3万个star。

衍生的React Native项目（andriod和ios版本）也在今年9月份完成发布。

1. 基本认识

React是一个用于构建用户界面的JavaScript库，而不是一个MVC框架，但可以使用React作为MVC架构的View层轻易地在已有项目中使用。

> 如果想参与angular好还是react好的讨论大战的话，请移步知乎

2. 为什么用React

> 高效DOM渲染

以前没有ajax技术的时候，web页面从服务端整体render出html送到浏览器端进行渲染，同样的，用户的一个改变页面的操作也会刷新整个页面来完成。直到有了ajax出现，实现页面局部刷新，带来的高效和分离让web开发者们惊叹不已。但随着而来的代价是，复杂的用户交互及展现需要通过大量的DOM操作来完成，这让页面的性能以及开发的效率又出现了新的瓶颈。

时至今日，谈到前端性能优化，减少DOM元素、减少reflow和repaint、编码过程中尽量减少DOM的查询等手段是大家耳熟能详的。而页面任何UI的变化都是通过整体刷新来完成的。幸运的是，React通过自己实现的DOM Diff算法，计算出虚拟页面当前版本和新版本之间的差异，最小化重绘，避免不必要的DOM操作，解决了这两个公认的前端性能瓶颈，实现高效DOM渲染。

- 我们知道，频繁的操作DOM所带来的性能消耗是很大的，而React之所以快，是因为它不直接操作DOM，而是引进虚拟DOM的实现来解决这个问题
- 对于页面的更新，React通过自己实现的DOM Diff算法来进行差异对比、差异更新，反映到页面上就是只重绘了更新的部分，从而提高渲染效率。

> 组件化

在业务开发中，遇到公共的模板部分，我们不得不将模板和规定的数据格式耦合在一起来实现组件。而在React中，我们可以使用JSX语法来封装组件，将组件的结构、数据逻辑甚至样式都聚合在一起，更加简单、明了、直观的定义组件。

有了组件化的实现，我们可以很直观的将一个复杂的页面分割成若干个独立组件，再将这些独立组件组合完成一个复杂的页面。这样既减少了逻辑复杂度，又实现了代码的重用。

React认为一个组件应该具有如下的特征：

- 可组合：一个组件可以和其他的组件一起使用或者可以直接嵌套在另一个组件内部，通过这样的组合方式，一个复杂的UI组件可以分拆成若干个简单的UI组件
- 可重用：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
- 可维护：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护

> 单向数据流

在React中，数据的流向是从父节点到子节点的单向流动，这样可以使组件简单并且容易把握，因为子节点是无状态的，只需要从父节点获取props渲染即可。这样带来的收益是，顶层组件的某个prop改变了，React就会向下递归遍历整棵组件树，重新渲染所有使用到了这个属性的组件。

单向数据流带来的几个重要的好处是：

- 相比之前的资源重组实现的组件，单向数据流可以很好的完成组件间的数据通信，否则的话，我们需要写一个事件机制来处理这个事情。
- 大家可能会问，这所倡导的单向流动，那相对MVC或是MVVM框架的双向数据绑定简直是弱爆了。那么这里需要理解的是，这里的单向，是循环流动的单向，数据是持续更新的。双向数据绑定是优秀便捷的实现，这个需要用实现的成本和业务场景来考量二者了。
- 对于单向数据流目前已经有很好的类库实现了，如flux reflux redux等。

3. 你需要持续关注这些

- React的版本

React还在持续的更新开发中，截至目前React的最新版是0.14.0版本，每一次的更新意味着API的改变亦或是包的拆解，关注版本的更新让你的代码和思想都跟上节奏。

- 学习资料

相比于之前的看不懂的官方文档，现在的中文论坛、文档、学习书籍慢慢完善起来了。可以有几个途径去获得相关的资料：

- github官方仓库
- React官网
- React中国官网
- 论坛

- 架构及周边生态
  - 架构
    - 和其他类库或是框架结合使用
    - 比如和backbone结合，弥补了backbone的薄弱的view层；
    - 和angular结合，让组件的渲染更快

  - 和flux相关类库结合
    - flux
    - reflux
    - redux

  - 生态
    - 使用webpack打包
    - 和ES6结合来封装你的组件
    - 测试相关

  - 浏览器兼容性等问题是你在使用后不得不考虑的问题。

React技术学习--基础篇（二）

> 特别提示：本教程的代码示例详见iUAP-FE/react

越是基础的东西，越是重要；越是原理的内容，越要去理清楚。

1. 下载React

> 有以下三种方式：

- npm下载react包

```
npm install react --save
```

- bower下载

```
bower install react --save
```

- 或者直接去官网下zip包

2. React版本和接口说明

提供的文件不一样

0.13版本

```
react.js
react-with-addons.js
JSXTransformer.js
```

0.14版本

```
react.js
react-dom.js
react-with-addons.js
```

React被拆分为**react**和**react-dom**两个

- react.js 是 React 的核心库

react包提供了一系列的API，以下列举几个常用的：

```
// 使用ES6的时候可以用这个API来定义一个组件
React.Component
// 创建一个组件类，并作出定义
React.createClass
// 创建并返回一个新的指定类型的 ReactElement
React.createElement
React.cloneElement
// 返回一个生成指定类型 ReactElements 的函数
React.createFactory
// 验证一个对象是否为ReactElement，返回boolean值
React.isValidElement
React.DOM
.....
```

- react-dom.js提供与 DOM 相关的功能，以下列举几个常用的：

react包提供了一系列与DOM相关的API

```
// 渲染一个 ReactElement 到 DOM 中，放在 container 指定的 DOM 元素下，返回一个到该组件的引用。
ReactDOM.render
// 从 DOM 中移除已经挂载的 React 组件，清除相应的事件处理器和 state
ReactDOM.unmountComponentAtNode
ReactDOM.findDOMNode
```

- 服务端渲染的几个 API 被独立出来，以下两个是常用的：

```
ReactDOMServer.renderToString
ReactDOMServer.renderToStaticMarkup
```

React.addons被拆分出若干个独立的包

- 说明下，这个文件是官方提供的已封装的一系列插件
- 在0.14版本将其中的插件封装成若干个独立的 package提供使用（至少五个，之前版本是直接在一个文件中引用）。

编译器优化

react-tools 及 **JSXTransformer.js** 已弃用

以前是采用JSXTransformer来解析JSX语法，现在是全面拥抱**Babel**（可以 **npm insttall babel -g** 安装**babel**进行**JSX**语法解析、或是加上**babel**提供的**browser.js**库进行解析）。

> 备注：如果没接触Babel的同学，请移步这里babeljs.io，Babel是一款强大的语言解析器，目前github上已经超过一万个star了，基于babel还可以自定义封装自己的解析器插件。

3. 运行代码

> 运行的两种方式

- 页面中加browser.js，script标签的type设置为text/babel(0.13版本为text/jsx)

```
<!DOCTYPE html>
<html>
  <head>
    <script src="../vendors/react/react.js"></script>
    <script src="../vendors/react/react-dom.js"></script>
    <!-- browser.js 的作用是将 JSX 语法转为 JavaScript 语法 -->
    <script src="../vendors/babel/browser.min.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <!-- JSX 语法，跟 JavaScript 不兼容。凡是使用 JSX 的地方，都要加上 type="text/babel" -->
    <script type="text/babel">
      var MyComponent = React.createClass({
          render: function (){
            return (
              <h1 className="header">我的第一个组件</h1>
            )
          }
      });
      ReactDOM.render(<MyComponent />, document.getElementById('example'));
    </script>
  </body>
</html>
```

- 页面中直接运行babel解析jsx的文件。

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
      <meta charset="UTF-8">
      <title>被解析的JSX</title>
      <script src="../vendors/react/react.js"></script>
      <script src="../vendors/react/react-dom.js"></script>
  </head>
  <body>
      <div id="example"></div>
      <script src="../build/jsx_compile.js"></script>
  </body>
</html>
```

4. JSX语法

Talk is cheap, Show me the code.

直接上一段稍微复杂些的JSX代码

- demo1：jsx_demo1.html

```
var MyList = React.createClass({
  render: function() {
    return (
      <ul>
        {
          /* 遍历this.props.children */
          this.props.children.map(function (child) {
            return <li>{child}</li>
          })
        }
      </ul>
    );
  }
});
ReactDOM.render(
  <MyList>
    <a href="https://www.facebook.com/">https://www.facebook.com/</a>
    <a href="https://twitter.com/">https://twitter.com/</a>
  </MyList>,
  document.getElementById('example')
);
```

- demo2：simple_jsx.html

```
var MyData = ['React', 'is', 'awesome'],
    MyStyles = {
      color: "#333",
      fontSize: "40px",
      fontWeight: "bold"
    };

ReactDOM.render(
  <div style={MyStyles}>
  {
    MyData.map(function (name) {
      return <span>{name} </span>
    })
  }
  </div>,
  document.getElementById('example')
);
```

> 使用JSX语法来封装组件有什么好处

- 熟悉的代码
- 更加语义化
- 更加抽象且直观

几个注意点

- render的方法中return的顶级元素只能是一个
- 如果要定义样式的时候，不能这样去写

```
// 不要出现类似的错误，style="opacity:{this.state.opacity};"
```

- 使用 className 和 htmlFor 来替代对应的class 和 for

5. 数据流

三个维度来看待**React**中数据流

> 在React中数据的流向是单向的，即从父节点流向子节点，这样就更方便组件的渲染（子组件只需要从父组件获取props渲染即可）
>
> 组件内部有自己的状态，这些状态只能组件内部修改，保持独立性
>
> React组件本身很简单，可以把它看成就是一个函数，而这个函数有两个传参，props和state，调用这个函数后悔返回一个虚拟的DOM。

理解组件内部的数据流向

demo: jsx_compile.html

```
/*
 * example
 */
var Counter = React.createClass({
  // 相当于是规范化的接口文档
  propTypes: {
    name: React.PropTypes.string.isRequired,
  },
  // 定义初始化的state
  getInitialState: function () {
    return { clickCount: 0 };
  },
  // 定义一个处理点击的回调方法
  handleClick: function () {
    this.setState(function(state) {
      return {clickCount: state.clickCount + 1};
    });
  },
  render: function () {
    return (<h2 onClick={this.handleClick}>点我点我! <br />被戳次数: {this.state.clickCount}</h2>);
  }
});

ReactDOM.render(
  <Counter name="myCounter" />,
  document.getElementById('example')
);
```

说明：

- state

我们可以将组件看成是一个状态机，每一个组件都有自己的state，改变组件可以使用 `setState` 或是 `replaceState` ，千万不要这样类似这样写 `this.state.name = ''` 。

- PropTypes

这是验证props的方式，类似于约定了一个接口文档。

- props

通过props，可以把任意类型的数据传递给组件

- getDefaultProps和getInitialState

分别是定义初始化props和state值的两个钩子函数，不一样的是，在组件的生命周期中，前者只会执行一次，具体下一部分细说。

理解父组件和子组件之间的单向数据流动

父组件通过props来向子组件传递数据

组件间的通信

> 后面进阶部分结合flux思想来进行分享

理解**state**和**props**

虽然state和prop都是存储数据的，但是要区分二者的区别：

- state存放的是流动的，变化的组件数据，而且，**state**只存在于组件的内部
- 把props当成是组件的数据源，一般用来存放组件初始后不变的数据和属性

需要提醒的是：

- 不要将props的数据复制到state中去
- 不要使用setProps改变组件的属性

- 要慎用replaceState

二者的结合则可完成组件的单向数据流动

6. 组件的全生命周期和对应的那些钩子函数

组件生命周期的设计

React为每个组件都提供了简洁的生命周期API，去响应组件在不同阶段（创建时，存在时，销毁时）执行相应的操作，更精确的管理每一个组件。

- 组件在高内聚的同时，往往需要暴露一些接口供外界调用，从而能够适应复杂的页面需求；
- 更精细的掌控对组件的管理，更强的性能管理。

钩子函数

什么是钩子函数，可以理解为在组件生命周期中某个确定的时间点执行的函数。以下是对钩子函数的总结：

- 实例化（渲染前）
  - getDefaultProps() 生命周期中只会执行一次
  - getInitialState()
  - componentWillmount()
  - render()

这意味着你可以在这个组件插入到DOM之前都可以调用这些API

- 组件存在期（渲染为真实的DOM）
  - componentDidMount()
  - shouldComponentUpdate()
  - componentWillUpdate()
  - componentWillUnmount()

- 销毁期
  - componentDidUnmount()

代码示例

> 结合以上知识点，来看一个基于React、jquery和bootstrap完成的一个简单的组件。

```
// 定义一个按钮组件
var BootstrapButton = React.createClass({
  render: function() {
    return (
      <a {...this.props}
        href="javascript:;"
        role="button"
        className={(this.props.className || '') + ' btn'} />
    );
  }
});
// 定义一个弹框组件
var BootstrapModal = React.createClass({
  // 节点插入到真实的DOM，使用jquery
  componentDidMount: function() {
    // 调用bootstrap插件
    $(this.refs.root).modal({backdrop: 'static', keyboard: false, show: false});
  },
  // 在组件销毁的时候，记得把之前绑定的方法给干掉
  componentWillUnmount: function() {
    $(this.refs.root).off('hidden', this.handleHidden);
  },
  close: function() {
    $(this.refs.root).modal('hide');
  },
  open: function() {
    $(this.refs.root).modal('show');
  },
  render: function() {
    var confirmButton = null;
    var cancelButton = null;

    if (this.props.confirm) {
      confirmButton = (
        <BootstrapButton
          onClick={this.handleConfirm}
          className="btn-primary">
          {this.props.confirm}
        </BootstrapButton>
      );
    }
    if (this.props.cancel) {
      cancelButton = (
        <BootstrapButton onClick={this.handleCancel} className="btn-default">
```

```
          {this.props.cancel}
        </BootstrapButton>
      );
    }

    return (
      <div className="modal fade" ref="root">
        <div className="modal-dialog">
          <div className="modal-content">
            <div className="modal-header">
              <button
                type="button"
                className="close"
                onClick={this.handleCancel}>
                &times;
              </button>
              <h3>{this.props.title}</h3>
            </div>
            <div className="modal-body">
              {this.props.children}
            </div>
            <div className="modal-footer">
              {cancelButton}
              {confirmButton}
            </div>
          </div>
        </div>
      </div>
    );
  },
  handleCancel: function() {
    if (this.props.onCancel) {
      this.props.onCancel();
    }
  },
  handleConfirm: function() {
    if (this.props.onConfirm) {
      this.props.onConfirm();
    }
  }
});

// 调用刚才咱们定义的两个组件，写咱们的业务组件
var Example = React.createClass({
  handleCancel: function() {
    if (confirm('亲，确定要取消么')) {
      this.refs.modal.close();
    }
```

```
  },
  render: function() {
    var modal = null;
    modal = (
      <BootstrapModal
        ref="modal"
        confirm="OK"
        cancel="Cancel"
        onCancel={this.handleCancel}
        onConfirm={this.closeModal}
        title="Hello, Bootstrap!">
          这是一个结合jQuery和Bootstrap而写的组件
      </BootstrapModal>
    );
    return (
      <div className="example">
        {modal}
        <BootstrapButton onClick={this.openModal} className="btn-default">
          Open modal
        </BootstrapButton>
      </div>
    );
  },
  openModal: function() {
    this.refs.modal.open();
  },
  closeModal: function() {
    this.refs.modal.close();
  }
});

ReactDOM.render(<Example />, document.getElementById('example'));
```

复合组件

多个简单的组件嵌套，可构成一个复杂的复合组件，从而完成复杂的交互逻辑，实现页面功能。

React技术学习--进阶篇（三）

> 技术进阶，学习表单事件、封装组件

DOM操作和事件处理

1. 动画

2. 服务端渲染

3. 开发工具

- browserify
- webpack

React技术学习--高级篇（四）

1. 测试

2. 架构