React

转自阮一峰 React 入门实例教程

demo

```
$ git clone git@github.com:ruanyf/react-demos.git
```

编译得到生产环境 `npm run build`

html 模板

```
<!DOCTYPE html>
<html>
  <head>
    <script src="../build/react.js"></script>
    <script src="../build/react-dom.js"></script>
    <script src="../build/browser.min.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script type="text/babel">
      // ** Our code goes here! **
    </script>
  </body>
</html>
```

最后一个`<script>`的type为`text/babel`，这是因为 React 独有的 JSX 语法，跟 JavaScript 不兼容。凡是使用 JSX 的地方，都要加上 `type="text/babel"`。

ReactDOM.render()

ReactDOM.render 是 React 的最基本方法，用于将模板转为 HTML 语言，并插入指定的 DOM 节点。

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('example')
);
```

JSX语法

遇到 HTML 标签（以 **<** 开头），就用 HTML 规则解析；遇到代码块（以 **{** 开头），就用 JavaScript 规则解析。

```
var names = ['Alice', 'Emily', 'Kate'];

ReactDOM.render(
  <div>
  {
    names.map(function (name) {
      return <div>Hello, {name}!</div>
    })
  }
  </div>,
  document.getElementById('example')
);
```

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

```
var HelloMessage = React.createClass({
  render: function() {
    return <h1>Hello {this.props.name}</h1>;
  }
});

ReactDOM.render(
// or
// React.createElement(HelloMessage, null)
  <HelloMessage name="John" />,
  document.getElementById('example')
);
```

上面代码中，变量 `HelloMessage` 就是一个组件类。模板插入 `<HelloMessage /> ` 时，会自动生成 `HelloMessage` 的一个实例（下文的“组件”都指组件类的实例）。所有组件类都必须有自己的 `render` 方法，用于输出组件。

注意，组件类的第一个字母必须大写，否则会报错，比如HelloMessage不能写成helloMessage。另外，组件类只能包含一个顶层标签，否

PropTypes

官方文档

组件的属性可以接受任意值，字符串、对象、函数等等都可以. 组件类的`PropTypes`属性，就是用来验证组件实例的属性是否符合要求

```
var MyTitle = React.createClass({
  propTypes: {
    title: React.PropTypes.string.isRequired,
```

```
  },

  render: function() {
     return <h1> {this.props.title} </h1>;
   }
});
```

此外，getDefaultProps 方法可以用来设置组件属性的默认

```
var MyTitle = React.createClass({
  getDefaultProps : function () {
    return {
      title : 'Hello World'
    };
  },

  render: function() {
     return <h1> {this.props.title} </h1>;
   }
});

ReactDOM.render(
  <MyTitle />,
  document.body
);
```

获取真实的DOM节点

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。

从组件获取真实 DOM 的节点，这时就要用到 ref 属性

```
var MyComponent = React.createClass({
  handleClick: function() {
    this.refs.myTextInput.focus();
  },
  render: function() {
    return (
      <div>
        <input type="text" ref="myTextInput" />
      <input type="button" value="Focus the text input" onClick={this.handleClick} />
      </div>
    );
  }
```

```
});

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

React组件支持很多事件，除了Click事件以外，还有`KeyDown`、`Copy`、`Scroll`等，完整的事件清单请查看官方文档。

this.state

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

```
var LikeButton = React.createClass({
  getInitialState: function() {
    return {liked: false};
  },
  handleClick: function(event) {
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'like' : 'haven\'t liked';
    return (
      <p onClick={this.handleClick}>
        You {text} this. Click to toggle.
      </p>
    );
  }
});

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

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。

表单

用户在表单填入的内容，属于用户跟组件的互动，所以不能用`this.props`读取

```
var Input = React.createClass({
  getInitialState: function() {
    return {value: 'Hello!'};
```

```
  },
  handleChange: function(event) {
    this.setState({value: event.target.value});
  },
  render: function () {
    var value = this.state.value;
    return (
      <div>
     <input type="text" value={value} onChange={this.handleChange} />
        <p>{value}</p>
      </div>
    );
  }
});

ReactDOM.render(<Input/>, document.body);
```

上面代码中，文本输入框的值，不能用 `this.props.value` 读取，而要定义一个 `onChange` 事件的回调函数，通过`event.target.value`读取用户输入的值。`textarea`元素、`select`元素、`radio`元素都属于这种情况，更

组件的生命周期

详细说明参考官方文档

组件的生命周期分成三个状态：

- Mounting：已插入真实 DOM
- Mounting：已插入真实 DOM
- 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。

1. componentWillMount()
2. componentDidMount()
3. componentWillUpdate(object nextProps, object nextState)
4. componentWillUpdate(object nextProps, object nextState)
5. 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

```
var Hello = React.createClass({
  getInitialState: function () {
    return {
      opacity: 1.0
    };
  },
```

```
  componentDidMount: function () {
    this.timer = setInterval(function () {
      var opacity = this.state.opacity;
      opacity -= .05;
      if (opacity < 0.1) {
        opacity = 1.0;
      }
      this.setState({
        opacity: opacity
      });
    }.bind(this), 100);
  },

  render: function () {
    return (
      <div style={{opacity: this.state.opacity}}>
        Hello {this.props.name}
      </div>
    );
  }
});

ReactDOM.render(
  <Hello name="world"/>,
  document.body
);
```

另外，组件的style属性的设置方式也值得注意，不能写成

```
style="opacity:{this.state.opacity};"
```

而要写成

```
style={{opacity: this.state.opacity}}
```

Ajax

组件的数据来源，通常是通过 `Ajax` 请求从服务器获取，可以使用 `componentDidMount` 方法设置`Ajax`请求，等到请求成功，再用`this.setState`方法重新渲染 UI

```
var UserGist = React.createClass({
  getInitialState: function() {
    return {
      username: '',
      lastGistUrl: ''
    };
```

```
  },

  componentDidMount: function() {
    $.get(this.props.source, function(result) {
      var lastGist = result[0];
      if (this.isMounted()) {
        this.setState({
          username: lastGist.owner.login,
          lastGistUrl: lastGist.html_url
        });
      }
    }.bind(this));
  },

  render: function() {
    return (
      <div>
        {this.state.username}'s last gist is
        <a href={this.state.lastGistUrl}>here</a>.
      </div>
    );
  }
});

ReactDOM.render(
 <UserGist source="https://api.github.com/users/octocat/gists" />,
  document.body
);
```

上面代码使用jQuery完成Ajax请求，这是为了便于说明。React本身没有任何依赖，完全可以不用jQuery，而使用其他库。

我们甚至可以把一个Promise对象传入组件

```
ReactDOM.render(
  <RepoList
   promise={$.getJSON('https://api.github.com/search/repositories?q=javascript&sort=stars')}
  />,
  document.body
);
```

上面代码从Github的API抓取数据，然后将Promise对象作为属性，传给RepoList组件。如果Promise对象正在抓取数据（pending状态），

```
var RepoList = React.createClass({
  getInitialState: function() {
    return { loading: true, error: null, data: null};
  },

  componentDidMount() {
```

```
    this.props.promise.then(
      value => this.setState({loading: false, data: value}),
      error => this.setState({loading: false, error: error}));
  },

  render: function() {
    if (this.state.loading) {
      return <span>Loading...</span>;
    }
    else if (this.state.error !== null) {
      return <span>Error: {this.state.error.message}</span>;
    }
    else {
      var repos = this.state.data.items;
      var repoList = repos.map(function (repo) {
        return (
          <li>
        <a href={repo.html_url}>{repo.name}</a> ({repo.stargazers_count} stars) <br/> {repo.d
          </li>
        );
      });
      return (
        <main>
          <h1>Most Popular JavaScript Projects in Github</h1>
          <ol>{repoList}</ol>
        </main>
      );
    }
  }
});
```

参考连接

1. React’s official site
2. React’s official examples
3. React (Virtual) DOM Terminology, by Sebastian Markbåge
4. The React Quick Start Guide, by Jack Callister
5. Learning React.js: Getting Started and Concepts, by Ken Wheeler
6. Getting started with React, by Ryan Clark
7. React JS Tutorial and Guide to the Gotchas, by Justin Deal
8. React Primer, by Binary Muse
9. jQuery versus React.js thinking, by zigomir