React 编程规范

基本规则

* 每个文件只包含一个React 组件
* 使用JSX语法
* 除非是一个非JSX文件中初始化app,否则不要使用React.createElement

**Class vs React.createClass**

* 除非有更好的理由使用混淆(mixins),否则就使用组件类继承React.Component。

示例:

// bad

const Listening = React.createClass({

render(){

return <div />

}

})

// good

class Listing extends React.Component {

render(){

return <div />

}

})

命名

* 扩展名 **:** 使用jsx作为React组件的扩展名
* 文件名：文件命名采用帕斯卡命名法(大驼峰)，如: ReservationCard.jsx
* 引用名**:**  组件引用采用帕斯卡命名法，其实例采用小驼峰命名法。

示例:

* 组件命名： 使用文件名作为组件名。例如: ReservationCard.jsx组件的引用名应该是ReservationCard。然而，对于一个目录的根组件，应该使用index.jsx作为文件名，使用目录名作为组件名。

// bad

const reservationCard = require(‘./ReservationCard’)

// good

const ReservationCard = require(‘./ReservationCard’)

// bad

const ReservationItem = <ReservationCard />

// good

const reservationItem = <ReservationCard />

// bad

const Footer = require(‘./Footer/Footer.jsx’)

//bad

const Footer= require(‘./Footer/index.jsx’)

// good

const Footer =require(‘./Footer’)

声明

* 不要通过displayName来命名组件，通过引用命名组件

对其

* 对于JSX语法，遵循下面的对齐风格。

// bad

export default React.createClass({

displayName: ‘ReservationCard’,

// stuff goes here

})

// good

export default class ReservationCard extends React.Component{

}

// bad

<Foo superLongParam=“bar”

auotherSuperLongParam=“baz” />

// good

<Foo

superLongParam=“bar”

auotherSuperLongParam=“baz”

/>

// 如果一行能写完就放一行

<Foo bar=“bar”/>

// 子元素缩进

</Foo

superLongParam=“bar”

auotherSuperLongParam=“baz”

>

<Spazz />

</Foo>

对其

* 对于JSX使用双引号，对其它所有JS属性使用单引号。

Ps: 为什么？ 因为JSX属性 不能包含被转移的引号， 并且双引号使得如”don’t”一样的连接词很容易被输出。常规的HTML属性也应该使用双引号而不是单引号，JSX属性反映了这个约定。

// bad

<Foo bar=‘bar’ />

// good

<Foo bar=“bar” />

// bad

<Foo style={{ left: “20px” }} />

// good

<Foo style={{ left: ‘20px’ }} />

// bad

<Foo/>

//very bad

<Foo />

// good

<Foo />

空格

* 在自闭和标签之前留一个空格

属性

* 属性名采用小驼峰式命名法

括号

* 当组件跨行时，要用括号包裹JSX标签。

// bad

<Foo

UserName=“hello”

photo\_number={12345678}

/>

// good

<Foo

userName=“hello”

photoNumber={12345678}

/>

// bad

render() {

return <MyComponent className=“long body” foo=“bar”>

<MyChild />

</MyComponent>;

}

// good

render() {

return (

<MyComponent className=“long body” foo=“bar”>

<MyChild />

</MyComponent>;

);

}

// good, when single line

render(){

const body = <div>hello</div>

return <MyComponent>{body}</MyComponent>

}

标签

* 没有子组件的父组件使用自闭和标签
* 如果组件有多行属性，闭合标签应写在新的一行上。

// bad

<Foo className=“stuff” /></Foo>

// good

<Foo className=“stuff” />

// bad

<Foo

bar=“bar”

baz=“baz” />

// good

<Foo

bar=“bar”

baz=“baz”

/>

方法

* 不要对React组件的内置方法使用underscore前缀

顺序

* 继承React.Component 的类的方法遵循下面的顺序.

1. constructor
2. optional static methods
3. getChildContext
4. componentWillMount
5. componentDidMount
6. componentWillReceiveProps
7. shouldComponentUpdate
8. componentWillUpdate

// bad

React.createClass({

\_onClickSubmit() {

// do stuff

}

// other stuff

})

// good

Class extends React.Component {

onClickSubmit(){

// do stuff

}

// other stuff

}

1. compoenntDidUpdate
2. componentWillUnmount
3. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
4. getter methods for render like getSelectReason() or getFooterContent()
5. Optional render methods like renderNavigation() or renderProfilePicture()
6. render

* 怎么定义 propTypes，defaultProps，contextTypes 等等…

// 代码

Import React, { PropTypes } from ‘react’;

const propsType = {

id: PropTypes.number.isRequired,

url: PropTypes.string.isRequired,

text: PropTypes.string,

};

const defaultProps = {

text: ‘Hello World’,

};

class Link extends React.Component {

static methodsAreOk (){

return true;

}

render(){

return <a href={this.props.url} data-id={ this.props.id } >{ this.props.text }</a>

}

}

Link.propTypes = propTypes;

* 使用 React.createClass 时，方法顺序如下：

1. displayName
2. propTypes
3. contextTypes
4. childContextTypes
5. mixing
6. statics
7. defaultProps
8. getDefaultProps
9. getInitialState
10. getChildContext
11. componentWillMount
12. componentDidMount
13. componentWillReceiveProps
14. shouldComponentUpdate
15. componentWillUpdate
16. componentDidUpdate
17. componentWillUnmount
18. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
19. getter methods for render like getSelectReason() or getFooterContent()
20. Optional render methods like renderNavigation() or renderProfilePicture()
21. render