Created by Firejune at 2016/01/11 23:10

Airbnb의 React/JSX 스타일 가이드

기왕 번역한 김에 하나 더 했습니다. Airbnb에서 Reac와 JSX 스타일 가이드도 작성했더군요. ES6과 JSX 기반의 React 코드를 작성하는 것을 마치 기본적인 것인 양 소개하고 있습니다. 요즘 Electron 기반 하이브리드 데스크탑 애플리케이션을 개발하고 있는데 전반에 걸쳐 React를 적용하는 중입니다. Electron은 구글 크롬 브라우저와 Node.js를 포함하고 있어서 하나의 브라우저에서 작동하는 것에만 집중할 수 있고 browserify 또는 webpack와 같은 모듈 번들러(module bundler)의 도움 없이 네이티브 require를 브라우저에서도 사용할 수 있다 보니, 의존성(dependency)이나 네임스페이스(namespace) 관리는 물론이고, 실시간 트랜스파일(Transpile)이 가능해서 ES6 코드를 마구 내질러도 크게 문제 될 것이 없는 개발환경을 만들 수 있다는 것은... 이건 뭐 그냥 완전히 다른 세상이라고 밖에 표현하지 못하겠군요.

1. 기본 규칙(Basic Rules)

하나의 파일에는 오직 하나의 React 컴포넌트를 사용합니다. 그러나, 다중 스테이트가 없는(Stateless) 또는 순수
는 허용됩니다. eslint rule: react/no - multi - comp.
항상 JSX 문법을 사용합니다.
JSX파일이 아닌 다른 app에서 초기화하는 경우를 제외하고는 React.createElement를 사용하지 않습니다.

2. 클래스(Class) vs React.createClass

특별한 이유로 믹스인(mixin)하는 경우를 제외하고는 class extends React.Component를 사용하세요.

eslint rules: react/prefer - es6 - class.

```
// bad
const Listing = React.createClass({
  render() {
    return <div />;
  }
});
// good
class Listing extends React.Component {
  render() {
    return <div />;
  }
}
```

3. 명명(Naming)

확장자: React 컴포넌트는 .jsx 확장자를 사용합니다.

파일명: 파일명에는 PascalCase(대문자로 시작)를 사용합니다. 예), ReservationCard.jsx.

참조명: React 컴포넌트의 참조 이름에는 PascalCase를 쓰고 그 인스턴스의 이름에는 camelCase(소문자로 시작)

eslint rules: react/jsx - pascal - case.

```
// bad
import reservationCard from './ReservationCard';
// good
import ReservationCard from './ReservationCard';
// bad
const ReservationItem = <ReservationCard />;
// good
const reservationItem = <ReservationCard />;
```

컴포넌트명: 컴포넌트명으로 파일명을 씁니다. 예), ReservationCard.jsx 파일은 ReservationCard라는 참조명을

```
// bad
import Footer from './Footer/Footer';
// bad
import Footer from './Footer/index';
// good
import Footer from './Footer';
```

4. 선언(Declaration)

displayName을 이용하여 컴포넌트명을 정하지 않습니다. 그대신, 참조에 의해 이름을 지정합니다.

```
// bad
export default React.createClass({
  displayName: 'ReservationCard',
  // stuff goes here
});
// good
export default class ReservationCard extends React.Component {
}
```

5. 조정(Alignment)

JSX 구문에 따른 정렬 스타일을 사용합니다.

eslint rules: react/jsx - closing - bracket - location.

```
// bad
<Foo superLongParam="bar"
     anotherSuperLongParam="baz" />
// good
<Foo
  superLongParam="bar"
  anotherSuperLongParam="baz"
/>
// if props fit in one line then keep it on the same line
<Foo bar="bar" />
// children get indented normally
<Foo
  superLongParam="bar"
  anotherSuperLongParam="baz"
>
  <Spazz />
```

6. 인용(Quotes)

JSX 속성(attributes)에는 항상 큰 따옴표(")를 사용합니다. 그러나 다른 모든 자바스크립트에는 작은 따옴표(sing

왜죠? JSX 속성(attributes)은 따옴표(quotes)의 탈출(escaped)을 포함할 수 없습니다. 그래서 큰 따옴표를 이용

eslint rules: jsx-quotes.

```
// bad
<Foo bar='bar' />
// good
<Foo bar="bar" />
// bad
<Foo style={{ left: "20px" }} />
// good
<Foo style={{ left: '20px' }} />
```

7. 공백(Spacing)

자신을 닫는(self-closing) 태그에는 항상 하나의 공백만을 사용합니다.

```
// bad
<Foo/>
// very bad
<Foo                 />
// bad
<Foo
 />
// good
<Foo />
```

8. 속성(Props)

prop 이름은 항상 camelCase(소문자로 시작)를 사용합니다.

```
// bad
<Foo
  UserName="hello"
  phone_number={12345678}
/>
// good
<Foo
  userName="hello"
  phoneNumber={12345678}
/>
```

명시적으로 true 값을 가지는 prop은 그 값을 생략할 수 있습니다.

eslint rules: react/jsx - boolean - value.

```
// bad
<Foo
  hidden={true}
/>
// good
<Foo
  hidden
/>
```

9. 괄호(Parentheses)

JSX 태그가 감싸여(Wrap) 있어 한 줄 이상인 경우 괄호(parentheses)를 사용합니다.

eslint rules: react/wrap - multilines.

```
// 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>;
  }
```

10. 태그(Tags)

자식(children)을 가지지 않는다면 항상 자신을 닫는(self-close) 태그로 작성합니다.

eslint rules: react/self-closing-comp.

```
// bad
<Foo className="stuff"></Foo>
// good
<Foo className="stuff" />
```

만약, 컴포넌트의 속성(properties)을 여러 줄에 있는 경우, 닫는 태그는 다음 줄에 작성합니다.

eslint rules: react/jsx-closing-bracket-location.

```
// bad
<Foo
  bar="bar"
  baz="baz" />
// good
<Foo
  bar="bar"
  baz="baz"
/>
```

11. 메소드(Methods)

렌더(Render) 메소드에서 이벤트 핸들러에 바인드(Bind)가 필요한 경우에는 생성자(constructor)에서 합니다.

왜죠? 렌더러 메소드에서 바인드(bind)를 호출하게 되면 랜더링 할 때 마다 매번 새로운 함수를 생성하게 됩니다.

eslint rules: react/jsx - no - bind.

```
// bad
class extends React.Component {
  onClickDiv() {
    // do stuff
  }
  render() {
    return <div onClick={this.onClickDiv.bind(this)} />
  }
}
// good
class extends React.Component {
  constructor(props) {
    super(props);
    this.onClickDiv = this.onClickDiv.bind(this);
  }
  onClickDiv() {
    // do stuff
  }
  render() {
    return <div onClick={this.onClickDiv} />
  }
}
```

React 컴포넌트의 내부 메소드에 밑줄(underscore)을 접두사로 사용하지 않습니다.

```
// bad
React.createClass({
  _onClickSubmit() {
    // do stuff
  },
  // other stuff
});
// good
class extends React.Component {
  onClickSubmit() {
    // do stuff
  }
  // other stuff
}
```

12. 호출순서(Ordering)

class extends React.Component의 호출순서(Ordering):

constructor
추가적인(optional) static 메소드
getChildContext

componentWillMount
componentDidMount
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
componentDidUpdate
componentWillUnmount
onClickSubmit()와 같은 *clickHandlers* 또는 *eventHandlers* 또는 onChangeDescription()
getSelectReason()와 같은 render를 위한 getter methods 또는 getFooterContent()
renderNavigation()와 같은 *추가적인 렌더러 메소드* 또는 renderProfilePicture()

render

React.createClass의 호출순서(Ordering):

displayName
propTypes
contextTypes
childContextTypes
mixins
statics
defaultProps
getDefaultProps
getInitialState
getChildContext
componentWillMount
componentDidMount
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
componentDidUpdate
componentWillUnmount
onClickSubmit()와 같은 *clickHandlers* 또는 *eventHandlers* 또는 onChangeDescription()
getSelectReason()와 같은 render를 위한 getter methods 또는 getFooterContent()
renderNavigation()와 같은 *추가적인 렌더러 메소드* 또는 renderProfilePicture()

render

eslint rules: [react/sort - comp](react/sort-comp).

propTypes, defaultProps, contextTypes, 등을 어떻게 정의할까요...

```
import React, { PropTypes } from 'react';
const propTypes = {
  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;
Link.defaultProps = defaultProps;
export default Link;
```

13. isMounted

isMounted는 사용하지 않습니다.

왜죠? isMounted는 안티-패턴(anti-pattern)입니다. ES6 클래스에서는 사용할수도 없습니다. 그리고 공식적으로

eslint rules: react/no-is-mounted.