# React/React Native 的ES5 ES6写法对照表

> 本文为转载，原文链接

很多React/React Native的初学者都被ES6的问题迷惑：各路大神都建议我们直接学习ES6的语法（`class Foo extends React.Component`），然而网上搜到的很多教程和例子都是ES5版本的，所以很多人在学习的时候连照猫画虎都不知道怎么做。今天在此整理了一些ES5和ES6的写法对照表，希望大家以后读到ES5的代码，也能通过对照，在ES6下实现相同的功能。

## 模块

### 引用

在ES5里，如果使用CommonJS标准，引入React包基本通过require进行，代码类似这样：

```
//ES5
var React = require("react");
var {
    Component,
    PropTypes
} = React;  //引用React抽象组件

var ReactNative = require("react-native");
var {
    Image,
    Text,
} = ReactNative;  //引用具体的React Native组件
```

在ES6里，import写法更为标准

```
//ES6
import React, {
    Component,
    PropTypes,
} from 'react';
import {
    Image,
    Text
} from 'react-native'
```

### 导出单个类

在ES5里，要导出一个类给别的模块用，一般通过module.exports来导出

```
//ES5
var MyComponent = React.createClass({
    ...
});
module.exports = MyComponent;
```

在ES6里，通常用export default来实现相同的功能：

```
//ES6
export default class MyComponent extends Component{
    ...
}
```

引用的时候也类似：

```
//ES5
var MyComponent = require('./MyComponent');

//ES6
import MyComponent from './MyComponent';
```

注意导入和导出的写法必须配套，不能混用！

# 定义组件

在ES5里，通常通过React.createClass来定义一个组件类，像这样：

```
//ES5
var Photo = React.createClass({
    render: function() {
        return (
            <Image source={this.props.source} />
        );
    },
});
```

在ES6里，我们通过定义一个继承自React.Component的class来定义一个组件类，像这样：

```
//ES6
class Photo extends React.Component {
    render() {
        return (
            <Image source={this.props.source} />
        );
    }
}
```

## 给组件定义方法

从上面的例子里可以看到，给组件定义方法不再用 `名字: function()` 的写法，而是直接用 `名字()` ，在方法的最后也不能有逗号了。

```
//ES5
var Photo = React.createClass({
    componentWillMount: function(){

    },
    render: function() {
        return (
            <Image source={this.props.source} />
        );
    },
});
```

```
//ES6
class Photo extends React.Component {
    componentWillMount() {

    }
    render() {
        return (
            <Image source={this.props.source} />
        );
    }
}
```

## 定义组件的属性类型和默认属性

在ES5里，属性类型和默认属性分别通过propTypes成员和getDefaultProps方法来实现

```
//ES5
var Video = React.createClass({
    getDefaultProps: function() {
        return {
            autoPlay: false,
            maxLoops: 10,
        };
    },
    propTypes: {
        autoPlay: React.PropTypes.bool.isRequired,
        maxLoops: React.PropTypes.number.isRequired,
        posterFrameSrc: React.PropTypes.string.isRequired,
        videoSrc: React.PropTypes.string.isRequired,
    },
    render: function() {
        return (
            <View />
        );
    },
});
```

在ES6里，可以统一使用static成员来实现

```
//ES6
class Video extends React.Component {
    static defaultProps = {
        autoPlay: false,
        maxLoops: 10,
    };  // 注意这里有分号
    static propTypes = {
        autoPlay: React.PropTypes.bool.isRequired,
        maxLoops: React.PropTypes.number.isRequired,
        posterFrameSrc: React.PropTypes.string.isRequired,
        videoSrc: React.PropTypes.string.isRequired,
    };  // 注意这里有分号
    render() {
        return (
            <View />
        );
    } // 注意这里既没有分号也没有逗号
}
```

也有人这么写，虽然不推荐，但读到代码的时候你应当能明白它的意思：

```
//ES6
class Video extends React.Component {
    render() {
        return (
            <View />
        );
    }
}
Video.defaultProps = {
    autoPlay: false,
    maxLoops: 10,
};
Video.propTypes = {
    autoPlay: React.PropTypes.bool.isRequired,
    maxLoops: React.PropTypes.number.isRequired,
    posterFrameSrc: React.PropTypes.string.isRequired,
    videoSrc: React.PropTypes.string.isRequired,
};
```

*注意:* 对React开发者而言，static成员在IE10及之前版本不能被继承，而在IE11和其它浏览器上可以，这有时候会带来一些问题。React Native开发者可以不用担心这个问题。

## 初始化state

ES5下情况类似，

```
//ES5
var Video = React.createClass({
    getInitialState: function() {
        return {
            loopsRemaining: this.props.maxLoops,
        };
    },
})
```

ES6下，有两种写法：

```
//ES6
class Video extends React.Component {
    state = {
        loopsRemaining: this.props.maxLoops,
    }
}
```

不过我们推荐更易理解的在构造函数中初始化（这样你还可以根据需要做一些计算）：

```
//ES6
class Video extends React.Component {
    constructor(props){
        super(props);
        this.state = {
            loopsRemaining: this.props.maxLoops,
        };
    }
}
```

# 把方法作为回调提供

很多习惯于ES6的用户反而不理解在ES5下可以这么做：

```
//ES5
var PostInfo = React.createClass({
    handleOptionsButtonClick: function(e) {
        // Here, 'this' refers to the component instance.
        this.setState({showOptionsModal: true});
    },
    render: function(){
        return (
            <TouchableHighlight onPress={this.handleOptionsButtonClick}>
                <Text>{this.props.label}</Text>
            </TouchableHighlight>
        )
    },
});
```

在ES5下，React.createClass会把所有的方法都bind一遍，这样可以提交到任意的地方作为回调函数，而this不会变化。但官方现在逐步认为这反而是不标准、不易理解的。

在ES6下，你需要通过bind来绑定this引用，或者使用箭头函数（它会绑定当前scope的this引用）来调用

```
//ES6
class PostInfo extends React.Component
{
    handleOptionsButtonClick(e){
        this.setState({showOptionsModal: true});
    }
    render(){
        return (
            <TouchableHighlight
                onPress={this.handleOptionsButtonClick.bind(this)}
                onPress={e=>this.handleOptionsButtonClick(e)}
                >
                <Text>{this.props.label}</Text>
            </TouchableHighlight>
        )
    },
}
```

箭头函数实际上是在这里定义了一个临时的函数，箭头函数的箭头 `=>` 之前是一个空括号、单个的参数名、或用括号括起的多个参数

名，而箭头之后可以是一个表达式（作为函数的返回值），或者是用花括号括起的函数体（需要自行通过return来返回值，否则返回的是undefined）。

```
// 箭头函数的例子
()=>1
v=>v+1
(a,b)=>a+b
()=>{
    alert("foo");
}
e=>{
    if (e == 0){
        return 0;
    }
    return 1000/e;
}
```

需要注意的是，不论是bind还是箭头函数，每次被执行都返回的是一个新的函数引用，因此如果你还需要函数的引用去做一些别的事情（譬如卸载监听器），那么你必须自己保存这个引用

```
// 错误的做法
class PauseMenu extends React.Component{
    componentWillMount(){
        AppStateIOS.addEventListener('change', this.onAppPaused.bind(this));
    }
    componentDidUnmount(){
        AppStateIOS.removeEventListener('change', this.onAppPaused.bind(this));
    }
    onAppPaused(event){
    }
}
```

```
// 正确的做法
class PauseMenu extends React.Component{
    constructor(props){
        super(props);
        this._onAppPaused = this.onAppPaused.bind(this);
    }
    componentWillMount(){
        AppStateIOS.addEventListener('change', this._onAppPaused);
    }
    componentDidUnmount(){
        AppStateIOS.removeEventListener('change', this._onAppPaused);
    }
    onAppPaused(event){
    }
}
```

从这个帖子中我们还学习到一种新的做法：

```
// 正确的做法
class PauseMenu extends React.Component{
    componentWillMount(){
        AppStateIOS.addEventListener('change', this.onAppPaused);
    }
    componentDidUnmount(){
        AppStateIOS.removeEventListener('change', this.onAppPaused);
    }
    onAppPaused = (event) => {
        //把方法直接作为一个arrow function的属性来定义，初始化的时候就绑定好了this指针
    }
}
```

# Mixins

在ES5下，我们经常使用mixin来为我们的类添加一些新的方法，譬如PureRenderMixin

```
var PureRenderMixin = require('react-addons-pure-render-mixin');
React.createClass({
  mixins: [PureRenderMixin],

  render: function() {
    return <div className={this.props.className}>foo</div>;
  }
});
```

然而现在官方已经不再打算在ES6里继续推行Mixin，他们说：Mixins Are Dead. Long Live Composition。

尽管如果要继续使用mixin，还是有一些第三方的方案可以用，譬如这个方案

不过官方推荐，对于库编写者而言，应当尽快放弃Mixin的编写方式，上文中提到Sebastian Markbåge的一段代码推荐了一种新的编码方式：

```
//Enhance.js
import { Component } from "React";

export var Enhance = ComposedComponent => class extends Component {
    constructor() {
        this.state = { data: null };
    }
    componentDidMount() {
        this.setState({ data: 'Hello' });
    }
    render() {
        return <ComposedComponent {...this.props} data={this.state.data} />;
    }
};
```

```
//HigherOrderComponent.js
import { Enhance } from "./Enhance";

class MyComponent {
    render() {
        if (!this.data) return <div>Waiting...</div>;
        return <div>{this.data}</div>;
    }
}

export default Enhance(MyComponent); // Enhanced component
```

用一个“增强函数”，来某个类增加一些方法，并且返回一个新类，这无疑能实现mixin所实现的大部分需求。

# ES6+带来的其它好处

## 解构&属性延展

结合使用ES6+的解构和属性延展，我们给孩子传递一批属性更为方便了。这个例子把className以外的所有属性传递给div标签：

```
class AutoloadingPostsGrid extends React.Component {
    render() {
        var {
            className,
            ...others,  // contains all properties of this.props except for className
        } = this.props;
        return (
            <div className={className}>
                <PostsGrid {...others} />
                <button onClick={this.handleLoadMoreClick}>Load more</button>
            </div>
        );
    }
}
```

下面这种写法，则是传递所有属性的同时，用覆盖新的className值：

```
<div {...this.props} className="override">
    ...
</div>
```

这个例子则相反，如果属性中没有包含**className**，则提供默认的值，而如果属性中已经包含了，则使用属性中的值

```
<div className="base" {...this.props}>
    ...
</div>
```

以上便完成了所有的介绍～