原文地址： <http://babeljs.io/blog/2015/06/07/react-on-es6-plus/>

[](http://7xiyp1.com1.z0.glb.clouddn.com/652b57761014ee46dc3114e22229b92f.png)

在今年对 Instagram Web 进行全新的设计的时候，我喜欢在写 React 组件的时候，用上一些 ES6+ 的新特性。请允许我列举这些能够改变你写 React 应用方式的新特性。比起以往，这些特性能够使你撸起码来更加容易、有趣！

**类（Class）**

使用 ES6+ 来编写 React 组件最明显的变化就是我们定义组件（类）的语法的方式。我们可以用定义一个继承了 React.Component 的ES6 类来代替原本使用 React.createClass 的来创建类的方式：

|  |  |
| --- | --- |
| 1 2 3 4 5 | class Photo extends React.Component {  render() {  return <img alt={this.props.caption} src={this.props.src} />;  } } |

我们可以发现这种写法使得定义组件的方式变得更加简洁：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | // The ES5 way var Photo = React.createClass({  handleDoubleTap: function(e) { … },  render: function() { … }, }); // The ES6+ way class Photo extends React.Component {  handleDoubleTap(e) { … }  render() { … } } |

这样我们可以少写一对圆括号、一个分号、每个方法的冒号和 function 关键字。

所有生命周期方法都可以采用这种方式来定义。 但是 componentWillMount 还可以用 constructor 来代替：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | // The ES5 way var EmbedModal = React.createClass({  componentWillMount: function() { … }, }); // The ES6+ way class EmbedModal extends React.Component {  constructor(props) {  super(props);  // Operations usually carried out in componentWillMount go here  } } |

**属性初始化（property initializers）**

在 ES6+ 类中，属性类型 prop type 和默认属性 default prop 可以通过类中的 static 来声明。同时，组件的初始状态（ initial state ）可以通过 ES7 的[属性初始化（property initializers）](https://gist.github.com/jeffmo/054df782c05639da2adb)来完成：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 | // The ES5 way var Video = React.createClass({  getDefaultProps: function() {  return {  autoPlay: false,  maxLoops: 10,  };  },  getInitialState: function() {  return {  loopsRemaining: this.props.maxLoops,  };  },  propTypes: {  autoPlay: React.PropTypes.bool.isRequired,  maxLoops: React.PropTypes.number.isRequired,  posterFrameSrc: React.PropTypes.string.isRequired,  videoSrc: React.PropTypes.string.isRequired,  }, }); // The ES6+ way class Video extends React.Component {  static defaultProps = {  autoPlay: false,  maxLoops: 10,  }  static propTypes = {  autoPlay: React.PropTypes.bool.isRequired,  maxLoops: React.PropTypes.number.isRequired,  posterFrameSrc: React.PropTypes.string.isRequired,  videoSrc: React.PropTypes.string.isRequired,  }  state = {  loopsRemaining: this.props.maxLoops,  } } |

ES7 中在构造函数（ constructor ）下的属性初始化操作中的 this 指向的是类的实例，所以初始状态（ initial state ）可以通过 this.prop （即传入的参数）来设定。

**箭头函数（Arrow function）**

React.createClass 方法在你的组件上做了一些额外的绑定工作，以确保在组件实实例的方法内部， this 指向的是组件实例自身。

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | // Autobinding, brought to you by React.createClass var PostInfo = React.createClass({  handleOptionsButtonClick: function(e) {  // Here, 'this' refers to the component instance.  this.setState({showOptionsModal: true});  }, }); |

由于我们使用 ES6+ 的语法定义类的时候没有采用 React.createClass 的方式，所以，这样看来我们不得不手动来绑定这些方法中 this 的指向：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 | // Manually bind, wherever you need to class PostInfo extends React.Component {  constructor(props) {  super(props);  // Manually bind this method to the component instance...  this.handleOptionsButtonClick = this.handleOptionsButtonClick.bind(this);  }  handleOptionsButtonClick(e) {  // ...to ensure that 'this' refers to the component instance here.  this.setState({showOptionsModal: true});  } } |

幸运的是，通过 ES6+ 的箭头函数（ Arrow functions ）和属性初始化（ property initializers ）这两个特性使得把函数的 this 指向绑定为组件的实例变得非常的简单：

|  |  |
| --- | --- |
| 1 2 3 4 5 | class PostInfo extends React.Component {  handleOptionsButtonClick = (e) => {  this.setState({showOptionsModal: true});  } } |

函数体内的 this 对象，绑定定义时所在的对象，而不是使用时所在的对象。而恰好属性初始化（ property initializers ）刚好在这个作用域内。

**动态属性名 & 字符串模板**

在 ES6+ 中对 *对象字面量的扩展* 使得我们可以在对象字面量中使用表达式来对属性命名。如果是在 ES5 中，我们也许只能这样做：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | var Form = React.createClass({  onChange: function(inputName, e) {  var stateToSet = {};  stateToSet[inputName + 'Value'] = e.target.value;  this.setState(stateToSet);  }, }); |

但是，在 ES6+ 中，我们不仅可以在对象字面量属性的定义中使用表达式，还有使用使用 *字符串模板* ：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | class Form extends React.Component {  onChange(inputName, e) {  this.setState({  [`${inputName}Value`]: e.target.value,  });  } } |

**析构 & 扩展运算符**

我们在编写组件的过程中，经常遇到要从父组件要把自己的很多属性多传给子组件的情况。有了 ES6+ 的 *析构* 和 *扩展运算符* 特性，这变得非常的方便：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | 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>  );  } } |

我们可以把 *扩展运算符* 属性和普通的属性结合起来使用，这样使得我们可以利用优先级来使用属性的默认值和属性的覆盖。下面这个元素会获得一个 override 的类（ class ），及时 this.props 中有传递 className 属性。

|  |  |
| --- | --- |
| 1 2 3 | <div {...this.props} className="override">  … </div> |

下面这种写法，可以给元素设定默认的 className：

|  |  |
| --- | --- |
| 1 2 3 | <div className="base" {...this.props}>  … </div> |

**最后**

我希望你能够享受 ES6+ 的这些特性给你在编写 React.js 中带来的好处。感谢我的同事他们为这篇文章作出的贡献，还有，特别的感谢 Babel 团队，使得我们可以随意的使用这些特性。