□□□□□

□□

- □□□□ ___
- componentDidUpdate
- shouldComponentUpdate
- componentWillUpdate
- componentWillReceiveProps
- componentWillMount
- componentWillUnmount

□□□

□□□□□□□□□□□□□□□□

- getDefaultProps
- getInitialState
- componentWillMount
- render
- componentDidMount

□□□ `getDefaultProps` □ `getInitialState` □□□□□□□□□□□

```
// □□□□□□□□ this.props □□
getDefaultProps: function(){
  return { /* something here */};
}

// □□□□□□□□ this.state □□
getInitialState: function(){
  return { /* something here */};
}
```

`componentDidMount`方法已经可以读取组件生成的 DOM。如果要与 DOM 互动，应该就在这个方法里面，而不是在`render`方法里面。

卸载组件时，按如下顺序执行。

- 卸载组件
- componentWillUnmount

重新挂载组件时，按如下顺序执行。

- getInitialState
- componentWillMount
- render
- componentDidMount

再次渲染组件时，组件接受到父组件传来的新参数，按如下顺序执行。

- 更新 Props
- componentWillReceiveProps
- shouldComponentUpdate
- componentWillUpdate
- render
- componentDidUpdate

如果组件自身的`state`更新了，按如下顺序执行。

1. 更新 state
2. shouldComponentUpdate
3. componentWillUpdate
4. render
5. componentDidUpdate

componentDidUpdate

`componentDidUpdate`方法在每次组件重新渲染（`render`方法）后执行。

```
componentDidUpdate: function(prevProps, prevState){
  //
}
```

该方法可以操作组件所在的 DOM，用于操作数据已经更新之后的组件。

```
componentDidUpdate: function() {
  this.scrollElement();
},

render: function() {
  return [...]
```

上面代码在组件每次重新渲染后，会执行 `this.scrollElement` 方法。

这里有一个问题，如果 `this.scrollElement` 里面需要操作 DOM，这时很可能 DOM 还没有完全生成。因此，可以使用 `requestAnimationFrame` 或 `setTimeout` 方法，保证操作时 DOM 已经生成。

```
scrollElement: function() {
  var _this = this;
  setTimeout(function () {
    window.requestAnimationFrame(function() {
      var node = _this.getDOMNode();
      if (node !== undefined) {
        node.scrollTop = node.scrollHeight;
      }
    })
  }, 0)
},
componentDidMount: function() {
  this.scrollElement();
},
```

shouldComponentUpdate

`shouldComponentUpdate` 方法会在每次重新渲染（ `render` 方法）之前，自动调用。它返回一个布尔值，决定是否应该进行此次渲染。默认为 `true` ，表示进行渲染，如果为 `false` ，就表示中止渲染。

```
shouldComponentUpdate: function(nextProps, nextState){
  // return a boolean value
  return true;
}
```

下面是父元素组件 `Color` 的代码。

```
getInitialState: function () {
  return {
    colors: new Immutable.List(this.props.colors)
  };
},

_addColor: function (newColor) {
  this.setState({
    colors = this.state.colors.push(newColor)
  });
},

render: function () {
  return (
    <div>
      <ColorList colors={this.state.colors} />
      <ColorForm addColor={this._addColor} />
    </div>
  );
}
```

上面代码中，父组件 `Color` 向子组件 `ColorList` 传入参数 `this.state.colors`。每当父组件的 `this.state.colors` 变动时，子组件就会重新渲染，这时子组件的 `shouldComponentUpdate` 就会自动调用。

```
shouldComponentUpdate: function (nextProps, nextState) {
  return nextProps.colors !== this.props.colors;
}
```

上面是子组件的 `shouldComponentUpdate` 方法，它接受两个参数，第一个是本次传入的新参数对象 `nextProps`，第二个是新的状态对象 `nextState`。在方法内部，`this.props` 和 `this.state` 表示当前（没有重新渲染之前）的参数对象和状态对象。

注意，`shouldComponentUpdate` 方法默认返回 `true`，这意味着即使 `state` 和 `props` 没有改变，只要调用 `this.setState`，就会导致组件重新渲染。

componentWillUpdate

一旦 `shouldComponentUpdate` 返回 `true`，`componentWillUpdate` 就会执行，主要用于为即将到来的更新做准备工作。

```
componentWillUpdate: function(nextProps, nextState){
  // perform any preparations for an upcoming update
}
```

注意，这个方法之中不应该调用 `this.setState`，因为它本身不应该触发更新。

componentWillReceiveProps

`componentWillReceiveProps`方法在父组件每次重新传给当前组件参数时调用，它在当前组件的 `render()` 之前调用。组件的第一次渲染不会调用这个方法。

它只在父组件更新 `props`时执行，当前组件本身调用 `setState` 而引发重新渲染，是不会执行这个方法的。在此方法中调用 `setState`也不会二次渲染的。`componentWillReceiveProps`可以根据已有的 `props`和刚刚传入的 `props`，进行 `state` 的更新，而不会触发组件的重新更新。

```
componentWillReceiveProps: function(nextProps) {
  this.setState({
    // set something
  });
}
```

在 `componentWillReceiveProps`之中，可以调用 `setState`方法。而 `componentWillUpdate`是用来回应 `state` 变化的方法。

`componentWillReceiveProps`在 `shouldComponentUpdate`和 `componentWillUpdate`之前调用。

这个方法可以用来在 `render()`调用之前，对 props进行调整，然后通过 `this.setState()`设置 state。老的 props可以用 `this.props`拿到。

```
componentWillReceiveProps: function(nextProps) {
  this.setState({
    likesIncreasing: nextProps.likeCount > this.props.likeCount
  });
}
```

父组件操作子组件的基本流程如下。

- 父组件向子组件传入一个新参数
- 子组件在 `componentWillReceiveProps`方法里面处理新的参数，必要时调用 `setState`方法
- 子组件在 `componentWillUpdate`方法里面处理新的 state，但是一般来说，只使用 `componentWillReceiveProps`方法就足够了

□□□□□□□□□

```
function rand(arr) {
  return arr[Math.floor(Math.random() * arr.length)];
}

var ADayInTheLife = React.createClass({
  getInitialState: function () {
    return {
      doing: "isCoding"
    };
  },

  handleButtonClick: function () {
    var activities = ["isCoding", "isEating", "isSleeping"];
    var randAct = rand(activities);
    this.setState({
      doing: randAct
    });
  },

  render: function () {
    return (
      <div>
        <button onClick={this.handleButtonClick}>Next Activity</button>
        <Jake activity={this.state.doing} />
      </div>
    );
  }

});

React.render(<ADayInTheLife />, document.body);
```

□□□□□□□□□ `ADayInTheLife` □□□□□□ `Jake` □

□□□□□□ `Jake` □□□□

```
var Jake = React.createClass({

  getDefaultProps: function () {
```

```
    return {
      activity: "nothingYet"
    };
  },

  getInitialState: function () {
    return {
      thinking: "nothing yet"
    };
  },

  calcThinking: function (activity) {
    var thoughts = {
      isCoding: "yay, code",
      isEating: "yum, code",
      isSleeping: "where's the code?"
    };

    this.setState({
      thinking: thoughts[activity]
    })
  },

  componentWillReceiveProps: function (nextProps) {
    this.calcThinking(nextProps.activity);
  },

  render: function () {
    return <div>Jake: <b>{this.props.activity}</b> and thinking
"{this.state.thinking}".</div>;
  }
});
```

□□□□□□□□□□□□ `Jake` □□□□□ `componentWillReceiveProps` □□□□□□□□□□□
`calcThinking` □□□□□□ `setState` □□□□□ `Jake` □□□□ state□□□□□

componentWillMount

`componentWillMount`方法在第一次渲染之前调用。它只会执行一次，在浏览器和服务器都会执行。一般用来对`props`和 `state`进行初始化处理。

```
  getInitialState: function() {
    return {
      items: this.props.initialItems || [],
      sort: this.props.config.sort || { column: "", order: "" },
      columns: this.props.config.columns
    };
  },
  componentWillMount: function() {
    this.loadData(this.props.dataSource);
  },
  loadData: function(dataSource) {
    if (!dataSource) return;

    $.get(dataSource).done(function(data) {
      console.log("Received data");
     this.setState({items: data});
     }.bind(this)).fail(function(error, a, b) {
      console.log("Error loading JSON");
     });
  },
```

上面代码中，`getInitialState`为首次渲染设置默认参数。在首次渲染之前，会执行`componentWillMount`方法。该方法内部调用`loadData`方法，发出 AJAX 请求。这个请求有可能成功，也有可能不成功，而且不知道需要多久才能完成。在AJAX请求返回结果，执行`setState`方法设置新的 state之前，该组件将以默认值渲染。所以，使用`getInitialState`方法设置默认参数的意义就在这里。

注意，该方法之中不能调用`setState`。

componentWillUnmount

`componentWillUnmount`方法在组件从 DOM 移除后调用。一种用途是清除`componentDidMount`方法中添加的定时器。

版本号 #2
由 壳壳虫 创建于 2017-04-09 10:10:41 UTC
由 壳壳虫 更新于 2017-06-09 13:16:30 UTC