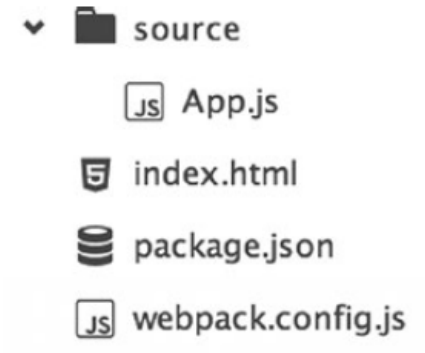
Development workflow

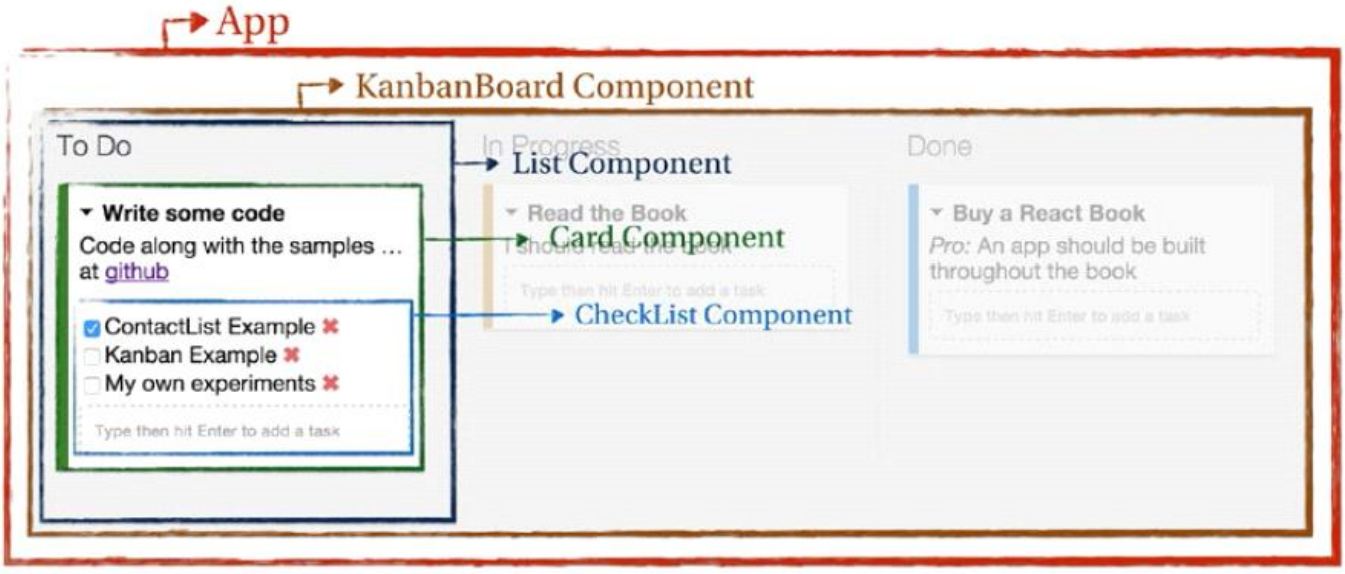
2016年3月10日
9:52

- Write JSX and transform it into regular JavaScript on the fly
- Write code in a module pattern
- Manage dependencies
- Bundle JavaScript files and use source maps for debugging

With this in mind, the basic project structure for a React project contains the following:

1. A source folder, to contain all your JavaScript modules.
2. An index.html file
3. A package.json file
4. A module packager or build tool, webpack as the preferred tool for this job.





Available events

2016年3月10日
13:12

Table 2-1. *Touch and Mouse Events*

onTouchStart	onTouchMove	onTouchEnd	onTouchCancel	
onClick	onDoubleClick	onMouseDown	onMouseUp	onMouseOver
onMouseMove	onMouseEnter	onMouseLeave	onMouseOut	onContextMenu
onDrag	onDragEnter	onDragLeave	onDragExit	onDragStart
onDragEnd	onDragOver	onDrop		

Table 2-2. *Keyboard Events*

onKeyDown	onKeyUp	onKeyPress

Table 2-3. *Focus and Form Events*

onFocus	onBlur	
onChange	onInput	onSubmit

Table 2-4. *Other Events*

onScroll	onWheel	onCopy	onCut	onPaste

Differences Between JSX and HTML

2016年3月10日
13:19

1. Tag attributes are camel cased.
2. All elements must be balanced.
3. The attribute names are based on the DOM API, not on the HTML language specs

HTML	JSX
<input type="text" **maxlength**="30" />	<input type="text" **maxLength**="30" />
<div id="box" **class**="some-class">	<div id="box" **className**="some-class">

JSX QFA:

No.	Not work	Working
Single Root Node	return (Hello World Have a nice day)	return(Hello World)

Conditional Clauses	<div className={if (condition)
	{ "salutation" }}>Hello JSX
		// way-1
		<div className={condition ? "salutation" : ""}>
		Hello JSX
		
		// way-2
		{ condition ?
		Hello JSX
		: null
		}
		
		// way -3
		{ condition &&
		Hello JSX
		}

Move the Condition Out	render() {
	return (
	<div className={if (condition)
	{ "salutation" }}>
	Hello JSX
	
	)
	}
		move the conditional outside of JSX, like:
		
		Render(){
		let className;
		If(condition){
		className = 'salutation';
		}
		
		return (Hello JSX)
		}

Comments in JSX

2016年3月10日
13:40

```
let content = (
    <Nav>
    {/* child comment, put {} around */}
    <Person
    /* multi
    line
    comment */
    name={window.isLoggedIn ? window.name : ''} // end of line comment
    />
    </Nav>
);
```

Rendering Dynamic HTML

2016年3月10日
13:44

React has built-in XSS attack protection, which means that by default it won't allow HTML tags to be generated dynamically and attached to JSX. This is generally good, but in some specific cases you might want to generate HTML on the fly. One example would be rendering data in markdown format to the interface.

React provides the **dangerouslySetInnerHTML** property to skip XSS protection and render anything directly

```
npm install --save marked
import marked from 'marked';
```

Then, you're going to use the function marked() provided by the library to convert the markdown to HTML (I have omitted some code not pertinent to this example for brevity):

```
{marked(this.props.description)}
```

Defining Inline Styles

2016年3月10日
13:56

In React’s components, inline styles are specified as a JavaScript object. Style names are camel cased in orderto be consistent with DOM properties (e.g. node.style.backgroundImage). Additionally, it’s not necessary tospecify pixel units - React automatically appends the correct unit behind the scenes. The following example
shows an example of inline styling in React:

```
import React, { Component} from 'react';
import {render} from 'react-dom';

class Hello extends Component{
      render(){
         let style = {
                  Width: 100,
                  Height: 100,
                  paddingLeft:50,
                  backgroundColor: '#000'
            };

       return (<div style={style}>Hello</div>);

      }
}
```

Working With Forms

2016年3月10日
14:06

Controlled Components	A form component with a value or checked prop is called a controlled component. In a controlled component, the value rendered inside the element will always reflect the value of the prop. By default the user won't be able to change it Special Cases: 1. use \n if you want newlines, for example <textarea value="This is a description.\n want newlines, for example" /> 2. Select <select value="B"> <option value="A">Mobile</option> <option value="B">Work</option> <option value="C">Home</option> </select>
Uncontrolled Components	Any input that does not supply a **value** is an uncontrolled component return (<form> Name: <input name="name" type="text" /> E-mail: <input name="email" type="mail" /> <button type="submit">Submit</button> </form>) If you want to set up an initial value for an uncontrolled form component, use the **defaultValue** prop instead of value.

// Search Component

```
import React, {Component} from 'react';

class Search extends Component{
    constructor(){
        super(...arguments);

        this.state = {
            searchTerm: 'react'
        };
    }

    onChangeHandle(event){
        this.setState({
            searchTerm: event.target.value
        });
    }

    render(){
```

```
        return (
            <div>
                Search Term:
                <input type='search'
                    value={this.state.searchTerm}
                    onChange={this.onChangeHandle.bind(this)}/>
            </div>
        );
    }
}

export default Search;
```

Virtual DOM Under the Hood

2016年3月10日
14:32

Some assumptions include:

• When comparing nodes in the DOM tree, if the nodes are of different types (say, changing a div to a span), React is going to treat them as two different sub-trees, throw away the first one, and build/insert the second one.

• The same logic is used for custom components. If they are not of the same type, React is not going to even try to match what they render. It is just going to remove the first one from the DOM and insert the second one.

• If the nodes are of the same type, there are two possible ways React will handle this:

- If it's a DOM element (such as changing to), React will only change attributes and styles (without replacing the element tree).

- If it's a custom component (such as changing <Contact details={false} /> to <Contact details={true} />), React will not replace the component. Instead, it will send the new properties to the current mounted component. This will end up triggering a new render() on the component, and the process will reinitiate with the new result.

Refs

2016年3月10日
14:39

```
class FocusText extends Component {
    handleClick() {
        // Explicitly focus the text input using the raw DOM API.
        this.refs.myTextInput.focus();
    }
    render() {
        // The ref attribute adds a reference to the component to
        // this.refs when the component is mounted.

        return (
            <div>
                <input type="text" ref="myTextInput" />
                <input
                    type="button"
                    value="Focus the text input"
                    onClick={this.handleClick.bind(this)}
                />
            </div>
        );
    }
}
```

Prop Validation

2016年3月10日
14:47

// In generate.

```
import React, { Component, PropTypes } from 'react';
import { render } from 'react-dom';

class Greeter extends Component {
    render() {
        return (
            <h1>{this.props.salutation}</h1>
        )
    }
}

Greeter.propTypes = {
    salutation: PropTypes.string.isRequired
}
render(<Greeter salutation="Hello World" />, document.getElementById('root'));
```

// Default Prop values

```
class Greeter extends Component {
    render() {
        return (
            <h1>{this.props.salutation}</h1>
        )
    }
}
Greeter.propTypes = {
    salutation: PropTypes.string
}
Greeter.defaultProps = {
    salutation: "Hello World"
}
render(<Greeter />, document.getElementById('root'));
```

// Built-in propTypes validators

Built-in propTypes validators

2016年3月10日
14:55

Validator	Description
`PropTypes.array`	Prop must be an array.
`PropTypes.bool`	Prop must be a Boolean value (true/false).
`PropTypes.func`	Prop must be a function.
`PropTypes.number`	Prop must be a number (or a value that can be parsed into a number).
`PropTypes.object`	Prop must be an object.
`PropTypes.string`	Prop must be a string.

Validator	Description

`PropTypes.oneOfType`	An object that could be one of many types, such as
	`PropTypes.oneOfType([`
	`  PropTypes.string,`
	`  PropTypes.number,`
	`  PropTypes.instanceOf(Message)`
	`])`

`PropTypes.arrayOf`	Prop must be an array of a certain type, such as `PropTypes.arrayOf(PropTypes.number)`
`PropTypes.objectOf`	Prop must be an object with property values of a certain type, such as `PropTypes.objectOf(PropTypes.number)`

`PropTypes.shape`	Prop must be an object taking on a particular shape. It needs the same set of properties, such as
	`PropTypes.shape({`
	`  color: PropTypes.string,`
	`  fontSize: PropTypes.number`
	`})`

Table 3-3. *Special PropTypes*

Validator	Description
`PropTypes.node`	Prop can be of any value that can be rendered: numbers, strings, elements, or an array.
`PropTypes.element`	Prop must be a React element.
`PropTypes.instanceOf`	Prop must be instance of a given class (this uses JS's `instanceof` operator.), such as `PropTypes.instanceOf(Message)`.
`PropTypes.oneOf`	Ensure that your prop is limited to specific values by treating it as an enum, like `PropTypes.oneOf(['News', 'Photos'])`.

Custom PropType Validators

2016年3月10日
15:05

```
import React, { Component, PropTypes } from 'react';
import marked from 'marked';
import CheckList from './CheckList';

let titlePropType = (props, propName, componentName) => {
    if (props[propName]) {
        let value = props[propName];
        if (typeof value !== 'string' || value.length > 80) {
            return new Error(
            `${propName} in ${componentName} is longer than 80 characters`
            );
        }
    }
}

class Card extends Component {
    constructor() {...}
    toggleDetails() {...}
    render() {...}
}

Card.propTypes = {
    id: PropTypes.number,
    title: titlePropType,
    description: PropTypes.string,
    color: PropTypes.string,
    tasks: PropTypes.arrayOf(PropTypes.object)
};
export default Card;
```

Which Components Should Be Stateful?

2016年3月10日
15:27

When in doubt, follow this four-step checklist.

- Identify every component that renders something based on that state.

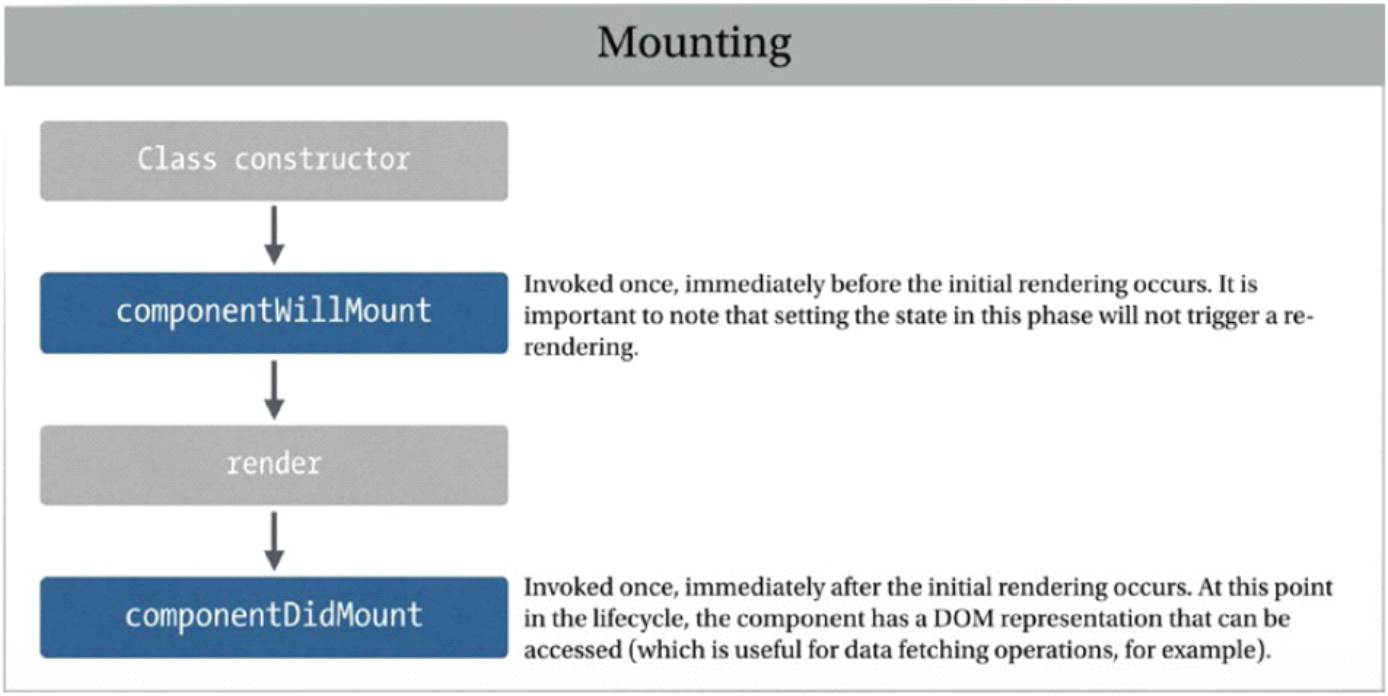
- Find a common owner component (a single component above all the components that need the state in the hierarchy).

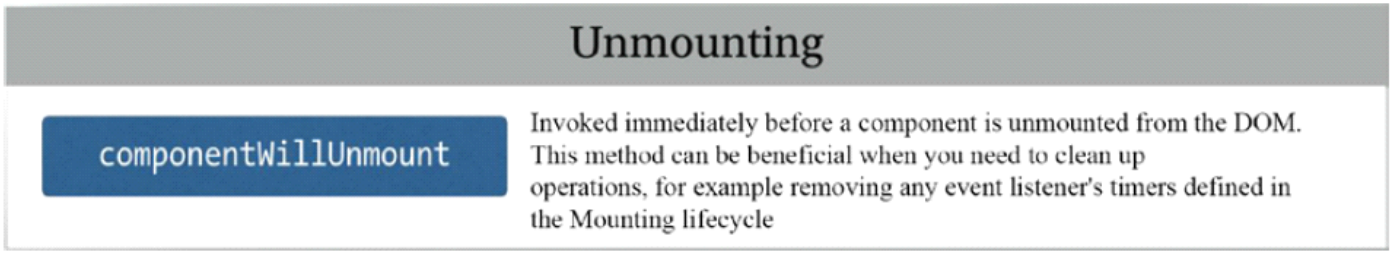
- Either the common owner or another component higher up in the hierarchy should own the state.

- If you can’t find a component where it makes sense to own the state, create a new component simply to hold the state and add it somewhere in the hierarchy above the common owner component.

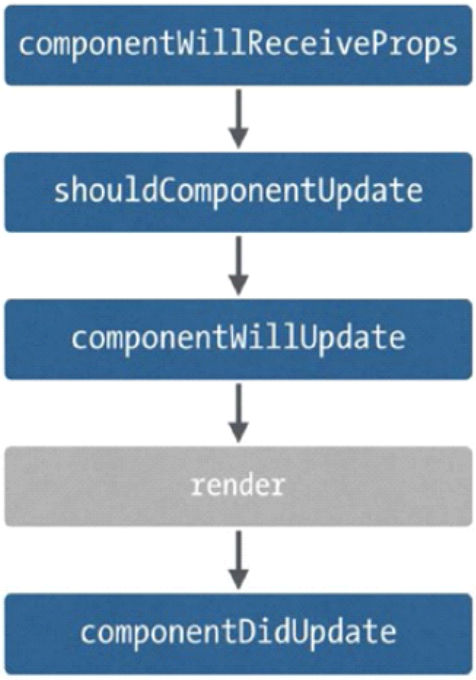
Component Lifecycle

2016年3月10日
15:36





Props Changes



Invoked when a component is receiving new props. Calling this.setState() within this function will not trigger an additional render.

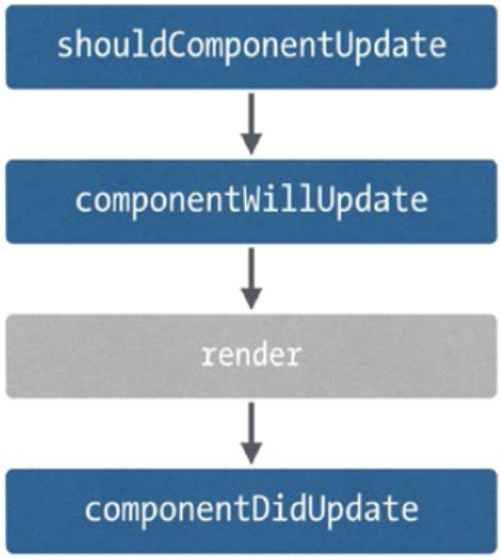
shouldComponentUpdate is a special function called before the render function and it gives the opportunity to define if a rerendering is needed or can be skipped. It is useful for performance optimizations and will be covered in detail in chapter 9.

Invoked immediately before rendering when new props or state are being received. Any state changes via this.setState are not allowed as this function should be strictly used to prepare for an upcoming update and not trigger an update itself.

Invoked immediately after the component's updates are flushed to the DOM.

State Changes

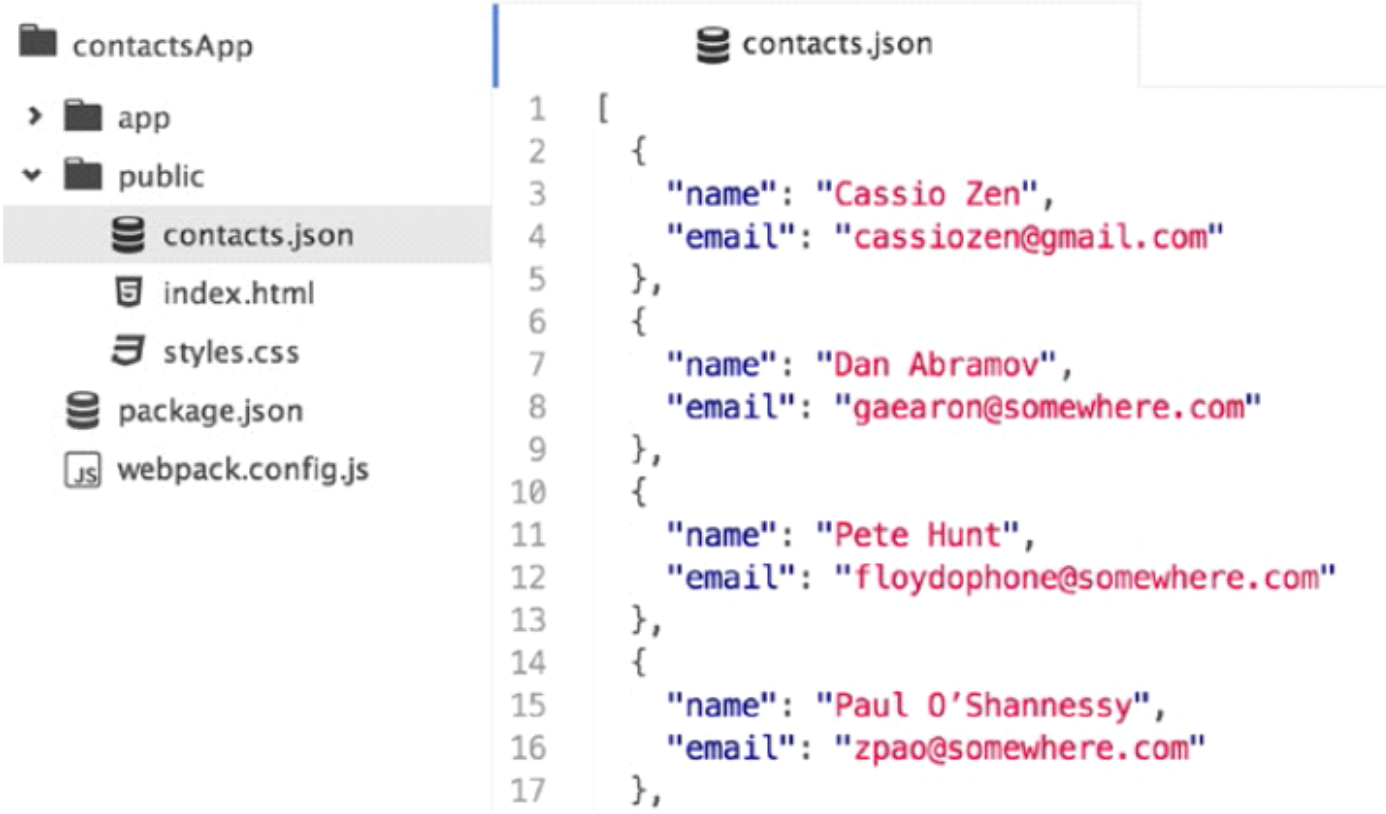
State changes fire almost the exact same lifecycle function sequence as prop changes, with one exception: There is no analogous method to componentWillReceiveProps. An incoming prop transition may cause a state change, but the opposite is not true. If you need to perform operations in response to a state change, use componentWillUpdate.



Data Fetching

2016年3月10日
15:46

```
npm install --save whatwg-fetch
```




```
import React, { Component, PropTypes } from 'react';
import { render } from 'react-dom';
import 'whatwg-fetch';

class ContactsAppContainer extends Component {
    constructor() {
        super();
        this.state = {
            contacts: []
        };
    }
    componentDidMount() {
        fetch('./contacts.json')
            .then((response) => response.json())
            .then((responseData) => {
                this.setState({ contacts: responseData });
            }).catch((error) => {
                console.log('Error fetching and parsing data', error);
            });
    }
```

```
    render() {
        return ( < ContactsApp contacts = { this.state.contacts }
            />
        );
    }
}

// No changes in any of the components bellow
class ContactsApp extends Component {
    constructor() {... }
    handleUserInput(searchTerm) {... }
    render() {... }
}

ContactsApp.propTypes = {... }

class SearchBar extends Component {
    handleChange(event) {... }
    render() {... }
}

SearchBar.propTypes = {... }

class ContactList extends Component {
    render() {... }
}

ContactList.propTypes = {... }
class ContactItem extends Component {
    render() {... }
}

ContactItem.propTypes = {... }

// You now render ContactAppContainer, instead of ContactsApp
render( < ContactsAppContainer / > , document.getElementById('root'));
```

React Immutability Helper

2016年3月10日
15:57

npm install –save react-addons-update

import update from 'react-addons-update';

	The update method accepts two parameters. The first one is the object or array that you want to update. The second parameter is an object that describes WHERE the mutation should take place and WHAT kind of mutation you want to make. So, given this simple object: `let student = {name:'John Caster', grades:['A','C','B']}`
	to create a copy of this object with a new, updated grade, the syntax for update is `let newStudent = update(student, {grades:{$push: ['A']}})` The object {grades:{$push: ['A']}} informs, from left to right, that the update function should 1. Locate the key grades (“where” the mutation will take place). 2. Push a new value to the array (“what” kind of mutation should happen).
	If you want to completely change the array, you use the command $set instead of $push: `let newStudent = update(student, {grades:{$set: ['A','A','B']}})`

	Array Indexes
	It' s also possible to use array indexes to find WHERE a mutation should happen. For example, if you want to mutate the first codeshare object (the array elopement at index 0),
	
	`let newTicket = update(originalTicket,{`
	`  codeshare: {`
	`  0: { $set: {company:'AZ', flightNo:'7320'} }`
	`  }`
	`});`

Available Commands:

Table 3-4. *React Immutability Helper Commands*

Command	Description

$push	Similar to Array's push, it adds one or more elements to the end of an array. Example:
	`let initialArray = [1, 2, 3];`
	`let newArray = update(initialArray, {$push: [4]});`
	`// => [1, 2, 3, 4]`

$unshift	Similar to Array's unshift, it adds one or more elements to the beginning of an array. Example:
	`let initialArray = [1, 2, 3];`
	`let newArray = update(initialArray, {$unshift: [0]});`
	`// => [0,1, 2, 3]`

Command	Description

`$splice`	Similar to Array's splice, it changes the content of an array by removing and/or adding new elements. The main syntactical difference here is that you should provide an array of arrays as a parameter, each individual array containing the splice parameters to operate on the array. Example:
	`let initial Array = [1, 2, 'a'];`
	`let newArray = update(initialArray, {$splice: [[2,1,3,4]]});`
	`// => [1, 2, 3, 4]`

`$set`	Replace the target entirely.

`$merge`	Merge the keys of the given object with the target. Example:
	`let ob. = {a: 5, b: 3};`
	`let newObj = update(obj, {$merge: {b: 6, c: 7}});`
	`// => {a: 5, b: 6, c: 7}`

`$apply`	Pass in the current value to the given function and update it with the new returned value. Example:
	`let obj = {a: 5, b: 3};`
	`let newObj = update(obj, {b: {$apply: (value) => value*2 }});`
	`// => {a: 5, b: 6}`

Basic Optimistic Updates Rollback

2016年3月11日
10:43

keep a reference to the old state and revert it back in case of problems

```
// Keep a reference to the original state prior to the mutations
// in case you need to revert the optimistic changes in the UI
let prevState = this.state;

fetch(..., {...})
.then((response) => {
if(!response.ok){
    // Throw an error if server response wasn't 'ok'
    // so you can revert back the optimistic changes

    // made to the UI.
    throw new Error("Server response wasn't OK")
    }
})
.catch((error) => {
    console.error("Fetch error:",error)
    this.setState(prevState);
});
```

Animation in React

2016年3月11日
11:01

npm install --save react-addons-css-transition-group

// ReactCSSTransitionGroup

Animate adding/removing	**<ReactCSSTransitionGroup transitionName="example"**
	transitionEnterTimeout={300}
	transitionLeaveTimeout={300}>
	{shoppingItems}
	</ReactCSSTransitionGroup>
	
	.example-enter
	.example-enter-active
	
	.example-leave
	.example-leave-active
	
	Every time a new item is added to the state, React will render the item with the additional className of example-enter. Immediately after, in the next browser tick, React will also attach the className example-enter-active.
	
	The same mechanism applies for removing elements from the DOM. Before removing a shopping item, React will add an example-leave className followed by example-leave-active. When the defined LeaveTimeout expires.
	
	/*animate for adding*/
	
	.example-enter {
	opacity: 0;
	transform: translateX(-250px);
	}
	
	.example-enter-active {
	opacity: 1;
	transform: translateX(0);
	transition: .3s;
	}
	
	/*animate for removing*/
	
	.example-leave {
	opacity: 1;
	transform: translateX(0);
	}
	
	.example-leave-active {
	opacity: 0;
	transform: translateX(250px);
	transition: .3s;
	}

Animate Initial Mounting	<ReactCSSTransitionGroup transitionName="example" transitionEnterTimeout={300} transitionLeaveTimeout={300} **transitionAppear={true}** **transitionAppearTimeout={300}**> {shoppingItems} </ReactCSSTransitionGroup>

```
/*animate for init mounting*/
.example-appear {
    opacity: 0;
    transform: translateX(-250px);
}

.example-appear-active {
    opacity: 1;
    transform: translateX(0);
    transition: 0.5s;
}
```

Full sample for animation

// AnimatedShoppingList.js

```
import React, { Component, PropTypes } from 'react';
import { render } from 'react-dom';
import update from 'react-addons-update';
import ReactCSSTransitionGroup from 'react-addons-css-transition-group';

class AnimatedShoppingList extends Component {
    constructor() {
        super(...arguments);

        this.state = {
            items: [
                { id: 1, name: 'milk' },
                { id: 2, name: 'yoyo' },
                { id: 3, name: 'juice' }
            ]
        };
    }

    handleChange(ev) {
        if (ev.key === 'Enter') {
            let newItem = {
                id: Date.now(),
                name: ev.target.value
            };

            ev.target.value = '';

            let nextState = update(this.state.items, { $push: [newItem] });

            this.setState({items: nextState});
        }
    }

    handleRemove(i) {
        let nextState = update(this.state.items, {
            $splice: [
                [i, 1]
            ]
        });

        this.setState({items: nextState});
    }

    render(){
     let shoppingItems = this.state.items.map((item, i) => {
          return (
               <div key={item.id}
                    className='item'
                    onClick={this.handleRemove.bind(this, i)}>{item.name}</div>
          );
```

```
      });
      return (
           <div>
                <ReactCSSTransitionGroup transitionName='example'
                                         transitionEnterTimeout={300}
                                         transitionLeaveTimeout={300}
                                         transitionAppear={true}
                                         transitionAppearTimeout={300}>
                     {shoppingItems}
                </ReactCSSTransitionGroup>

                <input type='text' value={this.state.newItem} onKeyDown={this.handleChange.bind(this)} />
           </div>
      );
     }
}

export default AnimatedShoppingList;
```

// CSS

```
/*animate for adding*/

.example-enter {
    opacity: 0;
    transform: translateX(-100px);
}

.example-enter-active {
    opacity: 1;
    transform: translateX(0);
    transition: .3s;
}


/*animate for removing*/

.example-leave {
    opacity: 1;
    transform: translateX(0);
}

.example-leave-active {
    opacity: 0;
    transform: translateX(100px);
    transition: .3s;
}


/*animate for init mounting*/

.example-appear {
    opacity: 0;
    transform: translateX(-100px);
}

.example-appear-active {
    opacity: 1;
    transform: translateX(0);
    transition: 0.5s;
}
```

Drag and Drop

2016年3月11日
14:47

use React DnD, a drag-and-drop library that lets us work in a “React way” (not touching the DOM, embracing unidirectional data flow, defining source and drop target logic as pure data, among other benefits). Under the hood, React DnD plugs into the available API.

npm install --save react-dnd react-dnd-html5-backend

The React DnD library provides three higher-order components that must be used on different components of your application: **DragSource, DropTarget, and DragDropContext**.

- **DragSource**: returns an enhanced version of the given component with the added behavior of being a “draggable” element;
- **DropTarget**: returns an enhanced component with the ability to handle elements being dragged into it;
- **DragDropContext**: wraps the parent component where the drag-and-drop interaction occurs, setting up the shared DnD state behind the scenes (it is also is the simplest to implement).

Sample:

Throttle Callbacks

2016年3月14日
11:20

A throttling function receives
two parameters, the original function you want to have throttled and wait. It returns a throttled version of the passed function that, when invoked repeatedly, will only actually call the original function at most once per every wait milliseconds. The throttling function you will implement is also smart enough to invoke the original function immediately if the calling arguments change.

```
export const throttle = (func, wait) => {
    let context, args, prevArgs, argsChanged, result;
    let previous = 0;

    return function() {
        let now, remaining;

        if (wait) {
            now = Date.now();
            remaining = wait - (now - previous);
        }

        context = this;
        args = arguments;
        argsChanged = JSON.stringify(args) != JSON.stringify(prevArgs);

        prevArgs = [...args];

        if (argsChanged || wait && (remaining <= 0 || remaining > wait)) {
            if (wait) {
                previous = now;
            }

            result = func.apply(context, args);
            context = args = null;
        }

        return result;
    }
};


class KanbanBoardContainer extends Component {
    constructor() {
            super(...arguments);

            this.updateCardStatus = throttle(this.updateCardStatus.bind(this), 500);
            this.updateCardPosition = throttle(this.updateCardPosition.bind(this), 500);
    }

}
```

Routing

2016年3月14日
13:06

// App.js

```
import React, {Component} from 'react';

import About from './About';
import Home from './Home';
import Repro from './Repro';

export default class AppRouting extends Component{
    constructor(){
        super(...arguments);

        this.state = {
            route: window.location.hash.substr(1)
        };
    }

    componentDidMount(){
        window.addEventListener('hashchange', ()=>{
            this.setState({
                route: window.location.hash.substr(1)
            });
        });
    }

    render(){
        let Child;

        switch (this.state.route) {
            case '/about':
                Child = About;
                break;
            case '/repro':
                Child = Repro;
                break;
            default:
                Child = Home;
                break;
```

```
        }

        return (
            <div>
                <header>App Basic Routing</header>
                <ul>
                    <li><a href='#/home'>Home</a></li>
                    <li><a href='#/repro'>Repro</a></li>
                    <li><a href='#/about'>About</a></li>
                </ul>
                <Child />
            </div>
        );
    }
}
```

// About

```
import React, {Component} from 'react';
import {render} from 'react-dom';

export default class About extends Component{
    render(){
        return (<h1>About</h1>);
    }
}
```

// Home

```
import React, {Component} from 'react';
import {render} from 'react-dom';

export default class Home extends Component{
    render(){
        return (<h1>Home</h1>);
    }
}
```

// Repro

```
import React, {Component} from 'react';
import {render} from 'react-dom';

export default class Repro extends Component{
```

```
    render(){
        return (<h1>Repro</h1>);
    }
}
```

React Routing

2016年3月14日
13:50

React Router provides three components to get started:
• Router and Route: Used to declaratively map routes to your application's
screen hierarchy.
• Link: Used to create a fully accessible anchor tag with the proper href. Of course this
isn't the only way to navigate the project, but usually it's the main form the end user will interact
with.

npm install --save react-router

Sample:

```
import React, { Component } from 'react';
import { render } from 'react-dom';
import { Router, Route, Link, hashHistory,IndexRoute } from 'react-router';
import About from './About';
import Repos from './Repos';
import Home from './Home';
class App extends Component {
    render() {
        return ( < div >
                < header > App < /header>
            < menu >
                    < ul >
                        < li > < Link to = "/about" > About < /Link></li >
                        < li > < Link to = "/repos" > Repos < /Link></li >
                    < /ul>
            < /menu>
            { this.props.children }
         < /div>
        );
    }
}
render(( < Router history={hashHistory } >
    < Route path = "/"     component = { App } >
        <IndexRoute component={Home}/>
        < Route path = "about"     component = { About }    />
        < Route path = "repos"     component = { Repos }    />
    < /Route>
 < /Router>
), document.getElementById('root'));
```

Some tips:

```
React.render((
<Router>
    <Route path="/" component={App}>
        <Route path="groups" components={{content: Groups, sidebar: GroupsSidebar}}/>
        <Route path="users" components={{content: Users, sidebar: UsersSidebar}}/>
    </Route>
</Router>
), element);

render() {
    return (
```

```
        <div>
            {this.props.children.sidebar}-{this.props.children.content}
        </div>
    );
}
```

Routes with Parameters

2016年3月14日
14:36

// Routes with Parameters

```
render() {
    let repos = this.state.repositories.map((repo) => (
        <li key={repo.id}>
            <Link to={"/repos/details/"+repo.name}>{repo.name}</Link>
        </li>
    ));
    return (
        <div>
            <h1>Github Repos</h1>
            <ul>
                {repos}
            </ul>
            {this.props.children}
        </div>
    );
}
```

// Setting Active Links

```
<menu>
    <ul>
        <li><Link to="/about" activeClassName="active">About</Link></li>
        <li><Link to="/repos" activeClassName="active">Repos</Link></li>
    </ul>
</menu>
```

// Props on the Route Configuration

```
<Router>
    <Route path="/" component={App}>
        <IndexRoute component={Home}/>
        <Route path="about" component={About} title="About Us" />
        <Route path="repos" component={Repos}>
        <Route path="details/:repo_name" component={RepoDetails} />
        </Route>
    </Route>
</Router>
```

Next, in the About component, you access the route configuration from this.props.route:
{this.props.route.title} // 'About Us'

Changing Routes Programmatically

2016年3月14日
14:58

Method	Description

pushState	The basic history navigation method transitions to a new URL. You can optionally pass a parameters object. Example:
	`history.pushState(null, '/users/123')`
	`history.pushState({showGrades: true}, '/users/123')`

replaceState	Has the same syntax as pushState, but it replaces the current URL with a new one. It's analogous to a redirect, because it replaces the URL without affecting the length of the history.
goBack	Go back one entry in the navigation history.
goForward	Go forward one entry in the navigation history.
Go	Go forward or backward in the history by n or -n
createHref	Makes a URL, using the router's config.

For this purpose, React Router automatically injects its history object into all components that it mounts. The history object is responsible for managing the browser' s history stack

this.props.history.pushState(null,'/error');

waitFor: Coordinating Store Update Order

2016年3月15日
13:44

In big Flux projects dealing with multiple stores, you may come to a situation where one store depends on data from another store. The Flux dispatcher provides a method called waitFor() to manage this kind of dependency; it makes the store wait for the callbacks from the specified stores to be invoked before continuing execution.

https://github.com/ichenzhifan/BankAccount.git

Asynchronous Data Fetching

2016年3月15日
14:00

How the Reconciliation Process Works

2016年3月16日
9:36

Whenever you change the state of a React component, it triggers the reactive re-rendering process. React will construct a new virtual DOM representing your application’s state UI and perform a diff with the current virtual DOM to work out what DOM elements should be mutated, added, or removed. This process is called reconciliation.

Batching:
In React, whenever you call setState on a component, instead of updating it immediately React will only mark it as “dirty”. That is, changes to your component’s state won’t take effect immediately; React uses an event loop to render changes in batch.

Sub-Tree Rendering:
When the event loop ends, React re-renders the dirty components as well as their children, al the nested components, even if they didn't change, will have their render method called.
This may sound inefficient, but in practice it is actually very fast, because React is not touching the actual DOM, all this happens in the in-memory virtual DOM.

shouldComponentUpdate: Before re-rendering a child component, React will always invoke its shouldComponentUpdate method. By default, shouldComponentUpdate always returns true, but if you reimplement it and return false, React will skip re-rendering for this component and its children.

React Perf:

npm install --save react-addons-perf

Validator	Description
`Perf.start()` and `Perf.stop()`	Start/stop the measurement. The React operations in between are recorded for analyses below.
`Perf.printInclusive()`	Prints the overall time taken.
`Perf.printExclusive()`	“Exclusive” times don’t include the time taken to mount the components: processing props, calling `componentWillMount` and `componentDidMount`, etc.
`Perf.printWasted()`	“Wasted” time is spent on components that didn’t actually render anything; in other words, the render stayed the same, so the DOM wasn’t touched.

1. shouldComponentUpdate(nextProps, nextState){
   // Don't trigger a re-render unless the digit value was changed.
   return nextProps.value !== this.props.value;
}

2. shallowCompare Add-on
   • The component where you want to apply the shallow compare is “pure” (in other words, it renders the same result given the same props and state).
   • You are using immutable values or React’s immutability helper to manipulate state.

   npm install --save react-addons-shallow-compare

   import shallowCompare from 'react-addons-shallow-compare';

   shouldComponentUpdate(nextProps, nextState) {

```
    return shallowCompare(this, nextProps, nextState)
}
```

Testing React Components

2016年3月16日
13:16

```
{
    "name": "react-components-test",
    "version": "1.0.0",
    "description": "",
    "main": "src/App.js",
    "scripts": {
        "test": "jest"
    },
    "jest": {
        "scriptPreprocessor": "<rootDir>/node_modules/babel-jest"
    },
    "author": "",
    "license": "ISC",
    "devDependencies": {
        "babel-core": "^6.6.4",
        "babel-preset-es2015": "^6.6.0",
        "babel-jest": "^9.0.3",
        "jest-cli": "^0.9.2"
    }
}
```

Get Started:

```
// sum.js
let sum = (value1, value2) => (
    value1 + value2
)
export default sum;
```

```
// In the __tests__ folder, create a sum-test.js file
jest.autoMockOff();
describe('sum', function() {
    it('adds 1 + 2 to equal 3', function() {
        var sum = require('../sum').default;
        expect(sum(1, 2)).toBe(3);
    });
});
```

React Test Utilities:

```
npm install --save-dev react-addons-test-utils
```

renderIntoDocument	let component = TestUtils.renderIntoDocument(<MyComponent />); You can then use findDOMNode() to access the raw DOM element and test its values	
ReactDOM.findDOMNode		
findRenderedDOMComponentWithTag		
TestUtils.Simulate.change	Simulating Events	

```
// CheckboxWithLabel-test.js

'use strict';

jest.unmock('../components/CheckboxWithLabel');

import React from 'react';
import ReactDOM from 'react-dom';
import TestUtils from 'react-addons-test-utils';

import CheckboxWithLabel from '../components/CheckboxWithLabel';

describe('CheckboxWithLabel', () => {
```

```
    // Render a checkbox with label in the document
    let checkbox = TestUtils.renderIntoDocument( < CheckboxWithLabel labelOn = 'On'
        labelOff = 'Off' / > );

    let checkboxNode = ReactDOM.findDOMNode(checkbox);

    /**
     * 1. verify the it's off by default
     */
    it('default to off label', () => {
        expect(checkboxNode.textContent).toEqual('Off');
    });

    /**
     * 2. defaults to unchecked
     */
    it('default to off label', () => {
     let checkboxElement = TestUtils.findRenderedDOMComponentWithTag(checkbox, 'input');
        expect(checkboxElement.checked).toBe(false);
    });

    /**
     * 3. Simulate a click and verify that it is now On
     */
    it('default to off label', () => {
        TestUtils.Simulate.change(
            TestUtils.findRenderedDOMComponentWithTag(checkbox, 'input')
        );

        expect(checkboxNode.textContent).toEqual('On');
    });
});
```

Shallow Rendering

Shallow rendering is a new feature introduced in React 0.13 that lets us output a component's virtual tree without generating a DOM node. This way we can inspect how the component would be built, but without actually rendering it. The advantages of this approach over using renderIntoDocument includes removing
the need for a DOM in the test environment (which is consequentially much faster), and the fact that is allows us to test React components in true isolation from other component classes. It does this by allowing us to test the return value of a component's render method, without instantiating any subcomponents.