Node Program

Lesson 2: Developing with React.js

React.js version: 15
Last updated: Nov 2016

Lists

What are Lists

Lists are often use on webpages. They consist of many similar items wrapped in a parent element. Examples include:

» Menus

» Ordered and unordered lists

» Grids

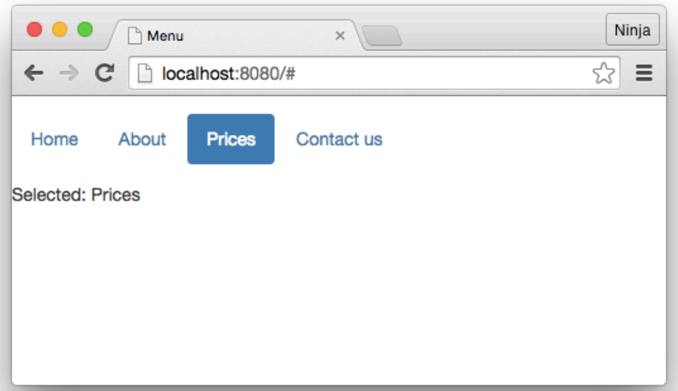
List Implementation

The easiest way to implement a list in React.js is to use array and map(), e.g.,

```
render() {
  return (
    <ul>
      {this.props.items.map((value, index) =>{
        return <li>{value}</li>
      })}
    </ul>
  )
}
```

Menu Example

This example renders list of menu items and uses Twitter Bootstrap.



Selected: Prices

The source code: `/menu` or http://plnkr.co/edit/dyTMGBoTIXckKediycQl?p=preview.

Props Features

Default Props

The `getDefaultProps` method is invoked once before the instance is created. The properties in the returned object will be set on `this.props` if they are not set by the parent.

Default Props Example

```
class Button extends React.Component {
  render(){
    return <button >{this.props.buttonLabel}</button>
  }
}
Button.defaultProps = {
  buttonLabel: 'lorem ipsum'
}
```

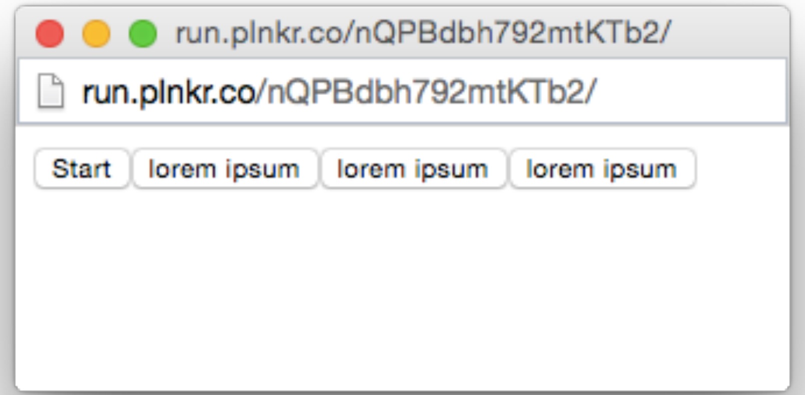
Parent With a Missing Props

This parent component `Content` is missing props on 3 Button components:

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <Button buttonLabel="Start"/>
        <Button />
        <Button />
        <Button />
      </div>
    )
  }
}
```

Default Props Demo

If the prop is missing the default value is used:


run.plnkr.co/nQPBdbh792mtKTb2/
run.plnkr.co/nQPBdbh792mtKTb2/
Start
lorem ipsum
lorem ipsum
lorem ipsum

Source code: `/default-props` or http://plnkr.co/edit/7JC7qg3Ka87i5ObETV7r?p=preview.

Prop Types

Prop Types

You can set the prop types on React.js classes. If the type doesn't match and you're in development mode, then you'll get a warning in the console.

Note: React.js suppresses this warning in production mode (more on the dev vs. prod later).

Front-end Validation Warning

Warning: Never rely on the front-end user input validation. Use it only for better User Experience (UX) and check everything on the server-side.

Development vs. Production

The way React.js team defines the development mode is when you're using un-minified version, and the production mode is when you're using minified version.

We provide two versions of React: an uncompressed version for development and a minified version for production. The development version includes extra warnings about common mistakes, whereas the production version includes extra performance optimizations and strips all error messages.

Validating Props

Use the `propTypes` property with the object that has props as keys and types as values. React.js types are in the React.PropTypes object. For example:

- `React.PropTypes.string`
- `React.PropTypes.number`
- `React.PropTypes.bool`
- `React.PropTypes.object`

Prop Type Example

This class will have an optional `title` prop of the string type:

```
class Button extends React.Component {
  //...
}
Button.propTypes = {
  title: React.PropTypes.string
}
```

`/prop-types` or http://plnkr.co/edit/fK74C6wrQeF5uRSno6Dy?p=preview

Required Prop Type

To make a prop required just add `isRequired` to the type. This class will have a `handler` prop of function type required:

```
class Button extends React.Component {
  //...
}
Button.propTypes = {
  handler:  React.PropTypes.func.isRequired
}
```

Prop Types Demo

The example in the `module2/prop-types` folder will produce these warnings:

```
Warning: Failed propType: Required prop `handler` was not specified in `Button`. Check the render method of `Content`.
Warning: Failed propType: Invalid prop `title` of type `number` supplied to `Button`, expected `string`. Check the render method of `Content`.
```

Only the unminifed version of React.js shows the warnings—development mode.

Custom Validation

Just return an instance of `Error`. For example, this code validate email with Regular Expression:

```
email(props, propName, componentName) {
  let emailRegularExpression = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i
  if (!emailRegularExpression.test(props[propName])) {
    return new Error('Email validation failed!')
  }
}
```

Additional Prop Types

There are many additional types and helper methods. Please refer to the documentation:

https://facebook.github.io/react/docs/reusable-components.html#prop-validation

Higher-Order Components

```
const LoadWebsite = (Component) => {
  class _LoadWebsite extends React.Component {
    constructor(props) {
      super(props)
      this.state = {label: 'Run'}
      this.state.handleClick = this.handleClick.bind(this)
    }
    // ...
    render() {
      console.log(this.state)
      return <Component {...this.state} {...this.props} />
    }
  }
  return _LoadWebsite
}
```

Rendering Children

Children Components

Instance A:

```
<Content>
  <h1>React.js</h1>
  <p>Rocks</p>
</Content>
```

Instance B:

```
<Content>
  <img src="https://facebook.github.io/react/img/logo.svg"/>
</Content>
```

Children Prop

There's an easy way to render all the children with `{this.props.children}`.

Children Prop Example

For example, we add a `div` and pass along children elements:

```
class Content extends React.Component {
  render() {
    return (
      <div>
        {this.props.children}
      </div>
    )
  }
}
```

Parent

The parent has children `<h1>` and `<p>`:

```
ReactDOM.render(
  <Content>
    <h1>React.js</h1>
    <p>Rocks</p>
  </Content>,
  document.getElementById('content')
)
```

Source code: `/children` or http://plnkr.co/edit/ykC29RjWxxmblI2HyfiV?p=preview.

Children is an Array

Children is an Array if n>1. You can access individual elements link this:

```
{this.props.children[0]}
{this.props.children[1]}
```

Children Truthy Check

There's only one element, this.props.children is NOT an array.

Use `React.Children.count(this.props.children)` to get the accurate count.

More helpers: https://facebook.github.io/react/docs/top-level-api.html#react.children

Forms

Form Elements

- input
- textarea
- option

Synthetic Event

Capture and Bubbling

Capture (first)

```
onClickCapture = {this.handleClickCapture}
```

Bubbling (later):

```
onClick = {this.handleClick}
```

Form Events

Form support these events:

- onChange
- onInput
- onSubmit

Form Elements

`<input>`, `<textarea>`, and `<option>` are special because they have mutable props (remember props are usually immutable)—value, checked and selected.

Capturing Enter

You can use `onKeyUp` event to capture enter and trigger the submission of the data:

```
keyup(e) {
  if (e.keyCode == 13) return this.sendData()
},
```

in render:

```
<form onKeyUp={this.keyup}>
```

Controlled Components

Controlled component means that the value prop is set. Typically it's tied to the `this.state.value`:

```
render() {
  let value = this.state.value
  return <input type="text" value={value} onChange={this.handleChange} />
}
```

Benefit of Controlled Components

Your element's internal state value will always be the same as the representation. It keeps things simple and in sync with React philosophy.

Controlled Component Example

For example, if we have an account number input field it needs to accept only numbers. To limit the input to number (0-9) we can use a controlled component which will weed out all non-numeric values:

```
//...
change(e) {
  this.setState({value: e.target.value.replace(/[^0-9]/ig, '')})
}
//...
```

Controlled Component Example

```
class Content extends React.Component {
  constructor() {
    this.state = {value: ''}
  }
  //...
  render() {
    return <div>
      Account Number: <input type="text"
        onChange={this.change}
        placeholder="123456"
        value={this.state.value}/>
      <br/>
      <span>{this.state.value.length>0 ? 'You entered: ' +
       this.state.value: ''}</span>
    </div>
  }
}
//...
```

Default Values

This is an anti-pattern because user will never be able to change the value in this controlled component:

```
render() {
   return <input type="text" value="Hello!" />
 }
```

The right pattern is to use `defautValue` prop for setting default values:

```
render() {
   return <input type="text" defaultValue="Hello!" />
 }
```

Try it

Source Code: `/controlled` or http://plnkr.co/edit/gfeCl8JPXqgJbG13Oc45?p=preview.

Uncontrolled Components

Uncontrolled component simply means that the value prop is not set. To capture the changes from an an uncontrolled component, use `onChange`. For example,

```
  render() {
    return <div>
      <input type="text"
        onChange={this.change}
        ref="textbox"
        placeholder="Hello!" />
      <span>{this.state.value}</span>
    </div>
  }
})
```

Refs

What is Refs

Refs are used to get the DOM element of a React.js component:

1. `render` has the ref attribute: `<input ref="email" />`
2. In code (e.g., event handler), access the instance via `this.refs.NAME` as in: `this.refs.email`

Refs' DOM

You can access the component's DOM node directly by calling `React.findDOMNode(this.refs.NAME)`, e.g.,

```
React.findDOMNode(this.refs.email)
```

Capturing Uncontrolled Components

This is the `change` method that updates the state:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {value: ''}
  }
  change(e) {
    console.log(e.target.value)
    console.log(React.findDOMNode(this.refs.textbox).value)
    this.setState({value: e.target.value})
  }
  render() {
    // ...
  }
}
```

Source code `/uncontrolled` or http://plnkr.co/edit/p1baE65AwKm52Yh6Lh6K?p=preview.

Style Attribute

CSS Style Attribute

You can set the style attribute using JS object literal or JSON and camel case (backgroundImage instead of background-image). For example, the first {} is for object and the second {} is for rendering:

```
<div style={{borderColor: 'blue', fontFamily: 'Arial'}}>
```

Style with Object

Of course, we can define the style as an object and use it in JSX with {}:

```
class Content extends React.Component {
  render() {
    let style = {border: '1px solid blue'}
    return (
      <div style={style}>
        <h1>Hello!</h1>
      </div>
    )
  }
}
```

Source code: `/style` or http://plnkr.co/edit/8OjJ1vBPH7sN9pNf065G?p=preview.

Summary

Summary

- Lists with the `map` method
- HOC is a function to extend a component
- Controlled vs. uncontrolled components
- Prop types
- Refs

Summary (Cont.)

- Prop validation with the `propTypes` property
- Development vs. production mode
- Passing children elements with `this.props.children`
- Inline style attribute with a JSON object and `{}`

Questions and Exercises

❓ ✋👍

Redux

```
const React = require('react')
const { render } = require('react-dom')
const { Provider } = require('react-redux')
const { createStore } = require('redux')
const reducers = require('./modules')
const routes = require('./routes')

module.exports = render((
  <Provider store={createStore(reducers)}>
    {routes}
  </Provider>
), document.getElementById('app'))
```

ch14 of React Quickly on GitHub azat-co/react-quickly

React Router

```
const ReactDOM = require ('react-dom')
const ReactRouter = require('react-router')
const {withRouter} = require('react-router')

ReactDOM.render((
  <Router history={hashHistory}>
    <Route path="/" component={Content} >
      <Route path="/about" component={About} />
      <Route path="/posts" component={Posts} posts={posts}/>
      <Route path="/posts/:id" component={Post}  posts={posts}/>
      <Route path="/contact" component={withRouter(Contact)} />
    </Route>
    <Route path="/login" component={Login}/>
  </Router>
), document.getElementById('content'))
```

ch13 of React Quickly on GitHub azat-co/react-quickly

Project: Message Board: React.js + Axios + Express + MongoDB

1. Data: Express, MongoDB, Universal JS, Redux
2. Setup: JSX, npm, Babel and Webpack

Demo

Project: Message Board

Source code: `code/react/board`

To run the project:

```
$ npm install
$ npm start
```

Navigate to http://localhost:3000

Workshop: Message Board 🔨🖥😁

1. Make it work (mongod?)
2. Add remove/delete/x icon/button to *each* message in views
3. Add a REST endpoint to delete
4. Add AJAX call to remove message
5. Deploy to cloud: Heroku, now.sh, AWS, etc.