Node Program

Lesson 2: Developing with React.js

React.js version: 0.14.3
Last updated: Jan 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

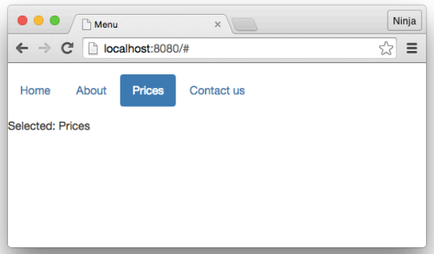
List Implementation

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

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

Menu Example

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



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

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)
```

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

```
var Button = React.createClass({
  getDefaultProps: function () {
    return {buttonLabel: 'lorem ipsum'}
  },
  render: function(){
    return <button >{this.props.buttonLabel}</button>
  }
})
```

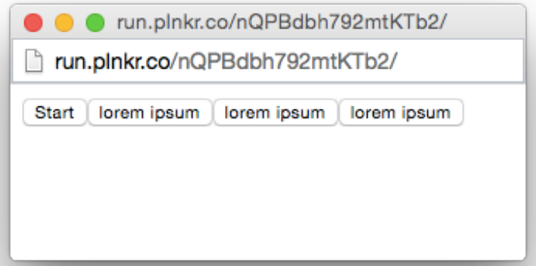
Parent With a Missing Props

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

```
var Content = React.createClass({
  render: function() {
    return (
      <div>
        <Button buttonLabel="Start"/>
        <Button />
        <Button />
        <Button />
      </div>
    )
  }
})
```

Default Props Demo

If the prop is missing the default value is used:



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

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:

```
var Button = React.createClass({
  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:

```
var Button = React.createClass({
  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: function(props, propName, componentName) {
  var 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

Mixins

Defining Mixins

Mixins allows to reuse code. You can share methods between React.js components.

Declaring a Mixin

To define a mixin just create an object with properties. For example to set the label state, define the `click` method as well as a mounting event:

```
var Mixin = {
  getInitialState: function(){
    return {text: 'lorem ipsum'}
  },
  click: function(e) {
    console.log(e.target)
  },
  componentDidMount: function(){
    console.log(React.findDOMNode(this))
  } //...
}
```

Mixins Usage

To use mixins, simply add the variable enclosed in a `[]` as the value of `mixins` property. For example, a `Component` element will use the properties defined in `Mixin`:

```
var Component = React.createClass({
  mixins: [Mixin],
  //...
  render: function() {
    return <span onClick={this.click}>{this.state.text}</span>
  }
})
```

Mixins Example

Let's share some properties among several components such as a button, link and an image. Maybe it's a menu or controls that have the same state and functionality, but different rendering.

Source code: `/mixins` or http://plnkr.co/edit/sNvJIpmh3yONyjWPR48c?p=preview.

Mixin Example

For example, this mixin `RunMixin` has several properties that set the label state, define the `click` method as well as mounting events:

```
var RunMixin = {
  getInitialState: function(){
    return {label: 'Run'}
  },
  componentWillMount: function() {
    console.log('component will mount')
  },
  click: function(e) {
    var iframe = document.getElementById('frame').src = 'http://reactjs.com'
  },
  componentDidMount: function(){
    console.log(React.findDOMNode(this))
  }
}
```

Mixins Example

The button component is using the `RunMixin` and magically gets its properties (e.g., `this.click`):

```
var Button = React.createClass({
  mixins: [RunMixin],
  render: function() {
    return <button onClick={this.click}>{this.state.label}</button>
  }
})
```

Mixins Example

The link component is using the same `RunMixin` and magically gets its properties:

```
var Link = React.createClass({
  mixins: [RunMixin],
  render: function(){
    return <a onClick={this.click} href="#">{this.state.label}</a>
  }
})
```

Mixins Example

The image component is using the same `RunMixin` and magically gets its properties:

```
var Logo = React.createClass({
  mixins: [RunMixin],
  render: function(){
    return <img onClick={this.click} width="40" src="logo.png" href="#"/>
  }
})
```

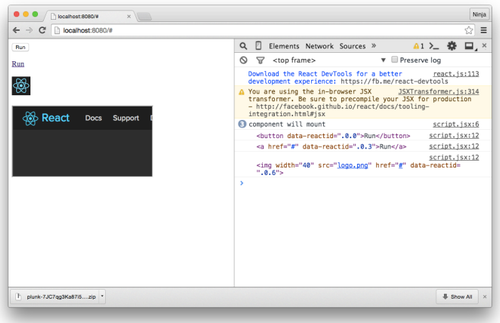
Mixins Example

The parent components renders the elements:

```
var Content = React.createClass({
  render: function() {
    return (
      <div>
        <Button />
        <br />
        <br />
        <Link />
        <br />
        <br />
        <Logo />
        <br />
        <br />
        <iframe id="frame" src=''/>
      </div>
    );
  }
})
```

Mixins Demo

The three elements will have the same functionality (loads the ifame):



Note: You cannot use React.js mixins with ES6 classes.

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:

```
var Content = React.createClass({
  render: function() {
    return (
      <div>
        {this.props.children}
      </div>
    )
  }
})
```

Parent

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

```
React.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

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

in render:

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

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

Capturing Uncontrolled Components

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

```
var Content = React.createClass({
  getInitialState: function(){
    return {value: ''}
  },
  change: function(e) {
    console.log(e.target.value)
    console.log(React.findDOMNode(this.refs.textbox).value)
    this.setState({value: e.target.value})
  },
```

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

Controlled Components

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

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

Controlled Component Example

```
var Content = React.createClass({
  getInitialState: function(){
    return {value: ''}
  },
  //...
  render: function() {
    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: function() {
   return <input type="text" value="Hello!" />
 }
```

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

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

Try it

Source Code: `/controlled` or http://plnkr.co/edit/gfeCl8JPXqgJbG13Oc45?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={{border:"1px solid blue"}}>
```

Style with Object

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

```
var Content = React.createClass({
  render: function() {
    var 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

→ Mixins with the `mixins: [Name]` property

→ 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 `{}`

Demo

Project: Message Board

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

Source code: `http://bit.ly/1StYYYy` or http://bit.ly/1StYYYy. To run the project:

```
$ npm install -g gulp
$ gulp
```

Navigate to http://localhost:3000

1. Copy `gulpfile.js` and `package.json`
2. Run `npm i`

Questions and Exercises

Workshop 🔨🖥😁

```
$ npm i -g thinking-in-react
```