

React Foundation

Module 3: User Input



Azat Mardan @azat_co



Component Methods

Calling Methods

It's possible to invoke components methods from the `{ }` interpolation:

```
class Content extends React.Component {  
  getA() {  
    return 10  
  }  
  render() {  
    return (  
      <div>  
        <p>This value is return by the method: {this.getA()} </p>  
      </div>  
    )  
  }  
}
```

Component Events

Events

Components have normalized (cross-browser) events such as

**onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp**

Declaring Events

React.js is declarative, not imperative. So we won't attach event like we would do with jQuery, instead we declare them in the JSX and classes:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {counter: 0}
    this.click = this.click.bind(this)
  }
  click(event) {
    this.setState({counter: ++this.state.counter})
  }
  render() {
    // ...
  }
}
```

Button onClick Event

The button has the `onClick={this.click}`.

The name must match the method of the Content component class:

...

```
render() {  
  return (  
    <div>  
      <button onClick={this.click}>Don't click me {this.state.counter} times!</button>  
    </div>  
  )  
}  
})
```

<http://plnkr.co/edit/owbmK9?p=preview>

Where to Put logic

In this example, click event handler was in the parent element. You can put the event handler on the child itself, but using parent allows you to exchange info between children components.

Let's have a button:

```
class ClickCounterButton extends React.Component {  
  render() {  
    return <button onClick={this.props.handler}>Don't click me! </button>  
  }  
}
```

Exchanging Props Between Children

This is a new component which displays value prop:

```
class Counter extends React.Component {  
  render() {  
    return <span>Clicked {this.props.value} times.</span>  
  }  
}
```

Parent Component

The parent component provides props one of which is a handler:

```
class Content extends React.Component {
  constructor(props) {
    super(props)
    this.state = {counter: 0}
    this.click = this.click.bind(this)
  }
  click(event) {
    this.setState({counter: ++this.state.counter})
  }
  render() {
    return (
      <div>
        <ClickCounterButton handler={this.click}/>
        <br/>
        <Counter value={this.state.counter}/>
      </div>
    )
  }
}
```

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(event) {  
    if (event.keyCode === 13) return this.sendData()  
}
```

in render:

```
<form onKeyUp={this.keyup.bind(this)}>
```

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(event) {
```

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

```
//...
```

Controlled Component Example

```
class Content extends React.Component {
  constructor() {
    this.state = {value: ''}
    this.change = this.change.bind(this)
  }
  //...
  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 `defaultValue` prop for setting default values:

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

Try it

Controlled demo: <http://plnkr.co/edit/ouADpl?p=preview>.

Uncontrolled Components

Uncontrolled component simply means that the value prop is not set. To capture the changes from 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 `ReactDOM.findDOMNode(this.refs.NAME)`, e.g.,

`ReactDOM.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: ''}
    this.change = this.change.bind(this)
  }
  change(event) {
    console.log(event.target.value)
    console.log(ReactDOM.findDOMNode(this.refs.textbox).value)
    this.setState({value: event.target.value})
  }
  render() {
    // ...
  }
}
```

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

Uncontrolled component demo: [http://plnkr.co/edit/zmmhXU?
p=preview](http://plnkr.co/edit/zmmhXU?p=preview).

Let's put together browser events,
component composition, props
and states! 🐎

Timer Project

Source code: `code/timer`

Timer

5 seconds

10 seconds

15 seconds

Time left: 11

ElementsConsoleSourcesNetworkTimelineProfilesApplicationReact

Trace React Updates

Highlight Search

Use Regular Expressions

<TimerWrapper>

<div className="row-fluid">

<h2>Timer</h2>

<div className="btn-group" role="group">

<Button time="5" startTimer=bound startTimer()>...</But

<Button time="10" startTimer=bound startTimer()>...</Bu

<Button time="15" startTimer=bound startTimer()>...</Bu

</div>

<Timer timeLeft=11>...</Timer>

<audio id="end-of-time" src="flute_c_long_01.wav" prelo

</div>

</TimerWrapper>

<TimerWrapper> (\$r in the console)

Props

Empty object

State

timeLeft: 11

timer: 3

TimerWrapper

Search by Component Name

© NodeProgram.com, Node.University and Azat Mardan 2017

34

Timer Overview

- >> 3 components: 2 presentational and one smart
 - >> 3 buttons from a single component
 - >> Webpack, JSX, Babel, modules, npm
 - >> Static (`node-static`) is not included in `package.json`
- `code/timer`

Demo

Workshop: Timer

1. Move `Timer` and `Button` to separate files
2. Create a pause and resume buttons (could be one button as a toggle)
3. Create a reset button (separate component)
4. Create a custom timer with an input field (separate component)
5. Change to minutes, not seconds