# State

Ustun Ozgur

2015-11-03 Tue

# State

- Handling apps with changing state
- Two important methods: `getInitialState` and `setState`
- `setState`: merges the input to current state
- Also `replaceState`, replaces the state with the input
- state can be accessed from render as `this.state`

# Example: Counter

```
var Counter = React.createClass({

  getInitialState: function () {
    return {counter: 0};
  },
  increment: function () {
    this.setState({counter: this.state.counter + 1});
  },
  render: function () {
    return <div onClick={this.increment}>
      Counter value is {this.state.counter}
    </div>;

  }

});
```

# Overview

- We have ‘onClick‘ handler that calls ‘setState‘ with state incremented by 1.
- This is the essence of state management with React.

# Passing State as Props to Children

- An owner component can pass its state as props to its children
- One component's state is another's props.

# Example: Counter in Counters

```
var Counter = React.createClass({
  render: function () {
    return <div>
      Counter value is {this.props.counter}
      <button onClick={this.props.click}>Increment</button>
      </div>
  }

});
```

# Example: Counters Component

```
var Counters = React.createClass({
  getInitialState: function () {
    return {counter1: 10, counter2: 0};  },
  incrementCounter1: function () {
    this.setState({counter1: this.state.counter1 + 1,
                   counter2: this.state.counter2 - 1})  },
  incrementCounter2: function () {
    this.setState({counter1: this.state.counter1 - 1,
                   counter2: this.state.counter2 + 1})  },
  render: function () {
    return <div>
      <Counter click={this.incrementCounter1}
       counter={this.state.counter1}/>
      <Counter click={this.incrementCounter2}
       counter={this.state.counter2}/>
      </div>  }})
```

# A More Succint Version using bind

```
var Counters = React.createClass({
  getInitialState: function () {
    return {counter1: 10, counter2: 0};    },
  incrementCounter: function (increment) {
    this.setState({
      counter1: this.state.counter1 + increment,
      counter2: this.state.counter2 - increment})    },
  render: function () {
    return <div>
      <Counter
       click={this.incrementCounter.bind(this, 1)}
       counter={this.state.counter1}/>
      <Counter
       click={this.incrementCounter.bind(this, -1)}
       counter={this.state.counter2}/>
      </div>    }})
```

# Summary

- Child components never update their props directly.
- They inform the parent component that some event has happened via functions passed as props.
- The parent modifies its state.
- React handles propagating the state as props downward.
- Everything is re-rendered.

# Exercise:

- Modify the todo app so that the list items are stored in a store variable instead of being passed as props.
- Implement toggling of completed status of todo items.
- Hint: Should the todo items be a list of strings as it was previously? Is a list of objects better?
- Implement filtering the todo list by completed items.
- Hint: Store whether the filter is in effect by saving it in a state variable.
- Implement removing a todo item.
- Implement showing the number of remaining items.