React: Thinking State First

Doug Neiner
github.com/LeanKit-Labs
Open Source Goodness
React: Thinking State First

Doug Neiner
Why another library?

Why should I care?
Why React.js?

- Cohesive Templates and Code
- Enforced Predictable Behavior
- Synthetic Event System
- Performance
- Can be integrated gradually
Our Canvas

• It can stretch to any size
• It may behave or appear differently to different people
• It can become a performance bottleneck when you mess with it
• So... how do we dynamically modify this canvas?
```javascript
var ct = document.querySelector( " .cart-count" );
ct.innerHTML = "5";
```
jQuery

$( ".cart-count" ).text( "5" );
$.each( someArray, function ( item ) {
  $( "#cart-items" ).append("<li class='cart-item'>" + item.title + "</li>" );
});
var $ul = $( "<ul/>" );
$.each( someArray, function ( item ) {
    $ul.append(
        $( "<li />", {
            "class": "cart-item", text: item.title
        } )
    );
});
$container.html( $ul );
Rinse. Repeat.
We need a better way
<ul>
  <% __.each(items, function (item) { %>
  <li class="cart-item"><%- item.title %></li>
  <% } ) %>
</ul>
Templates

• When do you render them?
• How often?
• How do you get data out of the rendered DOM?
Too Much Thinking

We need a better way
Binding + Templates

- Knockout.js
- Ember
- Angular
These Tools Are Solving The Same Problem
The “Template” Part

- Special attributes: `data-bind, ng-repeat`
- Learning template syntax: `{{#each}}`}
- Templates maintained separately from the code that uses them...
Templates maintained separately from the code that uses them...
We strongly believe that components are the right way to separate concerns rather than "templates" and "display logic." We think that markup and the code that generates it are intimately tied together. Additionally, display logic is often very complex and using template languages to express it becomes cumbersome. – React Docs (emphasis added)
We strongly believe that components are the right way to separate concerns rather than "templates" and "display logic." We think that markup and the code that generates it are intimately tied together. Additionally, display logic is often very complex and using template languages to express it becomes cumbersome. – React Docs (emphasis added)
A React Component (JSX)

React.createClass({
    render: function() {
        return <div className="cart">
            <span className="cart-count">
                { this.props.items.length }
            </span>
            <CartItems items={this.props.items} />
        </div>;
    }
});
A React Component (JSX)

React.createClass({
  render: function() {
    return <div className="cart">
      <span className="cart-count">
        { this.props.items.length }
      </span>
      <CartItems items={ this.props.items } />
    </div>;
  }
})
A React Component (JS)

React.createClass({
    render: function render() {
        return React.createElement( "div", { className: "cart" },
            React.createElement( "span", { className: "cart-count" },
            this.props.items.length
        ),
            React.createElement( CartItems, { items: this.props.items } )
    }
});
The “Template” Part

• Special attributes: `key`, `ref`

• Learning template syntax: `{}` + JS

• Templates maintained *with* the code that uses them.
Cohesion between our code and our templates in a language we already use.

Key Benefit
Predictable Behavior
Kindly Enforced by React.js
Props
- Options
- Attributes

State
- Variables
Props
Externally Controlled

State
Internally Controlled
Given the same props and state, the component will render the same thing. **Every time.**
<NewsItem title="Welcome" content="..." />

var NewsItem = React.createClass({
    render: function() {
        return (
            <div className="newsItem">
                <h2>{this.props.title}</h2>
                <p>{this.props.content}</p>
            </div>
        );
    }
});
<NewsItem title="Welcome" content="..." />

var NewsItem = React.createClass({
    render: function() {
        return (
            <div className="newsItem">
                <h2>{this.props.title}</h2>
                <p>{this.props.content}</p>
            </div>
        );
    }
});
var Menu = React.createClass( {
    getInitialState: function () {
        return { open: false };
    },
    toggleOpen: function() {
        this.setState({ open: !this.state.open });
    },
    render: function() {
        return <div onClick={ this.toggleOpen }>
            { this.state.open ? "Open" : "Closed" }
        </div>;
    }
});
var Menu = React.createClass( {
  getInitialState: function () {
    return { open: false }; 
  },
  toggleOpen: function() {
    this.setState({ open: !this.state.open });
  },
  ... 
  render: function() {
    return <div onClick={ this.toggleOpen }>
      { this.state.open ? "Open" : "Closed" } 
    </div>;
  }
} );
var Menu = React.createClass( {
  getInitialState: function () {
    return { open: false }; 
  },
  toggleOpen: function () {
    this.setState({ open: !this.state.open });
  },
  ... 
  render: function () {
    return <div onClick={this.toggleOpen}> 
      { this.state.open ? "Open" : "Closed" } 
    </div>; 
  } 
});
State in Action

var Menu = React.createClass( {
    getInitialState: function () {
        return { open: false }; 
    },
    toggleOpen: function() {
        this.setState({ open: !this.state.open });
    },
    ...
    render: function() {
        return <div onClick={ this.toggleOpen }>
            { this.state.open ? "Open" : "Closed" }
        </div>
    }
}) ;
Any time props or state change, the component will call render. **Every time.**

(Default Behavior)
Uh...
That’s predictable
Virtual DOM
(aka the part of React.js you have heard of)
State or props change first

Then the page is update to reflect the changes
Changes Flow
One Direction
Changes Flow Down

<NewsApp>
  <UnreadCount />
  <NewsList>
    <NewsItem>
  </NewsList>
</NewsApp>
Changes Flow Down

```javascript
this.setState({
  items: items,
  unread: itemIds
});
```
Changes Flow Down

<NewsApp>
  <UnreadCount count= {...} />
  <NewsList>
    <NewsItem>
  </NewsItem>
</NewsList>
</NewsApp>
Changes Flow Down

```
<NewsApp>
  <UnreadCount />
  <NewsList items={...}>
    <NewsItem>
  </NewsItem>
</NewsList>
```
Changes Flow Down

<NewsApp>
  <UnreadCount />
  <NewsList>
    <NewsItem {...item} unread={...}>
  </NewsList>
</NewsApp>
Changes Flow Down

When clicked, reduce UnreadCount by 1 and remove my class "is-unread"
Changes Flow Down

<NewsApp>

<UnreadCount />

<NewsList>

<NewsItem>

When clicked, reduce UnreadCount by 1 and remove my class "is-unread"
Changes Flow Down

When clicked, call this.props.newsItemRead(id)
Changes Flow Down

<NewsApp>
<UnreadCount />
<NewsList>
<NewsItem>

When clicked, publish("newsItemRead", id );
Changes Flow Down

```javascript
this.setState({
  unread: itemIds
})
```
A coding pattern that produces predictable components without sacrificing performance

Key Benefit
I’II Say Yes To Another Library

But what does my commitment look like?
Your Commitment

• Give control of as little or as much of your page as you want.

• A transpilation step (if you use JSX)

• Some shims/polyfills if you support IE8
A focused library that can be integrated gradually.
Still Not Convinced?

• Accessibility
• Server side rendering
• Other rendering engines
• Touch support
• React Developer Tools
I’ll Say No To Another Library

So what advice do you have for me?
Have a Single Source of Truth

• Not in the DOM, in JavaScript variables

• Changes are made first to the variables, then the view is updated to reflect the change.
Don’t Make Relative Changes

• Don’t “add 1” or “subtract 1” when adding or removing an item from a cart.

• Adjust the array of cart items, and update the count with the new length of that array.
Don’t Make Relative Changes

• Don’t call `toggleClass`, hoping the correct class gets added or removed.

• Store a `true` or `false` value in a variable. Invert the value when you want to toggle, and then call `toggleClass( "my-class", myVar )`
Have a Single Owner

• Never have two separate pieces of JavaScript modifying the same aspect of a DOM element (style, children, etc)
Thank You

doug@leankit.com
@dougneiner