React Tech Talk

Thomas LaToza
SWE 632
1/23/18

React

- Framework for building complex web user interfaces
- Enables apps to be built **declaratively**, efficiently rendering and updating HTML based on changes in app state
- Breaks up complex apps into encapsulated **components** written in JS rather than HTML that reduce dependencies and encourage reuse
- Interops well with other frontend web technologies
- Can also be used to build native mobile apps

Embedding HTML in Javascript

```
return <div>Hello {this.props.name}</div>;
```

- HTML embedded in JavaScript
  - HTML can be used as an expression
    - HTML is checked for correct syntax
- Can use { expr } to evaluate an expression and return a value
  - e.g., { 5 + 2 }, { foo() }
- Output of expression is HTML

Hello world example

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello world!
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage/>, mountNode
);
```

"Declare a HelloMessage component"

Declares a new component with the provided functions.

"Return the following HTML whenever the component is rendered"

Render generates the HTML for the component. The HTML is dynamically generated by the library.

"Render HelloMessage and insert in mountNode"

Instantiates component, replaces mountNode innerHTML with rendered HTML. Second parameter should always be a DOM element.

Properties

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

“Read this.props.name and output the value”

Evaluates the expression to a value.

“Set the name property of HelloMessage to John”

Components have a this.props collection that contains a set of properties instantiated for each component.

State

- Can update state
- this.setState(OBJ)
- Triggers call to render() to generate new HTML for new state

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

Working with state

- Constructor should initialize state of object

```
constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
```

- Use this.setState to update state

```
    this.setState({
      date: new Date()
    });
```

- Doing this will (asynchronously) eventually result in render being invoked
  - Multiple state updates may be batched together and result in a single render call

Nesting components

```
render() {
    return (
        <div>
            <PagePic pagename={this.props.pagename} />
            <PageLink pagename={this.props.pagename} />
        </div>
    );
}
```

Establishes ownership by creating in render function.

Sets pagename property of child to value of pagename property of parent

- UI is often composed of nested components
- Parent *owns* instance of child
  - Occurs whenever component instantiates other component in render function
  - Parent configures child by passing in properties through attributes

Component lifecycle

[component created]
constructor(...)
render()
componentDidMount()

[component is being destroyed]
componentWillUnmount()

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

Babel

```
<script src="https://cdnjs.com/libraries/babel-core/
5.8.34"> </script>

<script type="text/babel">
//JSX here
</script>
```

- React components usually written in an extension of JavaScript called JSX
- Using JSX requires a **transpiler**
- Takes JSX and outputs traditional Javascript (a.k.a ES5)
- Can use directly in web page or through build process

https://babeljs.io/

Status

- Open source, created and maintained by Facebook
- Initially released in 2013
- Actively maintained and updated
  - Newest release focus on performance
- Used widely by popular websites
  - e.g., Facebook, Airbnb, Uber, Netflix, Twitter, Pinterest, Reddit
- Wide variety of related frameworks that build on top of it

Competitors

- Other frontend JS frameworks
  - Angular, Vue.js, ember.js
- Traditional server side frameworks
  - PHP, JSP, ASP, Ruby on Rails, Django, ...

Summary

- Organizes web apps into encapsulated components
  - Easier to reuse, test, debug, change, ...
- Does the work in figuring out what HTML changes need to be made
  - Only need to be able to construct HTML from app state
- Embeds HTML in code rather than code in HTML
- Use of JSX requires either a build a process for frontend (usually) or added runtime overhead