React Workshop, PTW 2019 DevDays

Introduction

React and Friends: Introduction

What is React?

- A component library written by developers from Facebook
- Includes JSX, a domain-specific component rendering language
- React is the 'view' in client-side 'MVC'

The React "Component"

- A React **`Component`**:
    - a single logical entity rendered by React
    - mounted as a custom HTML tag
    - receives custom attributes as data
    - renders HTML as output

React Components are HTML Tags

Mounting a ficticious React component

```
<body>
   <Game                                          ①
         theme={'colorful'}                       ②
         highScores={[                            ③
             { name: 'AAA', score: 999 },
             { name: 'BBB', score: 888 }
         ]}
   />                                             ④
</body>
```

① React components are mounted as custom HTML Elements

② Custom attributes, known as **`props`**, feed data to the component

③ Data passed as props can be complex Javascript structures

④ Components can be bodyless when they don’t have children

Why Components?

- They are extremely re-usable

- They can be defined with ECMAScript class definitions or even functions
- They can be composed into a tree

Components are Composed into Trees

A tree of components

```
<body>
   <Game ...>                                       ①
     <ScorePanel ... />                             ②
     <GamePlayArea ...>                             ③
        <Controls .... />
     </GamePlayArea>
   </Game>
</body>
```

① Components can contain an element body

② Inner components can be bodyless if they contain their own view area

③ Components can nest other components to any *reasonable* level of depth

React Component Lifecycle

- Components live within React's lifecycle and can respond to events such as
  - Mounting
  - Receiving data
  - Unmounting
  - Deciding whether to update

Rendering and Receiving Data

- React Components are driven by a **`render`** method
- Components can be fed read-only properties from parent components (**`props`**)
- Components re-render when their fed props change

> Components can also be stateful… We'll discuss later

React Component Data Flow

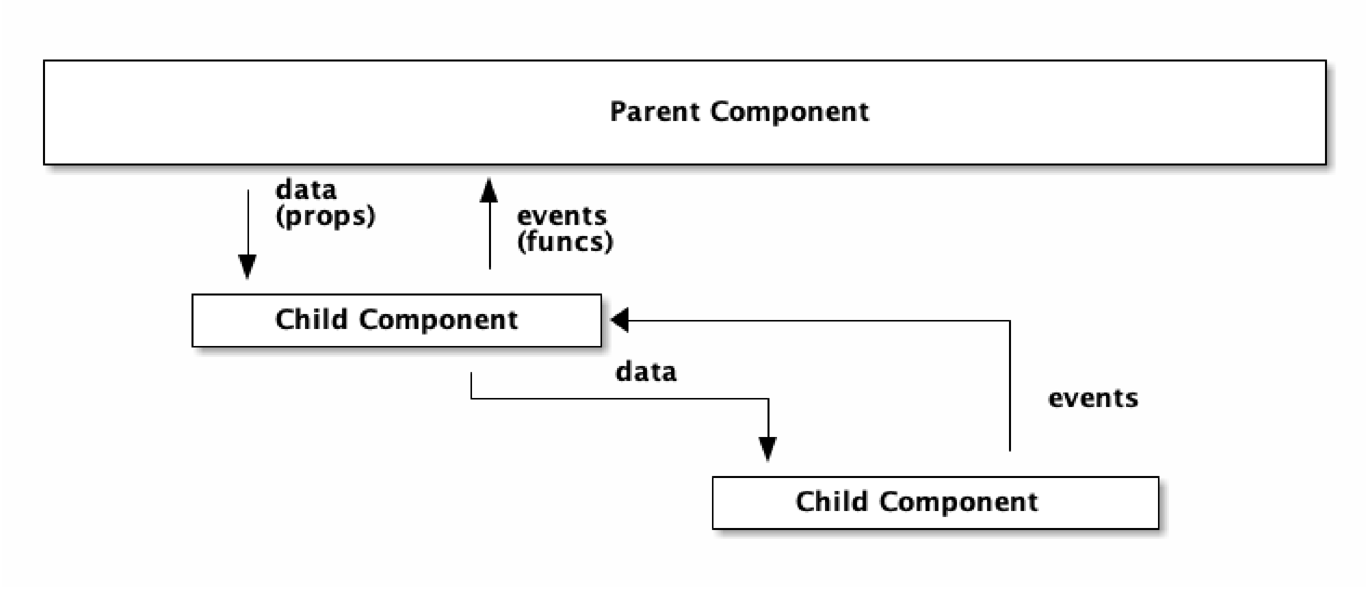

Figure 1. Data flows downward, events flow upward

React’s ecosystem

- Developers commonly use a collection of other APIs, some by Facebook, some by others, that circle around the core API
  - State management (redux)
  - Forms processing (formsy, react-forms, redux-forms, plain forms)
  - View routing (react-router)
  - Data APIs (Relay/GraphQL Servers)
  - Ajax APIs (axios, fetch, superagent, etc.)

Key React Concepts

- Components maintain a "Virtual DOM"
  - a memory-based model of the contents of the DOM
  - changes to the component are compared to the original virtual DOM content
  - differences are applied in efficient ways via batching
- Components have a lifecycle
  - Respond to initialization, data changes, unmounting, and other events
- React makes view development simpler with JSX

Learning how to work within JSX makes you a productive React developer

Learn by Doing

- Create a react app
- Create a simple functional component
- Experiment with JSX
- Pass in props, handle props in inner component
- Wrap content with outer components
- Light CSS Styling with wrapper components, StyledComponents
- BREAK!!

Learn by Doing, Part 2

- Create a class-based component
- Define a stateful component
- Handle events
- Basic managed forms
- BREAK!!!

Learn by Doing, Part 3

- Send state changes back to parent component
- React hooks (if time permits) for functional state management
- Using an AJAX API
- Using a GraphQL API
- END

Final thoughts, what we left out

- Redux / state engines - why you need them?
- Routing
- Testing (check out JEST and Enzyme for useful testing APIs)
- TypeScript for managing datatypes

- RxJS or other reactive library for managing data flow

Wrapup

- React is a component-driven system
- Break your apps down into trees of components
- Atomic design matches nicely