Chapter 1. Introduction to React.js

Background

In the early days of web application development, it was common to send a request to the server, the server would then respond with the full page. This was the only way to create web applications. This approach was very simple from a development perspective. As you didn't need to worry about the events happening in the browser. All you need to do was create the whole page from scratch every time.

Languages like PHP make this kind of development very simple, it is also very simple to create functional components in PHP to help the developer reuse code and reason about what their application was supposed to do. The simplicity of development helped make PHP an extremely popular language to write web applications.

Using this approach it was very hard to create an awesome user experience, as each time the user wanted to do something they had to make another request to the server and wait for the response. In doing so, the user lost all the state they had built up in their page.

In an effort to create better user experiences people started to write libraries to help render applications in the browser with Javascript. These libraries used a variety of means to control the DOM, from simple parameterized HTML templating systems to systems controlling the entire application. As people started to use these libraries in larger and larger applications, it becomes increasingly hard to reason about how your application is supposed to behave as these applications are the result of the long tangled thread of events. In comparison to the old PHP way of doing applications, these modern client side applications are very hard to do well.

React started as a port of a PHP framework by Facebook called XHP. Being a PHP framework, XHP was designed to render our the entire page every time a request is made. React was born to bring the PHP style work flow to client side applications.

React.js has a narrow scope. It is concerned with only two things:

1. Updating the DOM
2. Responding to events

React has no opinions on AJAX, routing, storage, or how to structure your data. It is not a Model-View-Controller framework; if anything, it is the V in MVC. This narrow scope gives you the freedom to incorporate React into a wide variety of systems. In fact it has been used to render views in several popular MVC frameworks.

Rendering the entire page every time some state changes is incredibly slow in Javascript. However, React has a very powerful rendering system.

Like high-performance 3D game engines, React is built around render functions that take the state of the world and translate it into a virtual representation of the resulting page. Whenever React is informed of a state change, it re-runs those functions to determine a new virtual representation of the page, then automatically translates that result into the necessary DOM changes to reflect the new presentation.

At a glance, this sounds like it should be slower than the usual JavaScript approach of updating each element on an as-needed basis. Behind the scenes, however, React does just that: it has a very efficient algorithm for determining the differences between the current virtual page representation and the new one. From those differences it makes the minimal set of updates necessary to the DOM.

What makes this a win for performance is that it minimizes reflows and unnecessary DOM mutations, both of which are common culprits for poor performance.

The bigger your interface gets, the more likely it is to have one interaction that triggers an update, which in turn triggers another update, which in turn triggers another. When these cascading updates are not batched properly, performance starts to degrade substantially. Worse, sometimes DOM elements are updated multiple times before arriving in their final state.

Not only does React's virtual representation diffing minimize these problems by performing the minimal set of updates in a single pass, it also simplifies the maintenance of your application. When the state of the world changes based on user input or external updates, you simply notify React of the state change and it takes care of the rest automatically. There's no need to micromanage.

Book Overview

This book will take you through 4 main topic areas to help you develop an edge with React.js.

Creating and Composing Components

The first 7 chapters of the book are all about creating and composing React components. These chapters will give a good understanding of how to use React.

1) Introduction to React.js
This first chapter introduces React.js, covering the background and the book overview.

2) Using JSX and basic React.js Components
JSX (Javascript XML) is a way of writing declarative XML style syntax in side Javascript. You will learn how to use JSX with React and how to build basic React.js Components. Most of the examples in this book and the example app use JSX.

3) React.js Component lifecycle
React.js is often creating and destroying components during the render process. React.js provides many functions you can hook into during the lifecycle of your components. You should gain an understanding of how to manage the lifecycle of a component to ensure you don't create memory leaks in your applications.

4) Data flow in React.js
It is important to know how data is passed down through the component tree and what data is safe to change. React.js has a very clear separation of `props` and `state`. This chapter will teach you how to use props and state correctly in your React.js components.

5) Event Handling
React.js implements event handling in a declarative manner. Event handling is an important part of any dynamic UI, learning to master them is essential, fortunately React.js makes this very simple.

6) Composing Components
React.js encourages you to make small, precise components that do a specific job. You then need to create orchestration layers in your application to compose these components. This chapter will teach you how to use your components in other components.

7) React.js Mixins
Mixins in React.js provide a way to share common functionality in many React.js components. Using mixins is another way you can break down your components into smaller more manageable parts.

Advanced Topics

Once you have mastered the basics of React you will move on to some more advanced topics. These next 6 chapters will help you really hone your React skills and understanding of how to build great React components.

8) Accessing the DOM from React.js
Even with all the power of the virtual DOM in React.js, sometimes you will still need to access the

or to get more control over your components. This chapter will teach you where in the lifecycle of your React.js components you can safely access the DOM and when to release your control on the DOM to avoid leaking your DOM nodes.

9) Building Forms with React.js
Using HTML form elements is one of the best ways of receiving input from your users. However HTML form elements are very stateful. React.js provides a way to move most of the state from your form elements into your React.js components. This gives you incredible control over your form elements.

10) Animations
As web developers we are already blessed with a very declarative way of defining high performance animations, CSS. React.js encourages the use of CSS for your animations. This chapter will take you the mechanism that React.js provides to help you leverage CSS for animating your React.js components.

11) Performance Tuning your Components
The virtual DOM in React.js gives you great performance right out of the box. There is always room for improvement. React.js provides a way to tell the render that it doesn't need to re-render your component if you know your component has not changed props or state. Doing this can greatly improve the speed of your applications.

12) Server Side Rendering
Many applications need SEO, fortunately React.js can be rendered to string in a non-browser environment like nodejs. Server side rendering can also improve first page load times of your applications also. However writing your application to support both server side rendering and client side rendering can be difficult. This chapter provides you with some strategies for isomorphic rendering and highlights some of the more challenging considerations you will encounter with server side rendering.

13) React.js Addons
React.js provides some of the advanced functionality via it's addons package. Many of these addons are delivered as mixins. Learning how to use these addons will help you build better functionality faster with React.js.

Tooling for React.js

React has some fantastic developer tools and test suits. Learning to use them will help you write robust applications. This section is broken into 3 chapters.

14) React.js Debugging Tools
React.js provide a Google Chrome plugin that enables you to inspect your application visualizing your React.js components. You will learn how to setup this plugin and how to use it.

15) Writing Tests for React.js

as your applications grow. Writing tests also help you write better code as it encourages you to write more modular code. This chapter will guide you through how to test all aspects of your React.js components.

16) Build Tools
As you write larger Javascript applications you will start to require a way to automate packaging your code for deployment. React.js is supported by the major code packaging tools, Browserify and Webpack. This chapter will show you how to configure Browserify and Webpack to support React.js and JSX.

Working with React.js

The final 3 chapters cover important aspects of working with React.

17) Architectural Patterns
React.js provides only "V" in "MVC", it is however very flexible in plugin in with other frameworks and systems. This chapter will help guide you in designing larger scale applications with React.js.

18) Other Javascript libraries in the React.js Family
Facebook open sourced React.js as part of it's over all architectural tools used in house. As time has progressed many other libraries have been released by Facebook that are designed to work seamlessly with React.js. This chapter will give you insight on how these other libraries fit into the React.js family.

19) Use Cases for React.js
While React.js focuses on the web, it can be used anywhere javascript is supported. This chapter looks at some other the other ways you can use React.js outside of the traditional web use case.