React

Dennis Mbae

Ad infinitum

Contents

Introduction

Used on Facebook, Instagram

Uses of React:

- Web apps

- Static sites
- Mobile
- Desktop
- Server-rendered
- Virtual Reality

github.com/chentsulin/awesome-react-renderer

Advantages of React

- Server-side rendering <- ReactDomServer.renderToString() -Nextjs, Gatsby, Phenomic
- Broad Browser Support
- Low risk way of migrating from an older website.
- Codesandbox - test files
- Updates the DOM in the most efficient way
- Testing Frameworks - Mocha, Jasmine, Tape, QUnit, AVA,Jest
- Large community and corporate backing.

Build steps

Minify -> transpile -> test and lint

Avoiding conflicts when using React

- Standardize on a version
- Upgrade react when upgrading libraries
- Upgrade as a team

5 things to consider when starting any project

- Dev environment - which boilerplate do you want to start with - create-react-app
- Whether to declare components as classes or functions
- How to handle types - propTypes,typeScript and Flow
- State - Flux, Redux, MobX
- Styling

Getting started

Components

- Similar to functions
- Input: props, state | output :UI
- Reusable and composable
- Can manage a private state

Function components and class components

Function - input:props(immutable):output -> DOM

Class - input:state(mutable): output -> DOM

Reactive updates

- React will react

-Takes updates to the browser

Virtual views in memory

- Generate HTML for JS
- No HTML template language
- Tree reconciliation. Only changes what needs to be changed and avoids re-rendering everything. More efficient.

function Hello() { return

Hello React!

; }

ReactDOM.render(, - used as dom element document.getElementById(‘mountNode’) - where we would like it to show up,);

- Components MUST start with Capital letters

function Button() { const [counter,setCounter] = useState(0); return <button onClick={() => setCounter(counter+1)}>{counter}; }

hooks

Functions that manipulate state variables

dynamic is encloded in single curly braces. eg

props

Passed down to functions from parent components - one way flow of data

We define state down in a tree as close as possible to the children

React only changes what is necessary. HTML takes more effort to do so

Modern JavaScript Crash Course

Scopes

Block scope - for ,if loop - leak

Function scope - functions - do not leak

Use let instead of var to not leak the scope of the variable

const - references are constant. Reference cannot be changed in memory. Objects like arrays have elements that can be changed but the reference cannot be changed. Strings and single value integers they are immutable.

e.g. for (var i = 1;i<=10;i++) …

you can still access i after the loop

Normal vs Arrow Functions

```
normal function(){

  the value of the this keyword inside a regular function depends on how the
  function was called (the object that called it)

}

Arrow Functions () =>{
  The value of this keyword inside an arrow function depends on where the function was defined.
}
```

Regular functions give access to their calling environment while arrow functions give access to their defining environment

Object literals

This is just a object in json format

dynamic naming

mystery = 'answer'

```
obj = {
 [mystery] = 5
}
```

obj.answer = 5

Destructuring

Extracts objects directly from the class

const PI = Math.PI

const E = Math.E

const {PI,E} = Math

can use it in functions instead of taking the whole object and can also have defaults

e.g. {precision = 2} = {}//last part makes it optional

const [first,...restofItems]

Template Strings

“ - can be dynamic and are literal as they are typed e.g if you skip a line

Promises

Promises and Async/Await- object that will deliver the data at a later point

Closures

Misc

Classes

functions are mutable

Project 1

URL

https://jscomplete.com/playground/rgs2.7

What components to create?

If you have no plan of what to do, start with what makes sense. Rename/ Delete later

JS:

```
class Card extends React.Component {
    render() {
      const profile = this.props;
      console.log(profile)
      return(
        <div className="github-profile" style={{ margin : '1rem  '}}>
          <img src = {profile.avatar_url}/>
          <div className = "info">
```

```
            <div className = "name">{profile.name}</div>
            <div className ="company">{profile.company}</div>
          </div>
        </div>
      );
    }
}

const CardList = (props) => {
  return(
    <div>
      {props.profiles.map(profile => <Card key = {profile.id} {...profile}/>)}
    </div>
  )
}

class Form extends React.Component {

    //userNameInput = React.createRef();
    //use ref in the jsx component and .current.value to access current value

    //we use controlled components to force the state of the UI through react rather
    // than the native DOM so react has access to live changes in the data
    state = { userName: ''};

    handleSubmit = async (event) => {
      event.preventDefault();
      const resp = await axios.get(`https://api.github.com/users/${this.state.userName}`)
      this.props.onSubmit(resp.data);
      this.setState({userName : ""});
    };

    render() {
      return(
        <form onSubmit = {this.handleSubmit}>
            <input
              type = 'text'
              placeholder = "Github.username"
              value = {this.state.userName}
              onChange = {event => this.setState({userName : event.target.value})}
              required
            />
          <button>Add card</button>
        </form>
      );
    }
}

class App extends React.Component {
    /*constructor(props) {
      super(props)
      this.state = {
        profiles: [];
      };//has to be an object
```

```
    }*/

    state = {
        profiles:[],
    };//has to be an object

    addNewProfile = (profileData) => {
      this.setState(prevState => ({
        profiles:[...prevState.profiles,profileData],
      }));
    };

    render() {
      return(
        <div>
          <div className="header">{this.props.title}</div>
          <Form onSubmit = {this.addNewProfile}/>
          <CardList profiles = {this.state.profiles}/>
        </div>
      );
    }
}

ReactDOM.render(
    <App title="The GitHub Cards App" />,
  mountNode,
);

CSS:

/* Nerdy secret: You can use "less" below! Don't tell anybody... */

.github-profile {
    margin: 1rem;
  img {
    width: 75px;
  }
  .info {
    display: inline-block;
    margin-left: 12px;
        .name {
        font-size: 1.25rem;
      font-weight: bold;
    }
  }
}

form {
    border: thin solid #ddd;
  background-color: #fbfbfb;
  padding: 2rem;
  margin-bottom: 2rem;
  display: flex;
  justify-content: center;
```

```
}

.header {
    text-align: center;
  font-size: 1.5rem;
  margin-bottom: 1rem;
}
```

We have dynamic js scripting in react

www.github.com/MicheleBertoli/css-in-js

Project 2

URL

https://jscomplete.com/playground/rgs3.9

Good part of code for a component?

-When you have many items that share data or behaviour e.g stars changing
-Readability

Do not name functions like javascript functions. They will override.

Try as much as possible to minimize the state. If you can extract data from the state, there is no need to add it to the state

Do all computations before the UI parts to ensure the code is more readable

i.e.

1 -> [state,hooks]

2 -> computations and logic

3 -> UI stuff

When you rest a state, you also need to reset the side-effects i.e. things that go along with the state e.g. timers

setInterval - countDown setTimeout

Side Effects

useEffect() - to introduce side effect- runs every time the component renders

```
useEffect (() => {
  const timerId = setTimeout(() => {
    setSecondsLeft(secondsLeft - 1)
  },1000)

  //clean
  return () => clearTimeout(timerId);

});
```

When you create a side effect, clean it to ensure it is only run when needed NOT when other components are needed.

Side effect- even when an object is unmounted, side effects still go on. Therefore we need to clean them up to avoid errors

- Whatever you return is the clean up. e.g. removing event listeners

Another way to change the game: - Mount and Unmount it from the DOM by using a different key therefore unmounting it and remounting a brand new state

Custom Hooks

-Like a state manager function -Good practice start with use e.g. useGameState

Setting up a development environment

- Different from production environment therefore must set up both locally
- Can use high level-tools e.g. create-react-app

Using create-react-app

npm

npx - Instead of mantaining a react package locally, npx fetches the most latest release directly from npm library everytime it is run

npx create-react-app anyname e.g. cra-test

npm run eject - permanet - copy all configurations and scripts used by the Webpack tool locally to your project

config - webpack.config.js

Create your own development environment for Node and React

1.Initializing

mkdir - make a directory. Change to the directory

npm init - for package.json

2.Installing Main/Production Dependencies

A full-stack JavaScript environment has 2 main types of dependencies, production dependencies that need to be installed on production servers and development dependencies that are only needed on local development machines.

3.Installing Development dependencies

These are dependencies that will not be needed in production but help in development.

4.Configuring Dependencies

eslint - same level as package.json

Can install Prettier to work with eslint

Configuring a test environment with jest

5.Creating an Initial Directory Structure

Default webpack looks for.

```
fulljs/
  dist/
    main.js
  src/
    index.js
    components/
      App.js
    server/
      server.js
```

dist(distribution) is where webpack writes files from src when we are ready for production

Note how I created a separate server directory for the backend code. It's always a good idea to separate code that you run in your trusted private backends from code that is to be run on public clients.

6.Configuring Webpack and Babel

7.Creating npm Scripts for Development

You need 2 commands to run this environment. You need to run your web server and you need to run Webpack to bundle the frontend application for browsers. You can use npm scripts to manage these.

```
// In package.json
scripts: {
  "test": "jest"
}
```

Add 2 more scripts in there. The first script is to run the server file with Nodemon and make it work with the same Babel configuration above. You can name the script anything. For example:

"dev-server": "nodemon –exec babel-node src/server/server.js –ignore dist/"

"dev-bundle": "webpack -w -d"

The -w flag in the command above is to run Webpack in watch mode as well and the -d flag is a set of built-in configurations to make Webpack generate a development-friendly bundle.

```
// In package.json
scripts: {
  "test": "jest",
  "dev-server": "nodemon --exec babel-node src/server/server.js --ignore dist/",
  "dev-bundle": "webpack -w -d"
}
```

Testing Everything with a Sample React Application

ReactDOM.hydrate instead of reactDOM.render

npm run dev-server

npm run dev-bundle

React with redux

JS Component Structure

Extraction:

import dependencies

function

export default functionName

Class types

1. React.createClass #deprecated
2. Javascript class extends ReactComponent
3. function
4. arrow functions

Functional components benefits

- Easier to understand
- Avoids this
- Less transpiled code
- Higher signal to noise ratio
- Enhanced code completion/intellisense
- Easy to test
- Perfomance
- Classes may be deprecated in the future

Container vs presentation components

- Stateful vs stateless
- Controller view vs view
- Behaviour vs all markup
- Pass data and actions down vs just receive data and actions via props

- Knows redux vs doesn't know
- Often stateful vs no state

When you notice that some components don't use props they receive but merely forward them down. It's a good time to introduce some container components

Ways to manage state

1. Lift State
2. ReactContext is also a state manager
3. Redux - store

When is redux helpful?

-Complex data flows

-Same data, many places

-Intercommunication

When to get to redux?

Start with state in a single component

Lift state to common parent

Use redux when lifting state gets annoying

Redux principles

-Single immutable store

-Can only change through actions

-Reducers update state

Redux Flow

React ->Action {type:data , ..somedata}->(Store<->Reducer returns new state)->React

let store = createStore(reducer)

Store - createStore in entry point for app

store.disaptch(action)

store.subscribe(listener)

store.getState()

replaceReducer(nextReducer)

Immutability

To change state, return a new object e.g numbers,strings-> an new object is created to replace the last one

Immutability safe Ways to clone existing states

1) Object.assign(target,….sources) e.g.

Object.assign({/dont forget this},state,{role: 'admin'})

2) const newState = {…state,role:'admin'}

1 and 2 create shallow copies - don't copy object references

```
const user = {
   name:'C',
   address:{
    state:"California"
   }
}
```

//it doesn't clone the nested address object

const userCopy = { …user };

//this clones the nested address object too

const userCopy = { …user , address : {…user.address}};

Only do it if you wish to change the nested object

Deep cloning is expensive and wasteful

Causes unneccessary renders

3) use immer or other libraries/immutable.js/seamless-immutable

Arrays in redux

- Avoid push,pop,reverse - they mutate the array
- Prefer map, filter, reduce, concat, spread - return new arrays

Why Redux is preffered to other libraries?

- Clarity
- Perfomance
- Awesome sauce - can see state changes

Enforcing immutability

use redux-immutable-state-invariant(recommended for production use only)

Reducers

- Update state and returns a new state
- Must be pure - no side effects, no mutating arguments, call non-pure functions e.g Math.Random()

Connecting react to redux

Use react-redux library

Container vs Presentational(again)

-Focus on how things work vs focus on how things look -Aware of redux vs unaware of redux -Subscribe to redux state vs read data from props -Dispatch redux actions vs invoke callbacks on props

Provider - attaches app top level component to store Connect - creates container components - Wraps component so it's connected to redux store

connect(mapStateToProps - what states to expose as props, mapDispatchToProps - what actions to expose on props)

Memoize for perfomance with a library like reselect

4 ways to handle mapDispatchToProps

a) Ignore it

this.props.dispatch(loadCourse())

b)Manually wrap it in dispatch

```
function mapDispatchToProps(dispatch){
  return{
    loadCourses: () => dispatch(loadCourses())
  }
}
```

c)Use bindActionCreators

```
function mapDispatchToProps(dispatch) {
  return{
    actions: bindActionCreators(actions,dispatch)
  }
}
```

d) Return object

```
const mapDispatchToProps = {
     incrementCounter
}
```

A chat with redux

React: Hey CourseAction, someone clicked this “Save Course” button

Action: Thanks React! I will dispatch an action so reducers that care can update state

Reducer: Ah, thanks action. Isee you passed me the current state and the action to perform. I’ll make anew copy of the state and return it

Store: Thanks for updating the state reducer. I’ll make sure that all connected components are aware

React-Redux: Woah, thanks for the new data Mr. Store. I’ll now intelligently detemine if I should tell React about this change so that it only has to bother with updating the UI when necessary

React: Ooo! Shiny new data has been passed down via props from the store! I’ll update the UI to reflect this!