

React JS

Notes for Professionals

Chapter 3: Using ReactJS with TypeScript

Section 3.1: ReactJS component written in TypeScript

Actually you can use ReactJS components in TypeScript as in Facebook's example. Just replace to TypeScript:

```
//HelloMessage.tsx
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

But in order to make full use of TypeScript's main feature (static type checking) should be to TypeScript:

1) Convert React.createClass example to ES6 Class:

```
//HelloMessage.tsx
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

2) Next add Props and State interfaces:

```
interface HelloMessageProps {
  name: string;
}
interface HelloMessageState {
  //empty in our case
}
```

```
class HelloMessage extends React.Component<HelloMessageProps, HelloMessageState> {
  constructor(props: HelloMessageProps) {
    super(props);
  }
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

Now TypeScript will display an error if the programmer forgets to pass props defined in the interface.

Section 3.2: Installation and Setup

To use TypeScript with React in a Node project, you must first have a project directory initialized with npm:

Installing via npm or yarn

Chapter 10: React Routing

Section 10.1: Example Routes.js file, followed by use of Router Link in component

Place a file like the following in your top level directory. It defines which components to render for which paths:

```
import React from 'react';
import { Route, IndexRoute } from 'react-router';
import New from './containers/new-guest';
import Show from './containers/show';

import Index from './containers/home';
import App from './components/app';

export default (
  <Route path="/" component={App}>
    <IndexRoute component={Index} />
    <Route path="/posts/new" component={New} />
    <Route path="/posts/:id" component={Show} />
  </Route>
);
```

Now in your top level index.js that is your entry point to the app, you need only render this Router component like so:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, browserHistory } from 'react-router';
// import the routes component as created in routes.js
import routes from './routes';

// entry point
ReactDOM.render(
  <Router history={browserHistory} routes={routes} />,
  document.getElementById('main')
);
```

Now it is simply a matter of using Link instead of <a> tags throughout your application. Using Link will communicate with React Router to change the React Router route to the specified link, which will in turn render the correct component as defined in routes.js

```
import React from 'react';
import { Link } from 'react-router';

export default function PostButton(props) {
  return (
    <Link to="/posts/:id" props={props}>
      <div className="post-button">
        {props.title}
        <span>{props.tags}</span>
      </div>
    </Link>
  );
}
```

React JS Notes for Professionals

Chapter 14: React AJAX call

Section 14.1: HTTP GET request

Sometimes a component needs to render some data from a remote endpoint (e.g. a REST API). A standard practice is to make such calls in components lifecycle method.

Here is an example, using `superagent` as AJAX helper:

```
import React from 'react';
import request from 'superagent';

class App extends React.Component {
  constructor() {
    super();
    this.state = {};
  }

  componentDidMount() {
    request
      .get('/search')
      .query({ query: 'React' })
      .query({ range: '1..5' })
      .query({ order: 'desc' })
      .set('X-Requested-With', 'XMLHttpRequest')
      .end((err, resp) => {
        if (!err) {
          this.setState({ responseData: resp.text });
        }
      });
  }

  render() {
    return (
      <div>
        <div>{this.state.responseData}</div>
        <div>Waiting for response...</div>
      </div>
    );
  }
}
```

React.render(<App />, document.getElementById('root'))

A request can be initiated by invoking the appropriate method on the request object, then calling `.end()` to send the request. Setting header fields is simple, invoke `.set()` with a field name and value.

The `.query()` method accepts objects, which when used with the GET method will form a query-string. The following will produce the path `/search?query=React&range=1..5&order=desc`.

POST requests

```
request.post('/new')
  .set('Content-Type', 'application/json')
  .send('{"name": "js", "tags": "react"}')
  .end(callback);
```

See [Superagent docs](#) for more details.

React JS Notes for Professionals

100+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with React	2
Section 1.1: What is ReactJS?	2
Section 1.2: Installation or Setup	3
Section 1.3: Hello World with Stateless Functions	4
Section 1.4: Absolute Basics of Creating Reusable Components	5
Section 1.5: Create React App	6
Section 1.6: Hello World	7
Section 1.7: Hello World Component	8
Chapter 2: Components	11
Section 2.1: Creating Components	11
Section 2.2: Basic Component	13
Section 2.3: Nesting Components	14
Section 2.4: Props	16
Section 2.5: Component states - Dynamic user-interface	17
Section 2.6: Variations of Stateless Functional Components	19
Section 2.7: setState pitfalls	20
Chapter 3: Using ReactJS with TypeScript	22
Section 3.1: ReactJS component written in TypeScript	22
Section 3.2: Installation and Setup	22
Section 3.3: Stateless React Components in TypeScript	23
Section 3.4: Stateless and property-less Components	24
Chapter 4: State in React	25
Section 4.1: Basic State	25
Section 4.2: Common Antipattern	25
Section 4.3: setState()	26
Section 4.4: State, Events And Managed Controls	28
Chapter 5: Props in React	30
Section 5.1: Introduction	30
Section 5.2: Default props	30
Section 5.3: PropTypes	31
Section 5.4: Passing down props using spread operator	32
Section 5.5: Props.children and component composition	33
Section 5.6: Detecting the type of Children components	34
Chapter 6: React Component Lifecycle	35
Section 6.1: Component Creation	35
Section 6.2: Component Removal	37
Section 6.3: Component Update	38
Section 6.4: Lifecycle method call in different states	39
Section 6.5: React Component Container	40
Chapter 7: Forms and User Input	42
Section 7.1: Controlled Components	42
Section 7.2: Uncontrolled Components	42
Chapter 8: React Boilerplate [React + Babel + Webpack]	44
Section 8.1: react-starter project	44
Section 8.2: Setting up the project	45

Chapter 9: Using ReactJS with jQuery	48
Section 9.1: ReactJS with jQuery	48
Chapter 10: React Routing	50
Section 10.1: Example Routes.js file, followed by use of Router Link in component	50
Section 10.2: React Routing Async	51
Chapter 11: Communicate Between Components	52
Section 11.1: Communication between Stateless Functional Components	52
Chapter 12: How to setup a basic webpack, react and babel environment	54
Section 12.1: How to build a pipeline for a customized "Hello world" with images	54
Chapter 13: React.createClass vs extends React.Component	58
Section 13.1: Create React Component	58
Section 13.2: "this" Context	58
Section 13.3: Declare Default Props and PropTypes	60
Section 13.4: Mixins	62
Section 13.5: Set Initial State	62
Section 13.6: ES6/React "this" keyword with ajax to get data from server	63
Chapter 14: React AJAX call	65
Section 14.1: HTTP GET request	65
Section 14.2: HTTP GET request and looping through data	66
Section 14.3: Ajax in React without a third party library - a.k.a with VanillaJS	66
Chapter 15: Communication Between Components	68
Section 15.1: Child to Parent Components	68
Section 15.2: Not-related Components	68
Section 15.3: Parent to Child Components	69
Chapter 16: Stateless Functional Components	71
Section 16.1: Stateless Functional Component	71
Chapter 17: Performance	74
Section 17.1: Performance measurement with ReactJS	74
Section 17.2: React's diff algorithm	74
Section 17.3: The Basics - HTML DOM vs Virtual DOM	75
Section 17.4: Tips & Tricks	76
Chapter 18: Introduction to Server-Side Rendering	77
Section 18.1: Rendering components	77
Chapter 19: Setting Up React Environment	78
Section 19.1: Simple React Component	78
Section 19.2: Install all dependencies	78
Section 19.3: Configure webpack	78
Section 19.4: Configure babel	79
Section 19.5: HTML file to use react component	79
Section 19.6: Transpile and bundle your component	79
Chapter 20: Using React with Flow	80
Section 20.1: Using Flow to check prop types of stateless functional components	80
Section 20.2: Using Flow to check prop types	80
Chapter 21: JSX	81
Section 21.1: Props in JSX	81
Section 21.2: Children in JSX	82
Chapter 22: React Forms	85
Section 22.1: Controlled Components	85

Chapter 23: User interface solutions	87
Section 23.1: Basic Pane	87
Section 23.2: Panel	87
Section 23.3: Tab	88
Section 23.4: PanelGroup	88
Section 23.5: Example view with `PanelGroup`s	89
Chapter 24: Using ReactJS in Flux way	91
Section 24.1: Data Flow	91
Chapter 25: React, Webpack & TypeScript installation	92
Section 25.1: webpack.config.js	92
Section 25.2: tsconfig.json	92
Section 25.3: My First Component	93
Chapter 26: How and why to use keys in React	94
Section 26.1: Basic Example	94
Chapter 27: Keys in react	95
Section 27.1: Using the id of an element	95
Section 27.2: Using the array index	95
Chapter 28: Higher Order Components	97
Section 28.1: Higher Order Component that checks for authentication	97
Section 28.2: Simple Higher Order Component	98
Chapter 29: React with Redux	99
Section 29.1: Using Connect	99
Appendix A: Installation	100
Section A.1: Simple setup	100
Section A.2: Using webpack-dev-server	101
Appendix B: React Tools	103
Section B.1: Links	103
Credits	104
You may also like	106

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:

<https://goalkicker.com/ReactJSBook>

This *React JS Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official React JS group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

Chapter 1: Getting started with React

Version Release Date

0.3.0	2013-05-29
0.4.0	2013-07-17
0.5.0	2013-10-16
0.8.0	2013-12-19
0.9.0	2014-02-20
0.10.0	2014-03-21
0.11.0	2014-07-17
0.12.0	2014-10-28
0.13.0	2015-03-10
0.14.0	2015-10-07
15.0.0	2016-04-07
15.1.0	2016-05-20
15.2.0	2016-07-01
15.2.1	2016-07-08
15.3.0	2016-07-29
15.3.1	2016-08-19
15.3.2	2016-09-19
15.4.0	2016-11-16
15.4.1	2016-11-23
15.4.2	2017-01-06
15.5.0	2017-04-07
15.6.0	2017-06-13
15.6.1	2017-06-14
15.6.2	2017-09-25
16.0.0	2017-09-26
16.1.0	2017-11-09
16.1.1	2017-11-13
16.3.0	2018-03-29
16.3.1	2018-04-03
16.3.2	2018-04-16

Section 1.1: What is ReactJS?

ReactJS is an open-source, component based front end library responsible only for the **view layer** of the application. It is maintained by Facebook.

ReactJS uses virtual DOM based mechanism to fill in data (views) in HTML DOM. The virtual DOM works fast owing to the fact that it only changes individual DOM elements instead of reloading complete DOM every time

A React application is made up of multiple **components**, each responsible for outputting a small, reusable piece of HTML. Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain internal state - for example, a TabList component may store a variable corresponding to the currently open tab.

React allows us to write components using a domain-specific language called JSX. JSX allows us to write our components using HTML, whilst mixing in JavaScript events. React will internally convert this into a virtual DOM, and will ultimately output our HTML for us.

React "reacts" to state changes in your components quickly and automatically to rerender the components in the HTML DOM by utilizing the virtual DOM. The virtual DOM is an in-memory representation of an actual DOM. By doing most of the processing inside the virtual DOM rather than directly in the browser's DOM, React can act quickly and only add, update, and remove components which have changed since the last render cycle occurred.

Section 1.2: Installation or Setup

ReactJS is a JavaScript library contained in a single file `react-<version>.js` that can be included in any HTML page. People also commonly install the React DOM library `react-dom-<version>.js` along with the main React file:

Basic Inclusion

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <script type="text/javascript" src="/path/to/react.js"></script>
    <script type="text/javascript" src="/path/to/react-dom.js"></script>
    <script type="text/javascript">
      // Use react JavaScript code here or in a separate file
    </script>
  </body>
</html>
```

To get the JavaScript files, go to [the installation page](#) of the official React documentation.

React also supports [JSX syntax](#). JSX is an extension created by Facebook that adds XML syntax to JavaScript. In order to use JSX you need to include the Babel library and change `<script type="text/javascript">` to `<script type="text/babel">` in order to translate JSX to Javascript code.

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <script type="text/javascript" src="/path/to/react.js"></script>
    <script type="text/javascript" src="/path/to/react-dom.js"></script>
    <script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>
    <script type="text/babel">
      // Use react JSX code here or in a separate file
    </script>
  </body>
</html>
```

Installing via npm

You can also install React using [npm](#) by doing the following:

```
npm install --save react react-dom
```

To use React in your JavaScript project, you can do the following:

```
var React = require('react');
var ReactDOM = require('react-dom');
```

```
ReactDOM.render(<App />, ...);
```

Installing via Yarn

Facebook released its own package manager named [Yarn](#), which can also be used to install React. After installing Yarn you just need to run this command:

```
yarn add react react-dom
```

You can then use React in your project in exactly the same way as if you had installed React via npm.

Section 1.3: Hello World with Stateless Functions

Stateless components are getting their philosophy from functional programming. Which implies that: A function returns all time the same thing exactly on what is given to it.

For example:

```
const statelessSum = (a, b) => a + b;

let a = 0;
const statefulSum = () => a++;
```

As you can see from the above example that, `statelessSum` is always will return the same values given `a` and `b`. However, `statefulSum` function will not return the same values given even no parameters. This type of function's behaviour is also called as a *side-effect*. Since, the component affects somethings beyond.

So, it is advised to use stateless components more often, since they are *side-effect free* and will create the same behaviour always. That is what you want to be after in your apps because fluctuating state is the worst case scenario for a maintainable program.

The most basic type of react component is one without state. React components that are pure functions of their props and do not require any internal state management can be written as simple JavaScript functions. These are said to be `Stateless Functional Components` because they are a function only of props, without having any state to keep track of.

Here is a simple example to illustrate the concept of a `Stateless Functional Component`:

```
// In HTML
<div id="element"></div>

// In React
const MyComponent = props => {
  return <h1>Hello, {props.name}</h1>;
};

ReactDOM.render(<MyComponent name="Arun" />, element);
// Will render <h1>Hello, Arun!</h1>
```

Note that all that this component does is render an `h1` element containing the `name` prop. This component doesn't keep track of any state. Here's an ES6 example as well:

```
import React from 'react'

const HelloWorld = props => (
  <h1>Hello, {props.name}</h1>
)
```



```

HelloWorld.propTypes = {
  name: React.PropTypes.string.isRequired
}

export default HelloWorld

```

Since these components do not require a backing instance to manage the state, React has more room for optimizations. The implementation is clean, but as of yet [no such optimizations for stateless components have been implemented](#).

Section 1.4: Absolute Basics of Creating Reusable Components

Components and Props

As React concerns itself only with an application's view, the bulk of development in React will be the creation of components. A component represents a portion of the view of your application. "Props" are simply the attributes used on a JSX node (e.g. `<SomeComponent someProp="some prop's value" />`), and are the primary way our application interacts with our components. In the snippet above, inside of `SomeComponent`, we would have access to `this.props`, whose value would be the object `{someProp: "some prop's value"}`.

It can be useful to think of React components as simple functions - they take input in the form of "props", and produce output as markup. Many simple components take this a step further, making themselves "Pure Functions", meaning they do not issue side effects, and are idempotent (given a set of inputs, the component will always produce the same output). This goal can be formally enforced by actually creating components as functions, rather than "classes". There are three ways of creating a React component:

- **Functional ("Stateless") Components**

```

const FirstComponent = props => (
  <div>{props.content}</div>
);

```

- **React.createClass()**

```

const SecondComponent = React.createClass({
  render: function () {
    return (
      <div>{this.props.content}</div>
    );
  }
});

```

- **ES2015 Classes**

```

class ThirdComponent extends React.Component {
  render() {
    return (
      <div>{this.props.content}</div>
    );
  }
}

```

These components are used in exactly the same way:

```

const ParentComponent = function (props) {
  const someText = "FooBar";

```