**MobX Tutorial #1 - MobX + React is AWESOME:**

MobX is a simple way to put a data layer to the React application

React is a functional reactive UI/view layer to all the data in the state of one’s application

“MobX + React” will be a simple framework

The work it takes to get started with something like Redux, Relay or Flux is sometimes daunting (令人畏惧的) and somehow unnecessary

/\* ./package.json \*/

{

"name": "react-mobx-todos",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "webpack-dev-server --content-base src --inline --hot"

// Both “npm run start” and “npm start” work

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"mobx": "^2.3.7",

"mobx-react": "^3.5.1",

"react": "^15.2.1",

"react-dom": "^15.3.0"

},

"devDependencies": {

"babel-core": "^6.17.0",

"babel-loader": "^6.2.4",

"babel-plugin-transform-class-properties": "^6.10.2",

"babel-plugin-transform-decorators-legacy": "^1.3.4",

"babel-preset-es2015": "^6.9.0",

"babel-preset-react": "^6.11.1",

"css-loader": "^0.23.1",

"react-addons-test-utils": "^15.3.0",

"style-loader": "^0.13.1",

"webpack": "^1.13.1",

"webpack-dev-server": "^1.14.1"

}

}

/\* ./webpack.config.js \*/

var debug = process.env.NODE\_ENV !== "production";

var webpack = require('webpack');

var path = require('path');

module.exports = {

context: path.join(\_\_dirname, "src"),

devtool: debug ? "inline-sourcemap" : null,

entry: "./js/main.js", // Not "./js/client.js"

module: {

loaders: [

{

test: /\.js$/,

exclude: /(node\_modules|bower\_components)/,

loader: 'babel-loader',

query: { presets: ['es2015', 'react'],

plugins: ["transform-decorators-legacy", "transform-class-properties"] }

// To work with MobX, we need the two plugins

// This is actually not necessary

},

{ test: /\.css$/, loader: "style-loader!css-loader" },

]

},

output: {

path: path.join(\_\_dirname, "src"),

filename: "main.min.js"

},

plugins: debug ? [] : [

new webpack.optimize.DedupePlugin(),

new webpack.optimize.OccurenceOrderPlugin(),

new webpack.optimize.UglifyJsPlugin({ mangle: false, sourcemap: false }),

],

};

<!-- ./src/index.html -->

<html>

<head>

<link href="https://fonts.googleapis.com/css?family=Slabo+27px" rel="stylesheet">

</head>

<body>

<div id="app"></div>

<script src="main.min.js"></script>

</body>

</html>

/\* ./src/js/main.js \*/

import "../css/main.css"

import React from "react"

import ReactDOM from "react-dom"

import TodoList from "./TodoList"

import TodoStore from "./TodoStore"

const app = document.getElementById("app")

ReactDOM.render(<TodoList/>, app)

// We render a to-do list

/\* ./src/css/main.css \*/

html,

body {

margin: 0;

padding: 0;

}

body {

font-family: 'Slabo 27px', serif;

background: #f5f5f5;

color: #4d4d4d;

min-width: 230px;

max-width: 550px;

margin: 0 auto;

-webkit-font-smoothing: antialiased;

-moz-font-smoothing: antialiased;

font-smoothing: antialiased;

font-weight: 300;

}

input {

border-radius: 5px;

padding: 5px;

border: 1px solid rgba(0,0,0,0.3);

margin-right: 10px

}

input::-webkit-input-placeholder {

font-style: italic;

font-weight: 300;

color: rgba(0,0,0,0.3);

}

input::-moz-placeholder {

font-style: italic;

font-weight: 300;

color: rgba(0,0,0,0.3);

}

input::input-placeholder {

font-style: italic;

font-weight: 300;

color: rgba(0,0,0,0.3);

}

h1 {

font-weight: 100;

font-size: 100px;

padding:0;

margin: 0;

}

/\* ./src/js/TodoList.js \*/

import React from "react";

export default class TodoList extends React.Component {

render() {

return <h1>MobX</h1>

}

}

/\* ./src/js/TodoStore.js \*/

/\* (empty) \*/

We now start using MobX

/\* ./src/js/TodoStore.js \*/

import { autorun, observable } from "mobx";

// autorun is imported in order to debug

/\* (We usually have the store like the following)

var store = {

todo: [],

createTodo() {

// ...

}

// We cannot really reflect the change in the UI unless we do it manually

}

\*/

// The MobX way to do the same task

class TodoStore {

@observable todos = ["Buy milk", "Buy eggs"];

@observable filter = "";

}

var store = new TodoStore;

window.store = store ; // For debugging

export default store;

/\*

autorun(() => {

// The function will be called whenever the store changes

console.log(store.filter);

// Try to type "store.filter = "milk"" in the console

// Type "store.filter" in the console

// It seems that this only returns a simple string

// The change event is executed by the setter and getter

console.log(store.todos[0]);

// Try to type "store.todos[0] = "Buy cheese"" in the console

// Type "store.todos" in the Chrome DevTools' console

// We can see that an object named "ObservableArray" is created. (It is quite a complex object.)

});

\*/

/\* ./src/js/main.js \*/

import "../css/main.css"

import React from "react"

import ReactDOM from "react-dom"

import TodoList from "./TodoList"

import TodoStore from "./TodoStore"

const app = document.getElementById("app")

ReactDOM.render(<TodoList store={TodoStore} />, app)

// We pass TodoStore as a prop

/\* ./src/js/TodoList.js \*/

import React from "react";

import { observer } from "mobx-react"; // New

@observer // We decorate this class

export default class TodoList extends React.Component {

render() {

return <h1>{this.props.store.todos[0]}</h1>;

// Try typing "store.todos[0] = "Buy fruits"" in Chrome DevTools' console

}

}

**MobX tutorial #2 - Computed Values and Nested/Referenced Observables:**

We are going to take a deeper look into MobX

/\* ./src/js/TodoList.js \*/

import React from "react";

import { observer } from "mobx-react";

@observer

export default class TodoList extends React.Component {

createNew(e) {

if (e.which === 13) {

// Enter key is pressed

this.props.store.createTodo(e.target.value);

e.target.value = "";

}

}

filter(e) {

this.props.store.filter = e.target.value;

}

toggleComplete(todo) {

// We have this function so that the checkbox can change its state

todo.complete = !todo.complete;

// Aside from manually change the state here,

// we can also fire off an action and let something outside the module handle it

}

render() {

const { clearComplete, filter, filteredTodos, todos } = this.props.store;

const todoLis = filteredTodos.map(todo => (

<li key={todo.id}>

<input type="checkbox" value={todo.complete} checked={todo.complete}

onChange={this.toggleComplete.bind(this, todo)}/>{todo.value}</li>

));

// We will now actually print the to-do list

return <div>

<h1>To-dos</h1>

<div>filter: {filter}</div>

<input className="filter" value={filter} onChange={this.filter.bind(this)}/>

<input className="create" onKeyPress={this.createNew.bind(this)}/>

<u1>{todoLis}</u1>

<a href="#" onClick={clearComplete}>Clear Completed Item(s)</a>

</div>

// We do not need to have clearAll.bind(this)

// With arrow function, this is automatically done

}

}

/\* ./src/js/TodoStore.js \*/

import { computed, observable } from "mobx";

class Todo {

@observable value;

@observable id;

@observable complete;

constructor(value) {

this.value = value;

this.id = Date.now();

this.complete = false;

}

}

class TodoStore {

@observable todos = [];

// With MobX, we can filter this list according to the filter in real time

@observable filter = "";

@computed get filteredTodos() {

// The get syntax binds an object property to a function that

// will be called when that property is looked up

var matchesFilter = new RegExp(this.filter, "i");

// We are going to do a case insensitive check

return this.todos.filter(todo => !this.filter || matchesFilter.test(todo.value))

// The filter() method creates a new array with all elements that

// pass the test implemented by the provided function

// If no filter is set or the to-do list item matches the filter,

// the item will show

}

clearComplete = () => {

// this.todos = [];

// We cannot point this.todos to a new array because it will mess up the reference

// This is the main difference between observable arrays and plain arrays

const incompleteTodos = this.todos.filter(todo => !todo.complete);

this.todos.replace(incompleteTodos); // We have replace method instead

}

createTodo(value) {

this.todos.push(new Todo(value));

}

}

var store = new TodoStore;

export default store;