which webpack 检查安装路径？

.eslintrc

A plugin Just for sublime, make sure on errors…

可以直接运行:

node server/server.js

如果修改了server.js的东西，要重启，所以

在package.json的npm scripts中加：“server”: “nodemon server/server.js”

Node monitors anytime a file changes, it monitor all of your code ,and restarts your server

需要全局装 npm install –g nodemon

Then

npm run server就好了

import React from 'react' is ES6 thing

等于 var React = require(‘react’)

import { render,methond1 } from 'react-dom' 是引入某个function

等于 var React = require(‘react’).render

ES6的 class syntax？？

export default App

allow us then to import App

webpack.config.js中的

module.exports = {} 又是es5的

back to server.js 有一句：

app.use(express.static('./dist'));

means whenever you look for a file ,the root is the dist file

module.exports = {

entry: ['./client/client.js'],

output: {

path: './dist',

filename: 'bundle.js',

publicPath: '/'

},

module: {

// where you define certain taks you wanna do

}

}

Bable is a whole library basically complie thing,like es6, jsx into js

可以使用webpack –config webpack.config.js

生成编译（es6，jsx）并打包后的bundle.js

在dev环境，使用sourcemap，查看未编译的源码中的错误

module.exports = {

devtool: 'inline-source-map'

在这之前每次修改，都需要 webpack –config webpack.config.js 一下，再npm run server

虽然用了nodemon，每次改动都会重启，但是实际上只对server.js的改动有实际效果，因为没实时的编译压缩，所以bundle.js还是没变

hot module reloading:

a functionality of webpack, allows us to see changes immediately,without having to use webpack to rebuilt our app everytime

-------------------------------------------part 4 --------------------------------------------

use webpack as a middleware in the sever,webpack is somelike hack in your sever.

And it’ll kinda of mimic a bundle.js. what it does is, it allows us to use a plugin called

Hotmodule reloading. What that means is that webpack can keep track your files and bundle,

So anything changes in your files, it’ll make changes immediately in the bundle

Nodemon checks everyfile,but actually don’t need listen to the components file changes

So:

"server":"nodemon server/server.js --ignore components"

安装babel-preset-react-hmre并配置到webpack.config.js中的bable中的query位置

Add some useful webpack plugins

plugings: [ //some webpack plugings

new webpack.optimize.OccurenceOrderPlugin(), //给import或require的文件order啥的，用于优化啥的

new webpack.HotModuleReplacementPlugin(),//let any changes be repalced

new webpack.NoErrorsPlugin() //wont allow the webpak to finish the built if errors, util fixed

],

Then

use webpack as a middleware in the sever.js

这样就不用cmd line中输入 webpack –config webpack.config.js了，为了达到这个目的，在server.js中, we’ll kindof inserting this command in server:

var config = require('../webpack.config.js');

var webpack = require('webpack');

var webpackDevMiddleware = require('webpack-dev-middleware');

var webpackHotMiddleware = require('webpack-hot-middleweare');

var compiler = webpack(config); //actually, here is make a webpack object and takes in the configuration,then the complie can handle withe the middlewares

app.use(webpackDevMiddleware(compiler,{noInfo: true, publicPath: config.output.publicPath}));

//this is what the dev kindaof simulate as your path, in our webpack.config.js，has a publickPath too. PublicPath is what webpack to simulate your publicPath to get that bundle.js(as mentioned earlier, the point of having webpack as a middleware is that it can do hard module reloading and kinda of simulates having your bundle.js but not acturally create it in your dist folder, instead it simulates it ,so the publicPath is what path is gonna track to. So when index.html calls for bunle.js, it’ll server up its special bundle.js. the hard module reloading keepable that bunle.js )

app.use(webpackHotMiddleware(compiler));

now,if we ‘npm run server ‘

we’ll starting the server, which will have the webpack middleware, the the webpack middleware knows about our config, because we created this compiler and give the config. And knows to use ‘’inline-source-map’(as it’s in the configuration). Knows where to enter.

And we have webpack.HotModuleReplacementPlugin() and 'react-hmre' (allows this? To talk to the HotModule)

One more thing todo :

entry: [

'webpack-hot-middleware/client'

'./client/client.js'

],

Error: invalid argument at pathToArray：

output: {

path: require("path").resolve('./dist'),

Error：

[HMR] Hot Module Replacement is disabled

Plugin拼错了…

总结：

在dev环境中，可以使用middleware等

在production环境，不需要以上这些，需要dist里的bundle.js

如何配置这两种环境，要以后说。。。。。

-------------------------------------part 5-------props and state----------------

React 工具

React dev tool

啥意思

constructor (props, context) {

super(props, context)

this.state = {

inputText: 'initial text'

}

}

onChange={this.handleChange.bind(this)}

-------------------------------------part 6----Flux-------------------

Some people say they sould use flux or rudex in the beginning

子组件出发父的函数，改变父的state，中间需要绑定许多this，非常麻烦：

<TextDisplay text={this.state.inputText} deleteLetter={this.deleteLetter.bind(this)} />

<button onClick={this.handleClick.bind(this)}>delete one letter</button>

即 without any special system for handling the state and props, when you do want to change

In a parent component or another, you can pass down functions. This is almost like callback style

Of doing it.

This is many reasons you don’t wanna do this. 会影响bi-direction dataflow？

Better way to do this, it to handle all these changes in a separate location (above the component)

Fackbook make one 🡪flux

-------------------------------------part 8----Redux -------------------

需要redux react-redux

-------------------------------------part 9----Redux -------------------

如果是一下写法

<input />

<button onClick={this.handleSubmit.bind(this)}

Input和button提交分离

可以改造为：

<form onSubmit = this.handleSubmit.bind(this)

<input />

<input type='submit' text='Submit'>

</form>

这样enter可以提交表单

注意语法的使用，以及纯函数的概念

return Object.assign({},state,{

todos: state.todos.map((todo) => {

return todo.id === action.id ?

Object.assign({}, todo, {completed:!todo.completed}) : todo

})

})

return Object.assign({},state,{

todos: state.todos.filter((todo) => {

return todo.id !== action.id

})

})

配置provider和connet之后，可以从上层传递诸如此类下去：

<TodoInput dispatch={this.props.dispatch} />

<TodoList dispatch={this.props.dispatch} todos={this.props.todos} />

One trick:

但是每次都要pass down这个dispatch！

并且每次用到action, his.props.dispatch(actions.completeTodo(this.props.todo.id));都要import action

解决方法是在app.js中

参照：function mapStateToProps(state){

return state

}

function mapDispatchToProps(dispatch) {

return {

actions: bindActionCreators(actions,dispatch)

}

}



这样就没有dispatch传过来了，而是包装过的actions

注意此时dispatch不能再用了

<TodoInput addTodo={this.props.actions.addTodo} />

Then

this.props.addTodo(this.state.inputText)

-------------------------------------part 10----Redux -------------------

Multible reuduces

Now, we are handling one state, that’s really long and complicated. We can use a library called

Combine-reduces:

1. Basically allow to separate reducer functionality into several ones
2. Combine them into one,which get’s passed into your store configuration
3. Anytime an ancion is triggered or dispatched, the state tree is splited up and pass to the

Specific reducer, which then return the part of the state tree, and then ‘combine-reduces’

Takes the results of the reduction? push them back together into one state tree

const rootReducer = combineReducers({

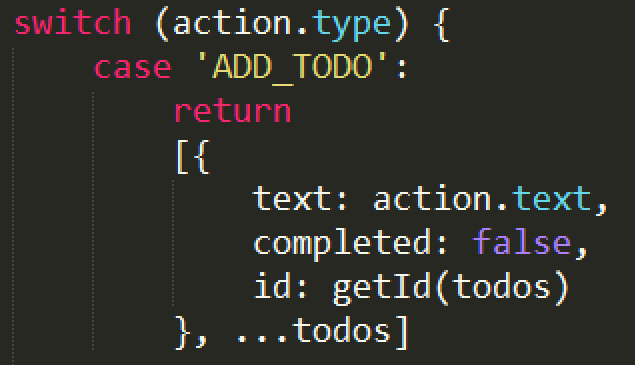
todos: todoReducer,

user: userReducer

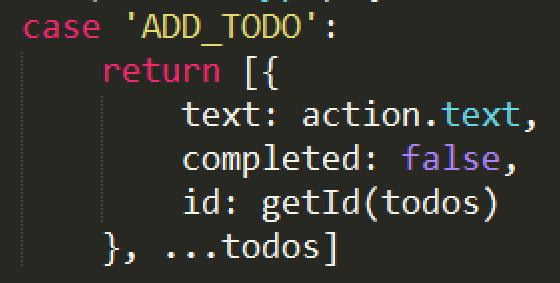
})

注意需要初始化

let UserReducer = function(user={}, action) {



尼玛，上面这样不行，还得下面这样：



How to style your app?