# Webpack-dev-server结合后端服务器的热替换配置

Webpack有一个很实用的功能叫做热替换（Hot-replace），尤其是结合React Hot Loader插件，开发过程中都不需要刷新浏览器，任何前端代码的更改都会实时的在浏览器中表现出来。

首先需要安装Webpack-dev-server,一个轻量的node.js express服务器。

npm install webpack-dev-server --save-dev

Webpack-dev-server十分小巧，这里的作用是用来伺服资源文件，不能替代后端的服务器，因此如果你还要进行后端开发，就要采用双服务器模式：一个后端服务器和一个资源服务器（即Webpack-dev-server)

配置双服务器热替换模式首先需要修改Wepack.config.js的entry条目

entry: [  
    'webpack-dev-server/client?http://0.0.0.0:9090',*//资源服务器地址*  
    'webpack/hot/only-dev-server',  
    './static/js/entry.coffee'  
]

注意，由于要结合了后端服务器，因此不采用  
*entry: {  
    app: ['webpack/hot/dev-server', './app/main.js']  
}*  
这种形式。

接下来修改output条目,将publicPath设置为webpack-dev-server服务器下资源目录的绝对路径

output: {  
    publicPath: "http://127.0.0.1:9090/static/dist/",  
    path: './static/dist/',  
    filename: "bundle.js"  
}

在Plugins中增加

new webpack.DefinePlugin({  
    'process.env.NODE\_ENV': '"development"'  
}),  
new webpack.HotModuleReplacementPlugin()

webpack变量需要在Webpack.config.js的开始处声明

var webpack = require('webpack');

接着中页面文件中插入打包后的资源文件，注意这里要采用资源服务器的绝对路径

<script src="http://127.0.0.1:9090/static/dist/bundle.js"></script>

双服务器的原理其实就是让后端服务器伺服的页面加载资源服务伺服的资源。

**如果要实现react的热加载，还需要[React Hot Loader插件](http://gaearon.github.io/react-hot-loader/getstarted/" \t "_blank) ，**

npm install react-hot-loader --save-dev

然后将react-hot loader放在jsx文件loader链的第一位

{ test: /\.jsx?$/, loaders: ['react-hot', 'jsx?harmony'], include: path.join(\_\_dirname, 'src') }

最后就是启动Webpack-dev-server服务器，注意这里要采用inline模式， 如果采用hot模式会出现错误。--content-base 为资源文件夹。

//////////////////////////////////////////////////////////////////////////////////////////////////////

该项目就是采用这样的配置方式。

webpack.config.js

**output**: {  
 **publicPath**: **"./public/"**,  
 **path**: path.join(\_\_dirname,**'public/'**),  
 **filename**: **'bundle.js'**},  
  
**entry**: [**"webpack-dev-server/client?http://localhost:3000"**,  
 **"webpack/hot/only-dev-server"**,

**'./src/index.js'**],

**loaders**: [  
 {  
 **test**: /\.js$/,  
 **exclude**: /node\_modules/,  
 **loaders**: [**'react-hot'**,**'babel?presets[]=react,presets[]=es2015'**]  
 },

……………………………………..

**plugins**: [  
 **new** *webpack*.HotModuleReplacementPlugin()  
]

///////////////////////////////////////////////////////////////////////////////////////////////

两套后端

**var** *webpack* = require(**'webpack'**);  
**var** *WebpackDevServer* = require(**'webpack-dev-server'**);  
**var config** = require(**'./webpack.config'**);  
  
**new** *WebpackDevServer*(*webpack*(**config**), {  
 **publicPath**: **config**.**output**.**publicPath**,  
 **hot**: **true**,  
 **historyApiFallback**: **true**,  
 **header**: {  
 **'Access-Control-Allow-Origin'**: **'\*'** }  
}).listen(3000,**'localhost'**,**function**(err,result) {  
 **if**(err) {  
 **return console**.log(err)  
 }  
 **console**.log(**'Listen at http://localhost:3000'**)  
});

/////////////////////////////////////////////////////////

**var express** = require(**'express'**);  
**var** path = require(**'path'**);  
**var** app = **express**();  
**var** port = process.**env**.PROT || 1337  
  
app.use(**express**.static(path.join(\_\_dirname,**''**)))  
app.use(**function**(req,res,next) {  
 res.set({  
 **'Access-Control-Allow-Origin'**: **'\*'** })  
 next()  
})  
app.get(**'\*'**,**function**(req,res) {  
 res.sendFile(path.join(\_\_dirname,**'index.html'**))  
})  
app.listen(port,**function**() {  
 **console**.log(**'is running on 1337'**)  
})

/////////////////////////////////////////////////////////////////

数据源：

{  
 **"fileName"**: **"1.jpg"**,  
 **"title"**: **"Heaven of time"**,  
 **"desc"**: **"企鹅。。。。。。。。。"**},

/////////////////////////////////////////////////////

**var** imageDatas = require(**'./data/imageDatas.json'**);  
  
*// 利用自执行函数， 将图片名信息转成图片URL路径信息*imageDatas = (**function** genImageURL(imageDatasArr) {  
 **for**(**var** i=0;i<imageDatasArr.**length**;i++) {  
 **var** singleImageData = imageDatasArr[i];  
 singleImageData.**imageURL** = require(**'./images/'** + singleImageData.**fileName**);  
 imageDatasArr[i] = singleImageData;  
 }  
 **return** imageDatasArr;  
})(imageDatas);

这样每组数据便多了一个**imageURL** props

////////////////////////////////////////////////////////////////

render() {  
 **var** {imageArrInfo} = **this**.**state**;  
 **console**.log(**this**.**state**);  
 imageDatas.forEach(**function**(value,index) {  
 **if**(!**this**.**state**.**imageArrInfo**[index]) {  
 **this**.**state**.**imageArrInfo**[index] = {  
 **pos**: {  
 **left**: (index-2) \* 234 + 10  
 },  
 **active**: index == 4  
 }  
 }  
 }.bind(**this**))  
  
 **for**(**var** i=0;i<imageArrInfo.**length**;i++) {  
 **if**(imageArrInfo[i].**active**) {  
 **break**;  
 }  
 }

这里的i非常重要，它会告诉下面的scene具体要展示哪一张图片（active one）

render 初始化，给每一个数组元素 设定了active以及 pos.

通过active 增加样式，通过pos移动整个图片队列，通过URL显示每一张图片

/////////////////////////////////////////////////////////////////

<**div className="stage" ref="stage"**>  
 <**Scene item=**{imageDatas[i]}/>  
 <**Controller items=**{imageDatas}

**imageArr=**{**this**.**state**.**imageArrInfo**} **handlePreControl=**{**this**.handlePreControl} **handleNextControl=**{**this**.handleNextControl} **handleCenter=**{**this**.handleCenter}/>  
</**div**>

///////////////////////////////////////////////////////////////////////////

**this**.**state** = {  
 **imageArrInfo**:[]  
};  
**this**.handleCenter = **this**.handleCenter.bind(**this**);  
**this**.handlePreControl = **this**.handlePreControl.bind(**this**);  
**this**.handleNextControl = **this**.handleNextControl.bind(**this**);

函数的绑定

向左移动的Click触发函数

handlePreControl() {  
 **var** imageNewInfo = **this**.**state**.**imageArrInfo**;  
 **var** length = imageNewInfo.**length**;  
 **var** index;  
 **for**(**var** i=0;i<imageNewInfo.**length**;i++) {  
 **if**(imageNewInfo[i].**active**) {  
 index = i;  
 **if**(i>0) {  
 imageNewInfo[i].**active** = **false**;  
 imageNewInfo[i-1].**active** = **true**;

//前一张图片增加active 当前张取消active  
 **break**;  
 }  
 }  
 }  
  
 **if**(index!=length-1&&index!=length-2&&index!=0&&index!=1){  
 **for**(**var** i=0;i<imageNewInfo.**length**;i++) {  
 imageNewInfo[i].**pos**.**left** += 234;  
 }  
 }

//在未到头的情况下，整个队列向右移动  
  
 **this**.setState({  
 **imageArrInfo**:imageNewInfo  
 })  
}

//把更改完的图片数组信息传回去

Scene 元件是展示屏， 它只要知道要展示那一张图片即可

<**Scene item=**{imageDatas[i]}/>

///////////////////////////////////////////////////////////////////////////

Scene 元件

第一步

constructor(props){  
 **super**(props)  
}

之后展示即可

<**img src=**{item.**imageURL**} **title=**{item.**title**}/>

<**p**>{item.**desc**}</**p**>

接下来的controller 组件

<**Controller items=**{imageDatas} **imageArr=**{**this**.**state**.**imageArrInfo**} **handlePreControl=**{**this**.handlePreControl} **handleNextControl=**{**this**.handleNextControl} **handleCenter=**{**this**.handleCenter}/>

第一步仍然是

constructor(props){  
 **super**(props)  
}

<**a href="javascript:;" className="pre-controller" onClick=**{**this**.**props**.handlePreControl}>  
 <**i className="icon-circle-left"**></**i**>  
</**a**>

<**a href="javascript:;" className="next-controller" onClick=**{**this**.**props**.**handleNextControl**}>  
 <**i className="icon-circle-right"**></**i**>  
</**a**>

两个触发事件的组件

套路仍然相同

子组件内

**onClick=**{**this**.**props**.**handleNextControl**}

根组件内

**handleNextControl=**{**this**.handleNextControl}

然后具体定义这个函数

////////////////////////////////////////////////////////////////////////

图片的显示通过遍历图片数据得到

<**Controller items=**{imageDatas} **imageArr=**{**this**.**state**.**imageArrInfo**}

//////////////////////////////////////////////////////////////////

**var** {items,imageArr} = **this**.**props**;  
**var** images = items.map(**function**(item,index) {  
 **return** (  
 <**ImageFigure key=**{index} **src=**{item.**imageURL**} **info=**{imageArr[index]} **alt="图片似乎出了点问题"** />  
 )  
})

<**div className="scroll-bar"**>  
 {images}  
</**div**>

**ImageFigure**

该元件利用 URL pos.left 以及 active 来确定这个图片到底该怎么展示，位置在哪，有没有active的特殊效果。

<**ImageFigure key=**{index} **src=**{item.**imageURL**} **info=**{imageArrInfo[index]} **alt="图片似乎出了点问题"** />

render() {

**var** {src,info} = **this**.**props**;  
**var** style = {};  
  
**if**(info.**pos**) {  
 style = info.**pos**}  
**var** imgClass = info.**active** ? **"active"** : **""**;  
  
**return** (  
 <**img src=**{src} **style=**{style} **className=**{imgClass}/>  
)

//pos: { left: 233 }

}