从test-react中学到的东西总结：

package.json

**"scripts"**: {  
 **"test"**: **"echo \"Error: no test specified\" && exit 1"**,  
 **"dev"**: **"node server.js"**,  
 **"dist"**: **"webpack --process --colors -p"**},

**"dependencies"**: {  
 **"flex.css"**: **"1.0.1"**,  
 **"react"**: **"^15.0.1"**,  
 **"react-dom"**: **"^15.0.1"**},  
**"devDependencies"**: {  
 **"autoprefixer-loader"**: **"^3.2.0"**,  
 **"babel-core"**: **"^6.18.2"**,  
 **"babel-loader"**: **"^6.2.4"**,  
 **"babel-preset-es2015"**: **"^6.6.0"**,  
 **"babel-preset-react"**: **"^6.5.0"**,  
 **"body-parser"**: **"^1.15.2"**,  
 **"css-loader"**: **"^0.23.1"**,  
 **"extract-text-webpack-plugin"**: **"^1.0.1"**,  
 **"file-loader"**: **"^0.8.5"**,  
 **"html-webpack-plugin"**: **"^2.22.0"**,  
 **"jsx-loader"**: **"^0.13.2"**,  
 **"less"**: **"^2.6.1"**,  
 **"less-loader"**: **"^2.2.3"**,  
 **"request"**: **"^2.72.0"**,  
 **"style-loader"**: **"^0.13.1"**,  
 **"url-loader"**: **"^0.5.7"**,  
 **"webpack"**: **"^1.13.0"**,  
 **"webpack-dev-server"**: **"^1.14.1"**}

webpack.config.js

**var** *webpack* = require(**'webpack'**);  
**var** *ExtractTextPlugin* = require(**'extract-text-webpack-plugin'**);  
*//css单独打包***var** *HtmlWebpackPlugin* = require(**'html-webpack-plugin'**);  
*//生成html*

plugins.push(**new** *ExtractTextPlugin*(**'[name].css'**));*//css单独打包*

**new** *HtmlWebpackPlugin*({  
 *//根据模板插入css/js等生成最终HTML* **filename**: **'./index.html'**,  
 **template**: **'./src/template/index.html'**,  
 **hash**:**true**, *//为静态资源生成hash值*}));

App.jsx

{  
 **this**.**state**.**list**.map((item) => {  
 **return** <**List item=**{item} **key=**{item.**id**} **editItem=**{**this**.editItem} **updataItem=**{**this**.updataItem} **delItem=**{**this**.delItem}/>  
 })  
}

**this**.searchName = (e) => {  
 **var** name = e.**target**.**value**;  
 **var** {list} = **this**.**state**;  
 *//exec 检索字符串中指定的值。返回找到的值，并确定其位置。  
 //new RegExp(pattern, attributes);  
 //参数 pattern 是一个字符串，指定了正则表达式的模式或其他正则表达式。  
 // new Boolean(value); //构造函数  
 // Boolean(value); //转换函数* list.map((item)=>{  
 **for**(**let** key **in** item) {  
 **if**(Boolean(**new** RegExp(name).exec(item[key]+**''**))) {  
 **return** item.**display** = **''**;*//匹配到了就显示* }  
 }  
 item.**display** = **'none'**;  
 });  
 **this**.setState({list});  
};

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

<**tr style=**{{display}}>

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

**this**.editItem = (data) => {  
 **var** {list} = **this**.**state**;  
 **for**(**let** i=0;i<list.**length**;i++) {  
 **if**(list[i].**id**==data.**id**) {  
 **let** { editState = **true** } = list[i];  
 list[i].**editState** = !editState;  
 **return this**.setState({list});

//这里仍然存在疑问  
 }  
 }  
};

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

**this**.updataItem = (data) => {  
 **var** {list} = **this**.**state**;  
 **for**(**let** i=0;i<list.**length**;i++) {  
 **if**(list[i].**id** == data.**id**) {  
 list[i] = data;  
 **break**;  
 }  
 }  
 **this**.setState({list});  
 fetch(**`/api/edit/**${data.**id**}**`**, {  
 **method**: **'post'**,  
 **headers**: {  
 **'Content-Type'**: **'application/x-www-form-urlencoded'***//application/x-www-form-urlencoded是常用的表单发包方式，普通的表单提交，或者js发包，默认都是通过这种方式，* },  
 **body**:*formData*(data)  
 }).then((res)=> {  
 **return** res.**json**();  
 })  
 .then((res) => {  
 alert(res.**msg**);*//更新成功* **if**(res.**data**) {  
 data.**id** = res.**data**.**id**;  
 **this**.setState({list});  
 }  
 }).catch((e)=>{  
 alert(**'更新失败'**);  
 })  
 };

**function** *formData*(data) {  
 **var** arr = [];  
 **for**(**let** key **in** data) {  
 arr.push(**`**${key} **=** ${data[key]}**`**);  
 }  
 **return** arr.join(**'&'**);  
}

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

对应的后端代码

app.post(**'/api/edit/:id'**,**function**(req,res) {  
 fs.readFile(dataurl,**'utf-8'**,**function**(err,text) {  
 **if**(err){  
 **return** res.**json**({  
 **msg**: **'修改失败'**,  
 **data**:**null**,  
 **status**:**false** })  
 }  
  
 **var** list = **JSON**.parse(text);  
 **var**{id,image,name,age,phone,phrase} = req.**body**;  
 **var** data = { id,image,name,age,phone,phrase };  
  
 **for**(**let** i=0;i<list.**length**;i++) {  
 **if**(list[i].**id**==id) {  
 list[i] = data;  
 fs.*writeFile*(dataurl,**JSON**.stringify(list));  
 **return** res.**json**({  
 **msg**:**'修改成功'**,  
 **data**: data,  
 **status**: **true** })  
 }  
 }  
  
 *//id<0 说明是新增* **if**(req.**params**.**id**<0) {  
 data.**id** = **new** Date().getDate();  
 list.push(data);  
 fs.*writeFile*(dataurl,**JSON**.stringify(list));  
 **return** res.**json**({  
 **msg**:**'保存成功'**,  
 **data**:data,  
 **status**:**true** })  
 }**else** {  
 **return** res.**json**({  
 **msg**:**'操作失败，数据不存在'**,  
 **data**:**null**,  
 **status**:**false** })  
 }  
 })  
});

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

**this**.delItem = (data) => {  
 **var** {list} = **this**.**state**;  
 **for**(**let** i=0; i< list.**length**;i++) {  
 **if**(list[i].**id** == data.**id**) {  
*//arrayObject.splice(index,howmany,item1,.....,itemX)  
//index 必需。整数，规定添加/删除项目的位置，使用负数可从数组结尾处规定位置。  
//howmany 必需。要删除的项目数量。如果设置为 0，则不会删除项目。* list.splice(i,1);  
 **break**;  
 }  
 }  
 **this**.setState({list});

///在这里前端的删除已经完成，下面是利用fetch 匹配一个后端的代码 进行//后端json数据的删除  
 **if**(data.**id**>0) {  
 */\* fetch() 必须接受一个参数——资源的路径。  
 无论请求成功与否，它都返回一个 promise 对象，  
 resolve 对应请求的 Response。\*/* fetch(**`/api/del/**${data.**id**}**`**)  
 .then((res)=>{  
 **return** res.**json**();  
 })  
 .then((res)=>{  
 alert(res.**msg**);*//删除成功* })  
 .catch((e)=>{  
 alert(**'删除失败'**);  
 })  
 }  
 }

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

后端的删除代码

app.get(**'/api/del/:id'**,**function**(req,res) {  
 fs.readFile(dataurl,**'utf-8'**,**function**(err,text) {  
 **if**(err){  
 **return** res.**json**({  
 **msg**: **'获取内容失败'**,  
 **data**: **null**,  
 **status**: **false** });  
 }  
 **var** list = **JSON**.parse(text);  
 *//JSON.parse() 方法将一个 JSON字符串解析成一个javascript值。  
 //JSON.stringify() 方法可以将任意的 JavaScript 值序列化成 JSON 字符串。* **var** {id} = req.**params**;  
 **for**(**let** i=0;i<list.**length**;i++) {  
 **if**(list[i].**id** == id) {  
 list.splice(i,1);  
 fs.*writeFile*(dataurl,**JSON**.stringify(list));  
 *//保存删除后的文件* **return** res.**json**({  
 **msg**:**'删除成功'**,  
 **data**:**null**,  
 **status**: **true** })  
 }  
 }  
 **return** res.**json**({  
 **msg**:**'删除失败，数据不存在'**,  
 **data**:**null**,  
 **status**:**false** })  
 })  
});

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

加载数据

前端

componentDidMount() {  
 fetch(**'/api/list'**)  
 .then((res)=>{  
 **return** res.**json**();  
 })  
 .then((list)=> {  
 **this**.setState({ list })  
 })  
 .catch((e)=>{  
 alert(**'加载失败，请稍后重试！'**);  
 })  
}

后端

app.get(**'/api/list'**,**function**(req,res) {  
 res.sendFile(dataurl);  
});

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

子元件如何传递function

<**button onClick=**{()=>**this**.**props**.**delItem**(**this**.**props**.**item**)}>删除</**button**>

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

**class** List **extends** Component {  
 render() {  
 **var** {id,image,name,age,phone,phrase,display,editState = **true**} = **this**.**props**.**item**;  
 **var** btnname = editState ? **'修改'** : **'保存'**;  
 **return** (  
 <**tr style=**{{display}}>  
 <**td**>{ id>0 ? id : **''**}</**td**>  
 <**td**>  
 <**img className="headimg" src=**{image} **alt=""**/>  
 </**td**>  
 <**td**>  
 <**input type="text" disabled=**{editState} **ref="name" defaultValue=**{name}/>  
 </**td**>  
 <**td**>  
 <**input type="number" disabled=**{editState} **ref="age" defaultValue=**{age}/>  
 </**td**>  
 <**td**>  
 <**input type="tel" disabled=**{editState} **ref="phone" defaultValue=**{phone}/>  
 </**td**>  
 <**td**>  
 <**textarea ref="phrase" defaultValue=**{phrase} **disabled=**{editState} />  
 </**td**>  
 <**td**>  
 <**button onClick=**{**this**.editItem}>{btnname}</**button**>  
 <**button onClick=**{()=>**this**.**props**.**delItem**(**this**.**props**.**item**)}>删除</**button**>  
 </**td**>  
 </**tr**>  
 )  
 }  
 constructor(props) {  
 **super**(props);  
  
 **this**.editItem = () => {  
 **var** {id,image,editState = **true**,editItem} = **this**.**props**.**item**;  
 */\*这里是把item当中的属性传给一个object editState 因为是布尔值所以先写出来\*/* **if**(editState) {  
 **this**.**props**.**editItem**(**this**.**props**.**item**);  
 }**else**{  
 **let** name = **this**.**refs**.**name**.**value**;  
 **let** age = parseInt(**this**.**refs**.**age**.value);  
 **let** phone = **this**.**refs**.**phone**.**value**;  
 **let** phrase = **this**.**refs**.**phrase**.**value**;  
  
 **if**(!name){  
 **return** alert(**'姓名不能为空'**)  
 }**else if**(!age || age <0) {  
 **return** alert(**'年龄不合法'**)  
 }**else if**(!/^\d+$/.test(phone)) {  
 **return** alert(**'手机号码不合法'**);  
 }  
  
 **this**.**props**.**updataItem**({id,image,name,age,phone,phrase,**editState**:**true**})  
 }  
 }  
 }  
}  
  
List.**propTypes** = {  
 **item**: PropTypes.**object**,  
 **editItem**: PropTypes.**func**,  
 **updataItem**: PropTypes.**func**,  
 **delItem**: PropTypes.**func**};  
List.**defaultProps** = {  
 **item**: {}  
};

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

改变关键的editState

**this**.editItem = (data) => {  
 **var** {list} = **this**.**state**;  
 **for**(**let** i=0;i<list.**length**;i++) {  
 **if**(list[i].**id**==data.**id**) {  
 **let** { editState = **true** } = list[i];*//获取该item中editState的值* list[i].**editState** = !editState;*/\*取反\*/* **return this**.setState({list});  
 }  
 }  
};

按键修改变保存 保存变修改

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

如何写后端服务器

**var** *webpack* = require(**'webpack'**);  
**var** *WebpackDevServer* = require(**'webpack-dev-server'**);  
**var config** = require(**'./webpack.config'**);  
**var bodyParser** = require(**'body-parser'**);  
**var** fs = require(**'fs'**);

**var** server = **new** *WebpackDevServer*(*webpack*(**config**), {  
 **publicPath**: **config**.**output**.**publicPath**,  
 **stats**: {  
 **color**: require(**'supports-color'**)  
 }  
});  
  
**var** dataurl = \_\_dirname + **'/data.json'**;  
**var** {app} = server;  
app.use(**bodyParser**.**json**());  
app.use(**bodyParser**.**urlencoded**({  
 **extended**: **false**}));  
*/\*express项目中通常使用body-parser进行post参数的解析，  
最常用的是其中的json和urlencoded的parser，  
可分别对以JSON格式的post参数和urlencoeded的post参数进行解析，  
均可获得一个JSON化的req.body，\*/*

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

排序

age(<**a href="javescript:;" onClick=**{**this**.sortMax}>大</**a**> | <**a href="javescript:;" onClick=**{**this**.sortMin}>小</**a**>)

**this**.sortMax = ()=>{  
 **var** {list} = **this**.**state**;  
 list.sort((o1,o2) => o1.**age** < o2.**age** ? 1 : -1);  
 *// 1 从大到小排列* **this**.setState({list});  
};  
  
**this**.sortMin = ()=>{  
 **var** {list} = **this**.**state**;  
 list.sort((o1,o2) => o1.**age** > o2.**age** ? 1 : -1);  
 **this**.setState({list});  
};  
*/\*  
 \*  
 \* var numbers = [4, 2, 5, 1, 3];  
 numbers.sort(function(a, b) {  
 return a - b;  
 });  
 a<b minus a b 从小到大 与原顺序相同 同向排序  
 a>b + b a 与原顺序相反 反向排序  
 console.log(numbers);  
  
 // [1, 2, 3, 4, 5]  
 \*/*