最近湖北省发生疫情，在家呆着也是无聊，于是就想正现在这个时间写一个服务端渲染的dome。

首先服务端渲染的好处我在这里就不说了，大家可以自行百度。在这里我主要说一下我在做服务端渲染时遇到的一些问题。

我这里使用的架构是：react16.x+react-router5+kao2+babel7+webpack4来完成的

首先我用的是CRA脚手架创建的react项目，并不是自己自定义的webpack文件，

因为CRA中的webpack默认是不对外展示的，需要我们手动执行npm run eject命令把webpack的配置进行导出。

这里之所以要导出，是因为我们需要修改webpack的默认配置，甚至要去修改其中的某些细节部分，这些东西，仅仅只靠customize-cra这类的库是做不到的，

这里也给大家说一下，如果大家想自己手动写服务端渲染的话，是必须要一个打包配置文件的，无论你是手写还是自动生成，都必须要有这个文件。

那么现在就开始我们的ssr之旅吧

首先我们需要一个react的前端项目，在这我已经用CRA生成了基本的项目了，这不过我对这个项目进行了小小的改动。但是并不会影响我们的ssr。

./src/App.jsx



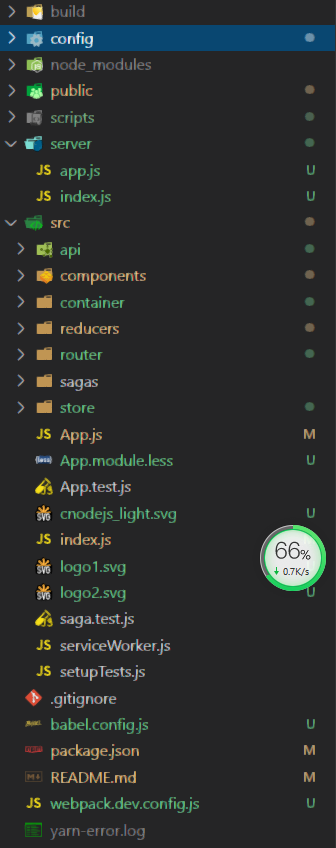
那么上图是我的前端文件内容。

我们只需要关注主要库就可以了，那个hot是用来进行react组件热加载的，我们在下面做的时候，也可以不使用。

那么如果我们要用服务端渲染的话，需要一个服务端文件。我们现在在项目根目录中建立一个server目录，里面存放我们的服务端文件。

这是我们的项目目录，可以看到我们在项目目录中新建了一个server文件用于存放服务器文件，这个文件夹中有两个文件，index.js，app.js。

其中index.js是后端入口文件，app.js是服务端主要执行文件。



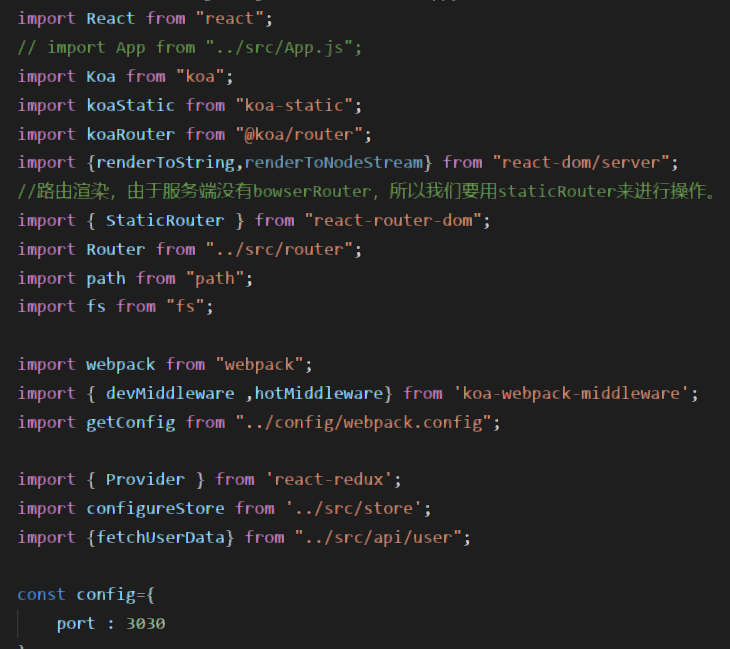
前端文件创建好之后，我们现在要利用node服务器来把客户端的页面吐出去，也就是将服务端生成的html进行合并然后利用react提供的渲染组件的函数进行组件生成，

然后我们把生成号的组件输出到客户端。

在客户端中，react渲染组件的方法是react.render但这个方法在服务端是无法使用的，为此，react提供了专属的服务端渲染方法renderToString和renderToNodeStream

在这里我们主要利用renderToString来进行讲解。

我们在server/app.js中引入我们需要的文件



由于我们这里用的后端框架是koa，所以我们引入koa，并引入一些react在服务端需要使用的库，react-dom/server，由于我们要在服务端也能正确解析调用react.createElement方法，

所以我们也要引入react。最后我们需要将客户端的./src/app.js文件引入进来。因为我们要在服务端将这组件给传输到客户端上去。

捕获.PNG

最后我们需要在server/index.js文件中把server/app.js文件引入进去

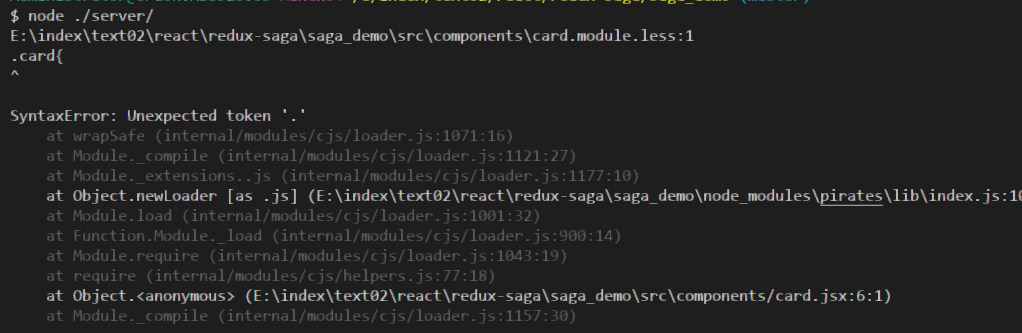


这里之所以引入@babel/register文件是因为我们在服务端会使用上es6或es7的语法，所以我们要引入这个库来对这些语法进行解析。

然后我们现在运行node服务器

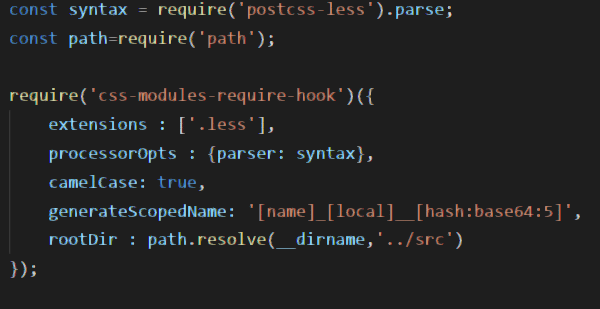
执行node ./server/

这时我们会发现程序报错了



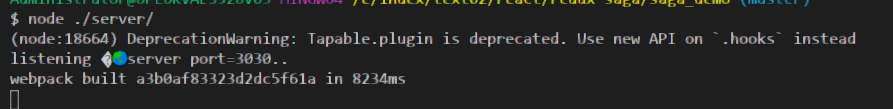
这里报错的原因是因为我们的程序无法识别less或是css语法。

这时我们需要在程序中引入css-modules-require-hook 这个库来解决语法无法识别问题。

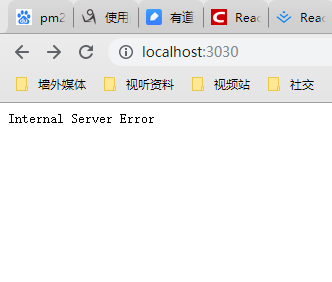


我这里使用的是less所以我用postcss-less来对less文件进行解析，如果你们使用的是其他css预编译器，可以去css-module-require-hook npm网站上查看文档。

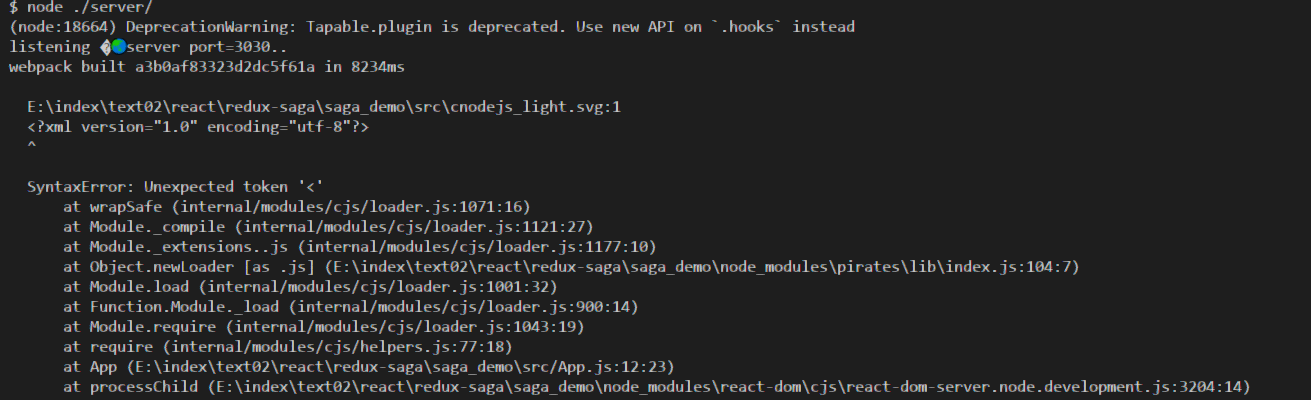
然后我们再次启动服务器，这时我们可以看到服务器已正常启动。



但是，这个时候我们访问localhost:3030的时候，发现页面给我们显示了一个错误信息



这一看就是服务器发生错误了，我们回到服务器的控制台看看到底什么地方出问题了。

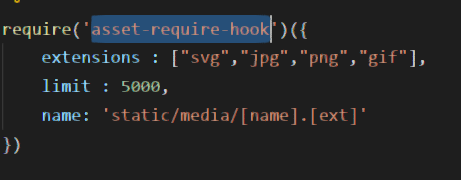


可以看到，这里报的错误信息是解析xml语法解析不了，这是什么原因呢，我们在项目中并没有使用xml文件，这里怎么会报xml错误呢。

其实我们仔细观察一下，发现在报错信息中有一个我们都很熟悉的文件后缀名，svg 学过svg的同学应该知道svg文件就是由xml语法构建的。

那这时我们就知道了，原来是服务器无法识别svg文件。这个时候我们就要引入

asset-require-hook 这个库来解析我们项目中使用的静态文件



在这里我们仅仅只处理图片文件，小于5kb的文件我们使用base64编码进行输出。

一定要注意这里的name字段，这个字段代表你在浏览器上的html引入图片时的路径，这个值一定要跟webpack中的图片输出路径一致不然会无法显示图片。

现在我们在运行服务器



可以发现服务器已经可以正常启动。

到这里我们服务端的初步渲染就已经完成了。

但是一个完整的服务端渲染远远不止这些，我们还需要完成redux的服务端渲染来做到前后端状态一致，这也就是所谓的前后端同构。

除了redux，我们还需要使用react-router进行服务端渲染，来保证前后端的路由处理也是一致的。这两个功能，大家可以看看我的源码。在里面有详细的解决方法。

项目的github