在index.html中

<!--把rem计算放到body头部，目的是为了保证css导入后才执行这段js-->  
<!--webpack打包后会直接把包插入到head-->  
<!--不写在index中而是卸载了页面中，就是为了保证这段程序加载速度快一些，没必要等到合并后的js加载，先把样式处理了-->  
<script>  
 ;(**function** anonymous() {  
 **function** computed() {  
 **let** HTML=document.documentElement;  
 console.log(HTML);  
 **let** winW=HTML.clientWidth;  
 desW=750;  
  
 **if**(winW>=750){  
 HTML.style.fontSize='100px';  
 **return**;  
 }  
 HTML.style.fontSize=winW/750\*100+'px';//html中单独编写的代码尽量不要出现{}等操作，例如：ES6模板字符串，否则webpack编译失败  
  
 }  
 computed();  
 window.addEventListener('resize',computed,**false**);  
 })();

2.transition（不好用，但是要知道）

**import** Transition **from** 'react-transition-group/Transition';  
  
**const** duration = 300;  
  
**const** defaultStyle = {  
 transition: `opacity ${duration}ms`,  
 opacity: 0,  
},transitionStyles = {  
 entering: { opacity: 0 },  
 entered: { opacity: 1 },  
};

使用

<Transition in={**this**.state.in} timeout={0}>  
 {  
 state=>{  
 **return** <ul className={'filterBox'} style={{...defaultStyle,...transitionStyles[state]}}>  
 <li>全部课程</li>  
 <li>react</li>  
 <li>vue课程</li>  
 <li>小程序课程</li>  
  
 </ul>  
 }  
 }  
</Transition>

3.路由的时候小技巧

<Switch>  
 <Route path={'/'} exact component={Home}/>  
 <Route path={'/course'} component={Home}/>  
 <Route path={'/mycourse'} component={Mycourse}/>  
 <Route path={'/person'} component={Person}/>  
 <Redirect to={'/course'}/>//小技巧：每个页面的渲染只有唯一的地址匹配，其他不行的都奔到首页  
</Switch>

4.二级路由

<Switch>  
 <Route path={'/person/info'} render={()=>{  
 **return** '个人信息';  
 }}/>  
 <Route path={'/person/login'} component={Login}/>  
 <Route path={'/person/register'} component={Register}/>  
 <Redirect from={'/person'} to={'/person/info'}/>  
</Switch>

5.API文件夹

真实react/vue项目中我们会把所有向服务器发送具体请求的操作写在api中，（不写在组件中，方便请求处理的统一管理）

Index.js :是把axios或者fetch等进行初始处理或者二次封装的地方

**import** axios **from** 'axios';  
**import** Qs **from** 'qs';  
  
axios.defaults.baseURL = 'http://localhost:8000';  
axios.defaults.withCredentials = **true**;  
axios.defaults.transformRequest = (data = {}) => Qs.stringify(data);  
axios.interceptors.response.use(result => result.data);  
**export default** axios;

withCredentials=true：允许跨域（并且允许携带cookie）

transformRequest： 把post/put通过请求主体传递给服务器的内容统一处理为x-www-url-encoded格式 xx=xxx&xx=xxxx

response：响应拦截器：把服务器返回的信息中响应主体的内容拦截返回，以后在then中获取的结果就是主体内容

6、路由的验证和渲染是同步的，不允许在校验中出现异步，因为在异步没有完成之前，根部不知道渲染谁，语法不支持这样的操作（**下面操作是不对的**）

{/\*<Route path={'/person/info'} render={async ()=>{\*/}  
 {/\*let result=await checkLogin();\*/}  
 {/\*if(parseFloat(result.code)===0){\*/}  
 {/\*return <Info/>;\*/}  
 {/\*}\*/}  
 {/\*return <Tip/>;\*/}  
{/\*}}/>\*/}

**正确方案：**

1. 在contructor中添加状态 this.isLogin=false;
2. 在componentWillMount中通过async await api中方法的结果，改变状态

//=>验证是否登录  
 **async** componentWillMount(){  
 **let** result=**await** checkLogin(),  
 isLogin=parseFloat(result.code)===0?**true**:**true**;  
 **this**.setState({  
 isLogin  
 })  
 }

1. 在render中 如下

<Route path={'/person/info'} render={()=>{  
 **if**(**this**.state.isLogin){  
 **return** <Info/>;  
 }  
 **return** <Tip/>  
}}/>

7、基于render返回的组件不是受路由管控的组件

解决办法：

**export default** withRouter(connect()(Tip));

8、redux临时存储

在获取个人信息时，可以在componentDidMount，获取api中queryInfo信息，判断如果当前是否登录状态，如果是登录状态，则给constructor中state的baseinfo赋值，render渲染-》可以实现，但是会存在一个问题:当前组件重新渲染，就会发送一个请求，（路由切换都会发送一个请求）

9、引入的包

**import** md5 **from** 'blueimp-md5';

10、我们之前说过，当路由切换的时候，对应的组件会重新的渲染，但是渲染也要分情况，

1. 之前渲染其他的组件的时候把当前组件彻底从页面中移除了，再次渲染当前组件，走的是第一次挂载的流程（也就是一切从头开始）

2. 如果当前组件之前没有彻底在页面中移除（本组件内容的子组件在切换），每一次走的是更新的流程，不是重新挂载的流程

11、

//在属性更新和路由切换的时候不能走componentWillUpdate,设置setState的时候会造成死循环，会走componentWillReceiveProps这个  
**async** componentWillReceiveProps(){  
 **let** result=**await** checkLogin();  
 console.log(result);  
 **let** isLogin=parseFloat(result.code)===0?**true**:**false**;  
 **this**.setState({  
 isLogin  
 })  
 }

7月2号

加入购物车流程：

1. 加入购物车
2. 从购物车移除
3. 从服务器获取最新的购物车信息（已支付、未支付）
4. 因为每一次页面刷新，redux存储的购物车信息都会消失，所以我们需要在页面刷新的时候，执行一个dipatch派发，把服务器中存储的购物信息存放到redux中（Nav是每一次页面刷新，不管在哪一个路由下都会执行的组件）

整个流程：

点击加入购物车，服务器请求，把服务器购物车的信息放到redux存起来，两部分，一部分未支付、一部分已支付，有一个问题，刷新，redux没有信息了，页面刷新的时候，页面比价在，在nav，还有问题，登录之后才能支付，上一次3个课程已支付，存在redux了，过了好多天，没有登录重新打开页面，在注册或者成功之后，我们需要重新把服务器信息（登录成功后我们需要重新获取已购买的课程信息，因为未登录下，获取的支付课程信息是获取不到的，但是登录后我们需要把购买信息同步到redux中，这样在我的课程中才能展示出来相关的信息

）存到redux中，

1. 实现全选和非全选

【全选】

现在的项目都是数据驱动（点击复选框也会把数据中的选中信息更新，从而更新组件，让其有选中或者非选中的效果）=》当前案例操作的是redux中的unpay里面的数据

1. 先把redux->unpay中的每一个商品信息额外增加一个属性，记录是否被选中（默认都是被选中的），还要有一个属性记录全选或者全不选的操作
2. 编写一个dispatch任务，用来控制哪些课程需要被选中或者不选中（包含全选和全不选）

【删除】

首先在所有数据中找出选中的那些项，分别执行api：removeShopCart（courseID）这个方法，等到删除操作完成（所有的删除操作都完成了），我们重新从服务器获取最新的购物车信息，更新redux中的状态（dispatch：queryUnpay）,redux信息更新，页面重新渲染

【支付】

类似于删除（支付之前需要先登录）

小组项目

后台：

1. 先定数据结构（JSON）分析数据