前端开发

开发环境设置

使用react-mobx-starter-master脚手架，解压更名为frontend。
在src中新增component、service、css目录。

注：没有特别说明，js开发都在src目录下

```
frontend/
    |-.babelrc
    |-.gitignore
    |-.npmrc
    |-index.html
    |-jsconfig.json
    |-LICENSE
    |-package-lock.json
    |-package.json
    |-README.md
    |-webpack.config.dev.js
    |-webpack.config.prod.js
    └─src/
        |- componet/
        |- service/
        |- css/
        |- index.html
        |- index.js
```


修改项目信息

```
{
  "name": "blog",
  "description": "blog project",
  "author": "wayne"
}
```

webpack.config.dev.js

```
proxy: {
    '/api': {
        target: 'http://127.0.0.1:8080', /*代理到本地8000的web server处理*/
        changeOrigin: true
    }
}
```

安装依赖

```
$ npm install
$ npm install react-router
$ npm install react-router-dom
```

缺省npm会安装到dependencies中

开发

前端路由

前端路由使用react-router组件完成

官网文档 https://reacttraining.com/react-router/web/guides/philosophy

基本例子 https://reacttraining.com/react-router/web/example/basic

使用react-router，更改src/index.js

```
import React from 'react';
import ReactDom from 'react-dom';
import {Route, Link, BrowserRouter as Router} from 'react-router-dom';


const Home = () => (
  <div>
    <h2>Home</h2>
  </div>
);

const About = () => (
  <div>
    <h2>About</h2>
  </div>
);

const App = () => (
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route path="/about" component={About} />
    </div>
  </Router>
);

ReactDom.render(<App />, document.getElementById('root'));
```

在地址栏里面输入`http://127.0.0.1:3000/` 或 `http://127.0.0.1:3000/about`试试看，能够看到页面的变化

App中，使用了Router路由组件，Router是根，且只能有一个元素，所以加了div。

Route负责静态路由
path是匹配路径，没有path总是匹配。
component是目标组件，
exact：布尔值，true时要求路径完全匹配。
strict：布尔值，true时，要求严格匹配，但是url字符串可以是自己的子串。

地址变化，Router组件会匹配路径，然后使用匹配的组件渲染。

登录组件

在component目录下构建react组件。

登录页模板
https://codepen.io/colorlib/pen/rxddKy?q=login&limit=all&type=type-pens

```
<div class="login-page">
  <div class="form">
    <form class="register-form">
      <input type="text" placeholder="name"/>
      <input type="password" placeholder="password"/>
      <input type="text" placeholder="email address"/>
      <button>create</button>
      <p class="message">Already registered? <a href="#">Sign In</a></p>
    </form>
    <form class="login-form">
      <input type="text" placeholder="username"/>
      <input type="password" placeholder="password"/>
      <button>login</button>
      <p class="message">Not registered? <a href="#">Create an account</a></p>
    </form>
  </div>
</div>
```

使用这个HTML模板来构建组件。

特别注意
搬到React组件中的时候，要将class属性改为className。
所有标签，需要闭合。

login.js

在component目录下建立login.js的登录组件。
使用上面的模板的HTML中的登录部分，挪到render函数中。

- 修改class为className
- 将`<a>`标签替换成`<Link to="?">`组件
- 注意标签闭合问题

```
import React from 'react';
import {Link} from 'react-router-dom';

export default class Login extends React.Component {
```

```
    render() {
        return (<div className="login-page">
            <div className="form">
                <form className="login-form">
                    <input type="text" placeholder="邮箱" />
                    <input type="password" placeholder="密码" />
                    <button>登录</button>
                    <p className="message">还未注册? <Link to="/reg">请注册</Link></p>
                </form>
            </div>
        </div>);
    }
}
```

index.js
在路由中增加登录组件

```
import Login from './component/login';

const App = () => (
  <Router>
    <div>
      <Route path="/about" component={About} />
      <Route path="/login" component={Login} />
      <Route exact path="/" component={Home} />
    </div>
  </Router>
);
```

访问 `http://127.0.0.1:3000/login` 就可以看到登录界面了。但是没有样式。

样式表

在src/css中，创建login.css，放入一下内容，然后在login.js中导入样式

```
body {
  background: #456;
  font-family: SimSun;
  font-size: 14px;
}

.login-page {
  width: 360px;
  padding: 8% 0 0;
  margin: auto;
}
.form {
  font-family: "Microsoft YaHei", SimSun;
  position: relative;
  z-index: 1;
  background: #FFFFFF;
  max-width: 360px;
```

```
  margin: 0 auto 100px;
  padding: 45px;
  text-align: center;
  box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.2), 0 5px 5px 0 rgba(0, 0, 0, 0.24);
}
.form input {
  outline: 0;
  background: #f2f2f2;
  width: 100%;
  border: 0;
  margin: 0 0 15px;
  padding: 15px;
  box-sizing: border-box;
  font-size: 14px;
}
.form button {
  text-transform: uppercase;
  outline: 0;
  background: #4CAF50;
  width: 100%;
  border: 0;
  padding: 15px;
  color: #FFFFFF;
  font-size: 14px;
  cursor: pointer;
}
.form button:hover,.form button:active,.form button:focus {
  background: #43A047;
}
.form .message {
  margin: 15px 0 0;
  color: #b3b3b3;
  font-size: 12px;
}
.form .message a {
  color: #4CAF50;
  text-decoration: none;
}
```

在login.js中导入样本表`import '../css/login.css'`报错了，原因没有配置css loader。
webpack.config.dev.js文件中增加css的部分

```
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/,
                use: [
                    { loader: 'react-hot-loader/webpack' },
                    { loader: 'babel-loader' }
                ]
            },
```

```
            {
                test: /\.css$/,
                exclude: /node_modules/,
                use: [ 'style-loader', 'css-loader' ]
            },
            {
                test: /\.less$/,
                use: [
                    { loader: "style-loader" },
                    { loader: "css-loader" },
                    { loader: "less-loader" }
                ]
            }
        ]
    }
```

可以看到界面，如下(http://127.0.0.1:3000/login)

注册组件

与登录组件编写方式差不多，创建component/reg.js，使用login.css

```
import React from 'react';
import { Link } from 'react-router-dom';
import '../css/login.css'

export default class Reg extends React.Component {
```

```
    render() {
        return (
            <div className="login-page">
                <div className="form">
                    <form className="register-form">
                        <input type="text" placeholder="姓名" />
                        <input type="text" placeholder="邮箱" />
                        <input type="password" placeholder="密码" />
                        <input type="password" placeholder="确认密码" />
                        <button>注册</button>
                        <p className="message">如果已经注册 <Link to="/login">请登录</Link></p>
                    </form>
                </div>
            </div>
        );
    }
}
```

在index.js中增加一条静态路由

```
import Reg from './component/reg';

const App = () => (
  <Router>
    <div>
      <Route path="/about" component={About} />
      <Route path="/login" component={Login} />
      <Route path="/reg" component={Reg} />
      <Route exact path="/" component={Home} />
    </div>
  </Router>
);
```

可以看到注册界面，如下(http://127.0.0.1:3000/reg)

导航栏组件

在index.js中增加导航栏组件，方便页面切换

```
const App = () => (
  <Router>
    <div>
      <div>
        <ul>
          <li><Link to="/">主页</Link></li>
          <li><Link to="/login">登录</Link></li>
          <li><Link to="/reg">注册</Link></li>
          <li><Link to="/about">关于</Link></li>
        </ul>
      </div>
      <Route path="/about" component={About} />
```

```
      <Route path="/login" component={Login} />
      <Route path="/reg" component={Reg} />
      <Route exact path="/" component={Home} />
    </div>
  </Router>
);
```

分层
视图层，负责显示数据，每一个React组件一个xxx.js。
服务层，负责业务数据处理逻辑，命名为xxxService.js
Model层，负责数据，数据从后端取

登录功能实现

view层，登录组件和用户交互。当button点击触发onClick，调用事件响应函数handleClick，handleClick中调用服务service层的login函数。

service层，负责业务逻辑处理。调用Model层数据操作函数

在src/service/user.js

```
export default class UserService {
    login (email, password) {
        // TODO
    }
}
```

src/component/login.js

```
import React from 'react';
import {Link} from 'react-router-dom';
import '../css/login.css'

export default class Login extends React.Component {
    handleClick(event) {
        console.log(event.target)
    }

    render() {
        return (<div className="login-page">
            <div className="form">
                <form className="login-form">
                    <input type="text" placeholder="邮箱" />
                    <input type="password" placeholder="密码" />
                    <button onClick={this.handleClick.bind(this)}>登录</button>
                    <p className="message">还未注册? <Link to="/reg">请注册</Link></p>
                </form>
            </div>
        </div>);
```

```
    }
}
```

这一次发现有一些问题，按钮点击会提交，导致页面刷新了。
要阻止页面刷新，其实就是阻止提交。使用event.preventDefault()。

如何拿到邮箱和密码？
event.target.form返回按钮所在表单，可以看做一个数组。
fm[0].value和fm[0].value就是文本框的值。

如何在Login组件中使用UserService实例呢？
可以在Login的构造器中通过属性注入；
也可以在外部使用props注入。使用这种方式。

```
import React from 'react';
import {Link} from 'react-router-dom';
import '../css/login.css'
import UserService from '../service/user';

const userService = new UserService();

export default class Login extends React.Component {
    render () {
        return <_Login service={userService} />;
    }
}

class _Login extends React.Component {
    handleClick(event) {
        event.preventDefault();
        let fm = event.target.form;
        this.props.service.login(
            fm[0].value, fm[1].value
        );
    }

    render() {
        return (<div className="login-page">
            <div className="form">
                <form className="login-form">
                    <input type="text" placeholder="邮箱" />
                    <input type="password" placeholder="密码" />
                    <button onClick={this.handleClick.bind(this)}>登录</button>
                    <p className="message">还未注册? <Link to="/reg">请注册</Link></p>
                </form>
            </div>
        </div>);
    }
}
```

UserService的login方法实现

代理配置

修改webpack.config.dev.js文件中proxy部分
注意，修改这个配置，需要重启dev server

```
    devServer: {
        compress: true,
        port: 3000,
        publicPath: '/assets/',
        hot: true,
        inline: true,
        historyApiFallback: true,
        stats: {
            chunks: false
        },
        proxy: {
            '/api': {
                target: 'http://127.0.0.1:8000',
                changeOrigin: true
            }
        }
    }
```

axios异步库

axios是一个基于Promise的HTTP异步库，可以用在浏览器或nodejs中。

使用axios发起异步调用，完成POST、GET方法的数据提交。可参照官网的例子

中文说明 https://www.kancloud.cn/yunye/axios/234845

安装npm
`$ npm install axios`

导入
`import axios from 'axios';`

service/user.js修改如下

```
import axios from 'axios';

export default class UserService {
    login (email, password) {
        console.log(email, password);

        axios.post('/api/user/login', {
            email:email,
            password:password
        })/* dev server会代理 */
        .then(
            function (response) {
                console.log(response);
                console.log(response.data);
```

```
                console.log(response.status);
                console.log(response.statusText);
                console.log(response.headers);
                console.log(response.config);
            }
        ).catch(
            function (error) {
                console.log(error);
            }
        )
    }
}
```

填入邮箱、密码，点击登录按钮，返回404，查看Python服务端，访问地址是`/api/user/login`，也就是多了/api。如何解决？
1、修改blog server的代码的路由匹配规则
不建议这么做，影响有点大
2、rewrite
类似httpd、nginx等的rewrite功能。本次测试使用的是dev server，去官方看看。https://webpack.js.org/configuration/dev-server/#devserver-proxy
可以查到pathRewrite可以完成路由重写。

```
    devServer: {
        compress: true,
        port: 3000,
        publicPath: '/assets/',
        hot: true,
        inline: true,
        historyApiFallback: true,
        stats: {
            chunks: false
        },
        proxy: {
            '/api': {
                target: 'http://127.0.0.1:8000',
                pathRewrite: {"^/api" : ""},
                changeOrigin: true
            }
        }
    }
```

重启dev server。

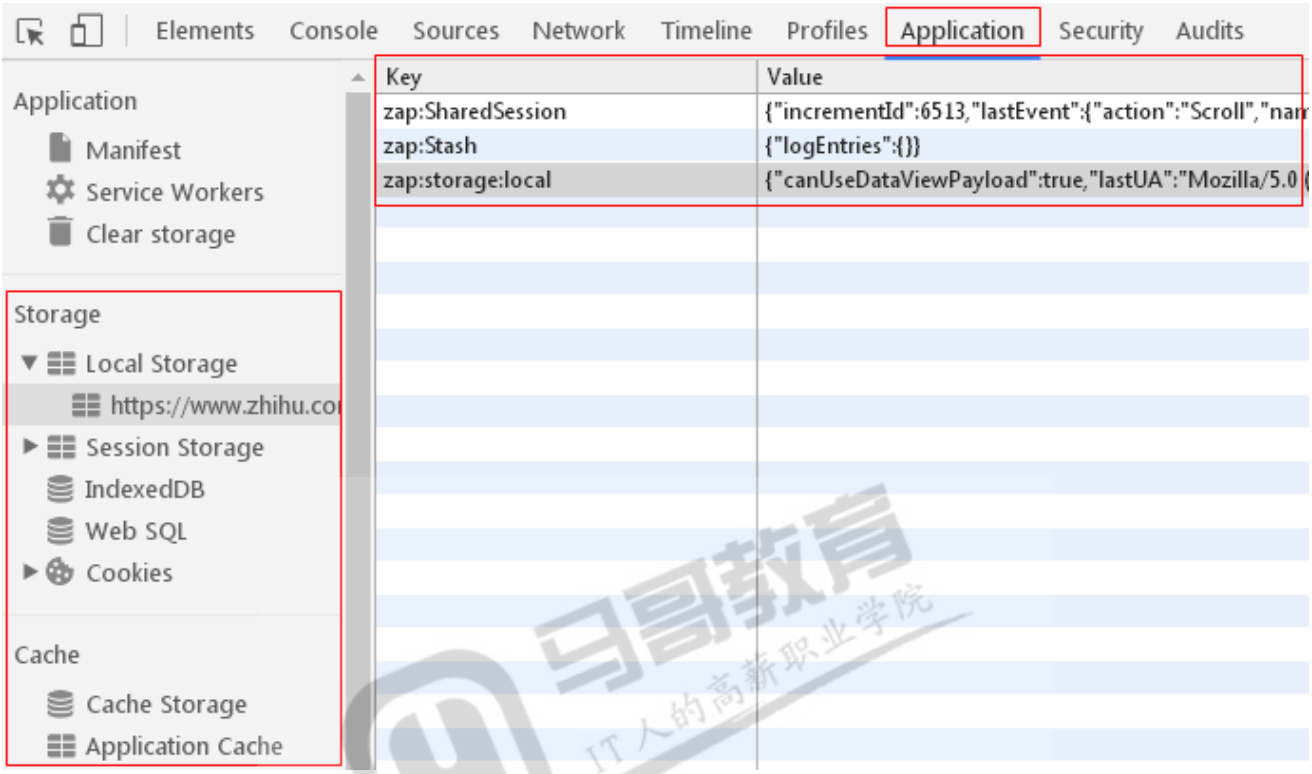
使用正确的邮箱、密码登录，返回了json数据，response.data中可以看到token、user。

---

token持久化——LocalStorage

使用LocalStorage来存储token。
LocalStorage是浏览器端持久化方案之一，HTML5标准增加的技术。
LocalStorage 是为了存储得到的数据，例如Json。
数据存储就是键值对。
数据会存储在不同的域名下面。
不同浏览器对单个域名下存储数据的长度支持不同，有的最多支持2MB。

Chrome浏览器中查看，如下



SessionStorage和LocalStorage差不多，它是会话级的，浏览器关闭，会话结束，数据清除。

IndexedDB

- 一个域一个datatable
- key-valuede检索方式
- 建立在关系型的数据模型之上，具有索引表、游标、事务等概念

store.js

store.js 是一个兼容所有浏览器的 LocalStorage 包装器，不需要借助 Cookie 或者 Flash。
store.js 会根据浏览器自动选择使用 localStorage、globalStorage 或者 userData 来实现本地存储功能。

安装

```
$ npm i store
```

测试代码

```
let store = require('store');

store.set('user', 'wayne');
console.log(store.get('user'));

store.remove('user');
```

```
console.log(store.get('user')); // undefined
console.log(store.get('user','a')); // a

store.set('user', {name:'wayne',age:30});
console.log(store.get('user').name);

store.set('school', {name:'magedu'});

store.each(function(value, key) { // 注意这里key、value是反的
    console.log(key, '-->', value)
});

store.clearAll();

console.log(store.get('user')); // undefined
```

安装store的同时，也安装了expire过期插件，可以在把kv对存储到LS中的时候顺便加入过期时长。

```
store.addPlugin(require('store/plugins/expire')); // 过期插件
store.set('token',res.data.token, (new Date()).getTime() + (8*3600*1000));
```

Mobx状态管理

Redux和Mobx

TodoList项目中，感觉基本功能都实现了，但是，state的控制有点麻烦。

社区提供的状态管理库，有Redux和Mobx。

Redux代码优秀，使用严格的函数式编程思想，学习曲线陡峭，小项目使用的优势不明显。
Mobx，非常优秀稳定的库，简单方便，适合中小项目使用。使用面向对象的方式，容易学习和接受。现在在项目中使用也非常广泛。Mobx和React是一对强力组合。

Mobx官网 https://mobx.js.org/
中文网 http://cn.mobx.js.org/

MobX 是由 Mendix、Coinbase、Facebook 开源，它实现了观察者模式。

观察者模式
观察者模式，也称为发布订阅模式，观察者观察某个目标，目标对象(Obserable)状态发生了变化，就会通知自己内部注册了的观察者Observer。

状态管理

需求

一个组件的onClick触发事件响应函数，此函数会调用后台服务。但是后台服务比较耗时，等处理完，需要引起组件的渲染操作。
要组件渲染，需要使用改变组件的props或state。

1、同步调用

同步调用中，实际上就是等着耗时的函数返回

```
import React from 'react';
import ReactDom from 'react-dom';

class Service {
  handle(n) {
    // 同步
    console.log('pending~~~~~~~~')
    for (let s=new Date(); new Date()-s < n*1000;);
    console.log('done');
    return Math.random();
  }
}

class Root extends React.Component {
  state = {ret:null}
  handleClick(event){
    // 同步返回值
    let ret = this.props.service.handle(4);
    this.setState({ret:ret});
  }

  render() {
    console.log('**********************');
    return (
      <div>
        <button onClick={this.handleClick.bind(this)}>触发handleClick函数</button><br />
        <span style={{color:'red'}}>{new Date().getTime()} Service的handle函数返回值是:
{this.state.ret}</span>
      </div>);
  }
}

ReactDom.render(<Root service={new Service()} />, document.getElementById('root'));
```

这里使用一个死循环来模拟同步调用，来模拟耗时的等待返回的过程。

2、异步调用

思路一、使用setTimeout

使用setTimeout，有2个问题。
1、无法向内部的待执行函数传入参数，比如传入Root实例。
2、延时执行的函数的返回值无法取到，所以无法通知Root

思路二、Promise异步执行

Promise异步执行，如果成功执行，将调用回调。

```
import React from 'react';
import ReactDom from 'react-dom';

class Service {
  handle(obj) {
```

```
    // Promise
    new Promise((resolve, reject) => {
      setTimeout(()=>resolve('OK'), 5000);
    }).then(
      value => { // 使用obj
        obj.setState({ret:(Math.random()*100)});
      }
    )
  }
}

class Root extends React.Component {
  state = {ret:null}
  handleClick(event){
    // 异步不能直接使用返回值
    this.props.service.handle(this);
  }

  render() {
    console.log('**********************');
    return (
      <div>
        <button onClick={this.handleClick.bind(this)}>触发handleClick函数</button><br />
        <span style={{color:'red'}}>{new Date().getTime()} Service中修改state的值是:
{this.state.ret}</span>
      </div>);
  }
}

ReactDom.render(<Root service={new Service()} />, document.getElementById('root'));
```

不管render中是否显示state的值，只要state改变，都会触发render执行。

3、Mobx实现

observable装饰器：设置被观察者
observer装饰器：设置观察者，将React组件转换为响应式组件

```
import React from 'react';
import ReactDom from 'react-dom';
import {observable} from 'mobx';
import {observer} from 'mobx-react';

class Service {
  @observable ret = -100;
  handle() {
    // Promise
    new Promise((resolve, reject) => {
      setTimeout(()=>resolve('OK'), 2000);
    }).then(
      value => {
        this.ret = (Math.random()*100);
        console.log(this.ret);
```

```
      }
    )
  }
}

@observer // 将react组件转换为响应式组件
class Root extends React.Component {
  //state = {ret:null} // 不使用state了
  handleClick(event){
    // 异步不能直接使用返回值
    this.props.service.handle(this);
  }

  render() {
    console.log('**********************');
    return (
      <div>
        <button onClick={this.handleClick.bind(this)}>触发handleClick函数</button><br />
        <span style={{color:'red'}}>{new Date().getTime()} Service中修改state的值是:
{this.props.service.ret}</span>
      </div>);
  }
}

ReactDom.render(<Root service={new Service()} />, document.getElementById('root'));
```

Service中被观察者ret变化，导致了观察者调用了render函数。

被观察者变化不引起渲染的情况：将上例中的`{this.props.service.ret}`注释`{/*this.props.service.ret*/}`。可以看到，如果在render中不使用这个被观察者，render函数就不会调用。

在观察者的render函数中，一定要使用这个被观察对象。

---

login登录功能代码实现

```
import axios from 'axios';
import store from 'store'
import {observable} from 'mobx';

// 过期插件
store.addPlugin(require('store/plugins/expire'));

export default class UserService {
    @observable loggedin = false; //+ 被观察者

    login (email, password) {
        console.log(email, password);

        axios.post('/api/user/login', {
            email:email,
            password:password
```

```
        })/* dev server会代理 */
        .then(
            response => { // 此函数要注意this的问题
                console.log(response.data);
                console.log(response.status);
                //+ 存储token，注意需要重开一次chrome的调试窗口才能看到
                store.set('token',
                response.data.token,
                (new Date()).getTime() + (8*3600*1000));
                this.loggedin = true; //+ 修改被观察者
            }
        ).catch(
            function (error) {
                console.log(error);
            }
        )
    }
}
```

```
//component/login.js
import React from 'react';
import {Link, Redirect} from 'react-router-dom';
import '../css/login.css'
import UserService from '../service/user';
import {observer} from 'mobx-react';

const userService = new UserService();

export default class Login extends React.Component {
    render () {
        return <_Login service={userService} />;
    }
}

@observer
class _Login extends React.Component {
    handleClick(event) {
        event.preventDefault();
        let fm = event.target.form;
        this.props.service.login(
            fm[0].value, fm[1].value
        );
    }

    render() {
        if (this.props.service.loggedin) {
            return <Redirect to='/' />; //+ 跳转
        }
        return (<div className="login-page">
            <div className="form">
                <form className="login-form">
                    <input type="text" placeholder="邮箱" value='tom@magedu.com' />
```

```
                    <input type="password" placeholder="密码" value='abc' />
                    <button onClick={this.handleClick.bind(this)}>登录</button>
                    <p className="message">还未注册? <Link to="/reg">请注册</Link></p>
                </form>
            </div>
        </div>);
    }
}
```

注意，测试时，开启Django所写后台服务。

测试成功，成功登录，写入Localstorage，也实现了跳转