测试工具及环境:

jest+babel-jest+enzyme 测试环境的安装见前的端测试环境搭建文档

测试流程:

手动写 test 文件，对前端主要部件进行测试，主要涉及演出浏览、详情、拍卖、推荐等部件，没有测试非核心功能如登录、注册等

测试详细结果:

1.测试 DetailShowTab

```
PASS  src/test/DetailShowTab.test.js (5.275s)
  测试DetailShowTab组件
    √ 1.测试Detail组件能被正常渲染 (7ms)
    √ 2.测试callback函数 (10ms)

  console.log src/components/DetailShowTab.js:7
    1

------------------|----------|----------|----------|----------|-------------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------|----------|----------|----------|----------|-------------------|
All files         |      100 |      100 |      100 |      100 |                   |
 DetailShowTab.js |      100 |      100 |      100 |      100 |                   |
------------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        6.155s
Ran all test suites matching /DetailShowTab.test.js/i.
```

由于这个控件相对简单，而且没有为了处理特殊情况而产生的分支，所以覆盖率可以达到100%

2.测试 auctionCard

```
> frontend@0.1.0 test D:\summer_project\7.20\frontend
> jest --colors --coverage "AuctionCard.test.js"

 PASS  src/test/AuctionCard.test.js (8.058s)
  测试AuctionCard组件
    √ 1.测试AuctionCard组件能被正常渲染,且能正常拍卖 (75ms)
    √ 2.测试拍卖事件还没开始 (16ms)
    √ 3.测试拍卖已经结束 (19ms)
    √ 4.测试模拟点击购买 (91ms)
    √ 5.测试componentDidMount (30ms)
    √ 6.测试constructor (19ms)
    √ 7.测试flushState (22ms)
    √ 8.测试getTimeType (18ms)
    √ 9.测试componentWillMount (18ms)
    √ 10.测试componentWillUnmount (18ms)
    √ 11.测试beforeunload (19ms)
    √ 12.测试giveOffer (19ms)
    √ 13.测试giveOffer (18ms)
    √ 14.测试giveOffer (17ms)
```

```
------------------|----------|----------|----------|----------|-------------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------|----------|----------|----------|----------|-------------------|
All files         |    64.47 |    66.67 |    40.54 |    64.47 |                   |
 components       |    75.27 |    66.67 |    68.75 |    75.27 |                   |
  AuctionCard.js  |    75.27 |    66.67 |    68.75 |    75.27 |... 89,192,200,227 |
 services         |     45.1 |      100 |    17.65 |     45.1 |                   |
  goodsService.js |       50 |      100 |       25 |       50 |... 21,25,26,30,31 |
  userService.js  |    40.74 |      100 |    11.11 |    40.74 |... 35,39,40,44,45 |
 utils            |     62.5 |      100 |       25 |     62.5 |                   |
  ajax.js         |    57.14 |      100 |       25 |    57.14 |          15,18,21 |
  history.js      |      100 |      100 |      100 |      100 |                   |
------------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        9.091s
```

我测试了组件的渲染和基本的函数，但是可以看到覆盖率还是不高，基本都是一些防止用户骚操作或者并发情况下的特殊情况。由于单独一个前端无法测试并发情况，所以有些不能覆盖。我把没覆盖的代码罗列到下边并分析原因：

```
187         const callback = (data)=>{
188             if(data.status>=0){
189                 message.success( content: '恭喜您竞价成功');
190             }
191             else{
192                 message.error( content: '抱歉，有人更早比您出了更高的价格，请刷新页面重新出价');
193             }
194         }
195         updateAuction(json,callback);
196     }
197
```

第 192 行没覆盖到的原因是，此段代码是防止在同一个 interval 中有多名用户同时拍卖，后端会给出价较低的用户发送一个 message，因为我这里只有一个前端，所以覆盖不到

```
198     isTheCandidate=()=>{
199         if(this.state.isTheCandidate){
200             return <div>您是最高出价人</div>
201         }
202         else{
203             return<div>您不是最高出价人</div>
204         }
205     }
```

第 200 行覆盖不到的原因是单独一个前端只有一个测试用户，两种情况没法完全覆盖

```
        if(triggerFlag === false){
            startTrigger = setInterval( handler: ()=>{
                this.flushState();
            }, timeout: 1000);
            triggerFlag = true;
        }
```

第 227 行没法覆盖的原因是在测试时，我只生成了一次 AuctionCard，没法利用 setInterval 动态刷新数据。flushState 的作用就是动态刷新数据，我进行了单独测试。

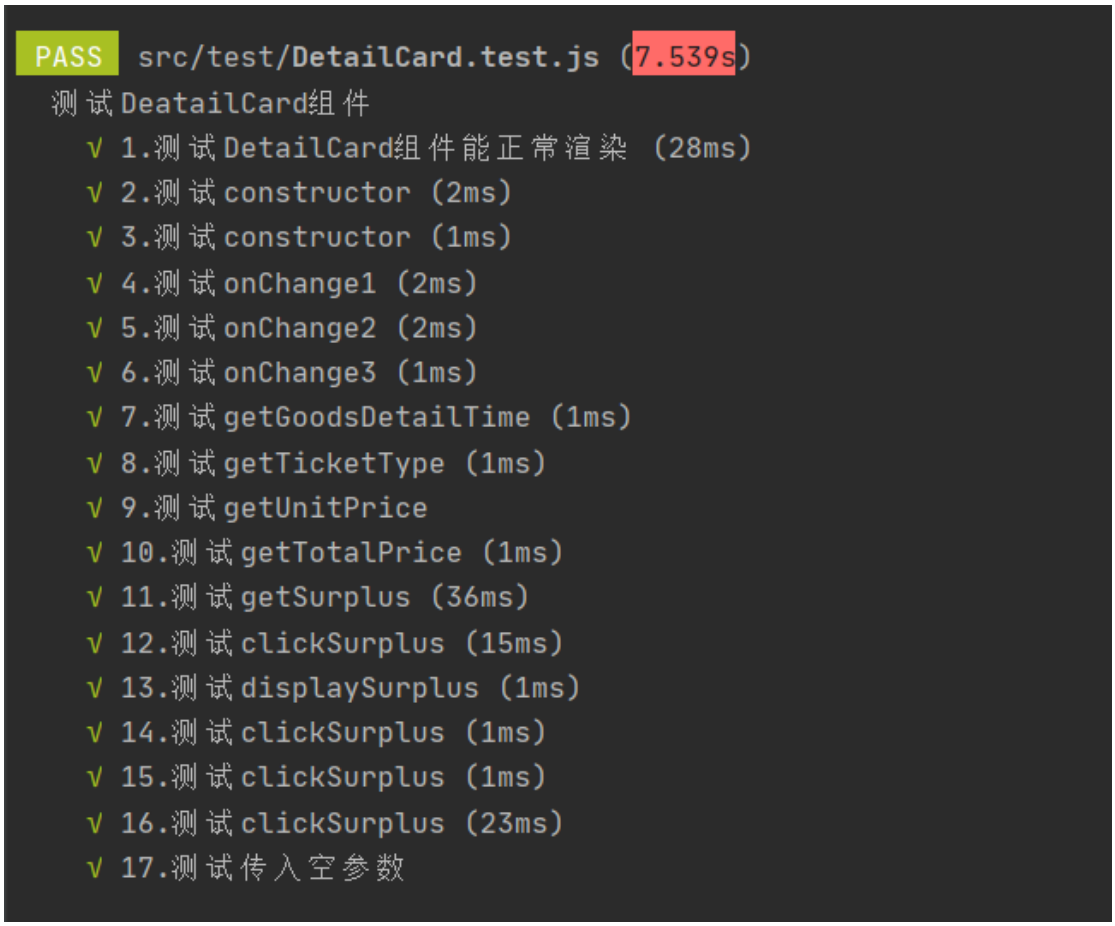
3.测试 RecommedList

```
PASS src/test/RecommendationList.test.js
  测试Recommendation组件
    √ 1.测试Recommendation组件能被正常渲染 (4ms)

------------------|----------|----------|----------|----------|-------------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------|----------|----------|----------|----------|-------------------|
All files         |      100 |      100 |      100 |      100 |                   |
 RecommendList.js |      100 |      100 |      100 |      100 |                   |
------------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        3.627s
Ran all test suites matching /RecommendationList.test.js/i.
```

这个组件目前也相对简单，没有难以覆盖的分支，所以覆盖率也能到 100%

4.测试 DetailCard



```
------------------|----------|----------|----------|----------|-------------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------|----------|----------|----------|----------|-------------------|
All files         |    60.51 |       48 |    53.19 |    59.26 |                   |
 components       |    67.65 |       48 |    84.62 |    66.15 |                   |
  DetailCard.js   |    67.65 |       48 |    84.62 |    66.15 |... 62,265,268,281 |
 services         |    41.18 |      100 |    11.76 |    41.18 |                   |
  goodsService.js |    41.67 |      100 |     12.5 |    41.67 |... 31,35,36,40,41 |
  userService.js  |    40.74 |      100 |    11.11 |    40.74 |... 35,39,40,44,45 |
 utils            |     62.5 |      100 |       25 |     62.5 |                   |
  ajax.js         |    57.14 |      100 |       25 |    57.14 |          15,18,21 |
  history.js      |      100 |      100 |      100 |      100 |                   |
------------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       17 passed, 17 total
Snapshots:   0 total
Time:        8.468s
```

对于详情界面也进行了对渲染和函数的测试，对于有些不能覆盖到的代码，我在下面将进行分析：

```
componentDidMount() {
    const callback = (data) => {
        this.setState( state: {goodsData:data.data});
        this.setState( state: {goodsDetailTime:data.data.goodsDetails[0].time});
        this.setState( state: {ticketsType:data.data.goodsDetails[0].ticketType});
        this.setState( state: {unitPrice:data.data.goodsDetails[0].price});
        this.setState( state: {totalPrice:data.data.goodsDetails[0].price})
        this.getGoodsDetailTime(data.data);
        this.getTicketType(data.data,data.data.ticketsType);
        this.getUnitPrice(data.data,data.data.ticketsType);
    };
    if(this.props.info === null)
        return;
    const requestData = {goodsId:this.props.info};
    getGoodsByGoodsId(requestData,callback);

```

首先 62 行这里，测试 componentDidMount 竟然不能调用 getDetailTime,我没太想明白是因为什么，还需要继续探索

```
        const callback = (data)=>{
            if(data.status>=0){
                this.setState( state: {orderId:data.data.orderId});
                message.success( content: data.msg + "请至订单界面查询订单信息" +"\n"+"您的订单号是"+data.data.orderId);
            }
            else{
                message.error(data.msg);
            }
        }
        addOrder(json,callback);
    }
```

265 行因为没法与后端交互，我也没写对应的 mock 函数，所以没法向后端发送 addOrder 的请求

```
256                 const callback = (data)=>{
257                     if(data.status>=0){
258                         this.setState( state: {orderId:data.data.orderId});
259                         message.success( content: data.msg + "请至订单界面查询订单信息" +"\n"+"您的订单号是"+data.data.orderId);
260                     }
261                     else{
262                         message.error(data.msg);
263                     }
264                 }
265                 addOrder(json,callback);
266             }
267             else{
268                 message.error( content: "该选项无货");
269             }
270         }
271         else{
272             message.error( content: "请登录");
273         }
274     }
```

268 行因为我没选取无货的测试数据，所以没法覆盖，但是在实际使用中可以看到已经可以处理无货的情况，不影响使用

```
278     render(){
279         if(this.props.info === null)
280             return null;
281         if(this.state.user===null)
282             return <Card className={"detail-card"}>请登录</Card>;
```

281 行没测试到没登录的情况，这一点我们还在商讨如果没登陆是否可以看到详情页面，所以这一点可以先放置一下。

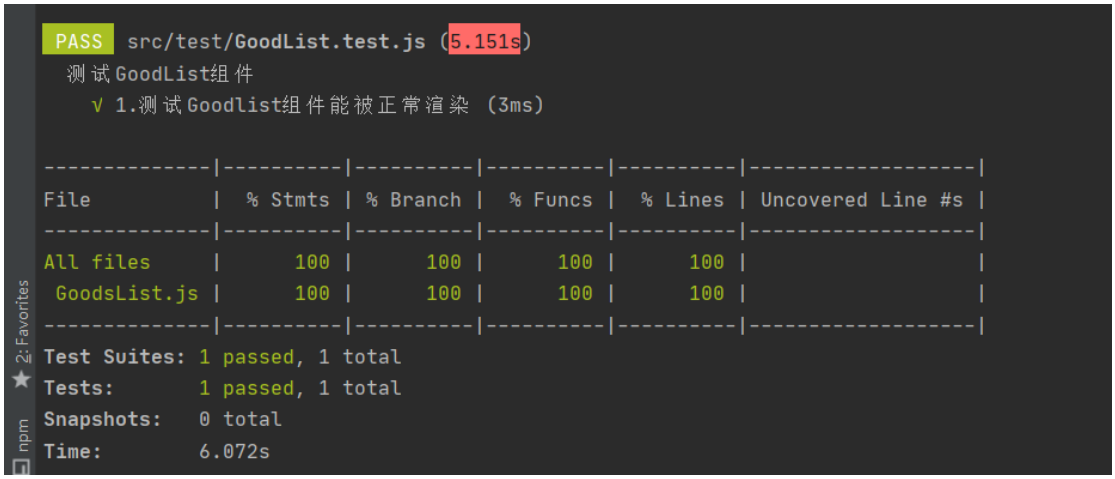
5.测试 GoodsList

```
PASS src/test/GoodList.test.js (5.491s)
  测试GoodList组件
    √ 1.测试Goodlist组件能被正常渲染 (3ms)

--------------|----------|----------|----------|----------|-------------------|
File          |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
--------------|----------|----------|----------|----------|-------------------|
All files     |      100 |      100 |      100 |      100 |                   |
 GoodsList.js |      100 |      100 |      100 |      100 |                   |
--------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        6.385s
Ran all test suites matching /GoodList.test.js/i.
```

因为用的是 shallow 不渲染下层的 goods，所以覆盖率很容易到达 100%

6.测试 Goods



Goods 与 GoodsList 差不多，都是只有一个参数，没有较多的分支结构，所以覆盖率很容易达到 100%。

测试总结:

本次测试主要掌握了使用 jest+enzyme 工具对 react 前端进行测试，其中涉及到的组件的参数、state 等问题有很多坑，都已经写入前端踩坑指南.md 文档中了，以后再遇到相关问题可以先查看自己写的经验文档。另外在测试中我发现单纯的前端测试，很难使得代码覆盖率到达 100%，因为有很多分支需要后端的支持，不然前端就需要写很复杂的 mock 函数，而这也是我本次单元测试所欠缺的，还需要继续学习。