



---

区块链食品溯源系统

## 区块链食品溯源实战案例-2

---

实验手册

V1.0



---

## 目录

实验 2 区块链食品溯源平台的智能合约 .....	1
1.启动 WeBASE-Front .....	1
2.编写智能合约 .....	1
3.编译和部署智能合约 .....	4
实验 3 区块链食品溯源平台前后端开发 .....	6
1.修改代码中与智能合约交互相关的初始信息 .....	6
2.各角色的用户添加食品信息 .....	7
3.获取食品的相关信息 .....	8
4.获取食品的溯源信息 .....	9
5.实现食品溯源的前端代码 .....	11
6.测试 .....	23

## 实验 2 区块链食品溯源平台的智能合约

### 实验概述:

本实验需完成 Trace 智能合约的开发。

### 任务步骤:

#### 1.启动 WeBASE-Front

(1) 打开终端。此步骤不在赘述，参考平台搭建与运维实战案例；

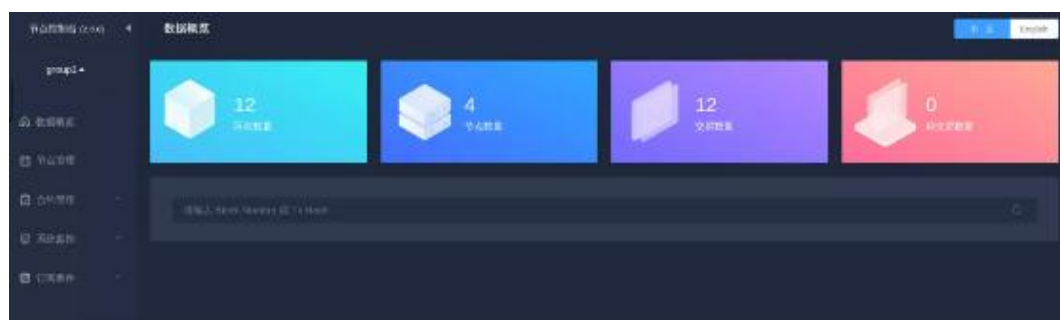
(2) 进入 fisco 目录，用以下命令启动节点：

```
bash nodes/127.0.0.1/start_all.sh
```

(3) 然后再切换到 webase-front，用以下命令启动 webase-front:

```
bash start.sh
```

打开火狐浏览器，输入 127.0.0.1:5002/WeBASE-Front 地址打开 webase-front。如下图所示。



#### 2.编写智能合约

依次点击合约管理——>合约 IDE，然后点击箭头加载全部合约，共包括六个合约。其中我们要完成的主合约代码为 Trace.sol。



---

此合约是一个食品工厂合约，负责具体食品溯源信息的生成。需要用 `import` 来导入食品信息的合约 `FoodInfoItem.sol`，中间商合约 `Distributor.sol`，超市合约 `Retailer.sol` 以及生厂商合约 `Producer.sol`。【已经完成的内容】

首先，声明 `Trace` 合约，继承自 `Producer`，`Distributor` 和 `Retailer` 合约。

```
1  pragma solidity >=0.4.22 <0.7.0;
2  pragma experimental ABIEncoderV2;
3  import "./FoodInfoItem.sol";
4  import "./Distributor.sol";
5  import "./Producer.sol";
6  import "./Retailer.sol";
7
8
9  //食品工厂合约，负责具体食品溯源信息的生成
10 - contract Trace is Producer, Distributor, Retailer{
11
```

其次，声明两个变量，一个是映射，通过食品溯源 `id` 映射到具体的食品溯源合约地址；另一个是数组，用于存放所有的食品。

```
10 - contract Trace is Producer, Distributor, Retailer{
11
12     mapping (uint256 => address) foods; //食品溯源id到具体食品溯源合约的映射表
13     uint[] foodList;
14
```

构造函数的生成。构造生厂商、中间商以及超市三个角色。

```
    //构造函数
    constructor(address producer, address distributor, address retailer)
    public Producer(producer)
    Distributor(distributor)
    Retailer(retailer){
    }
```

①生成食品溯源信息的接口，通过生厂商来添加新的食品。包括食品名称，溯源码，当前用户名称，质检情况四个字段。根据以下步骤完成此函数的编写，具体内容如下图所示。

- a、首先需要检查溯源码是否存在，存在即不可以继续添加；
- b、用 `new FoodInfoItem` 合约实例化添加一个新食品的信息；
- c、将新添加的食品信息的合约地址，建立和食品溯源码的映射关系；
- e、然后将新食品放在数组 `foodList` 中；
- f、返回此新添加食品的合约地址。

```

22 //生成食品溯源信息接口
23 //只有Producer能调用
24 //name 食品名称
25 //traceNumber 食品溯源id
26 //traceName 当前用户名称
27 //quality 当前食品质量
28 function newFood(string name, uint256 traceNumber, string traceName, uint8 quality)
29 public onlyProducer returns(address)
30 {
31     require(foods[traceNumber] == address(0), "traceNumber already exist");
32     FoodInfoItem food = new FoodInfoItem(name, traceName, quality, msg.sender);
33     foods[traceNumber] = food;
34     foodList.push(traceNumber);
35     return food;
36 }
37

```

②编写食品分销过程中增加溯源信息的接口。其中包括食品的溯源 id，当前用户的名称，质检的情况。根据以下步骤完成代码的编写，具体如下图所示。

- a、首先需要检查溯源码是否存在，存在即不可以继续添加；
- b、利用 FoodInfoItem 方法获取对应溯源码的食品信息，然后再利用 addTraceInfoByDistributor 方法反复添加食品的流转信息。【递归】

```

//食品分销过程中增加溯源信息的接口
//只有Distributor能调用
//traceNumber 食品溯源id
//traceName 当前用户名称
//quality 当前食品质量
function addTraceInfoByDistributor(uint256 traceNumber, string traceName, uint8 quality)
public onlyDistributor returns(bool) {
    require(foods[traceNumber] != address(0), "traceNumber does not exist");
    return FoodInfoItem(foods[traceNumber]).addTraceInfoByDistributor(traceName, msg.sender,
    quality);
}

```

③在食品出售过程中增加食品溯源信息的接口。其中包括食品溯源 id，当前用户名称，当前食品质量。根据以下步骤完善代码，具体如下图所示。

- a、首先需要检查溯源码是否存在，存在即不可以继续添加；
- b、用递归的方法，合约调用者（即当前用户）添加食品流转信息。

```

//食品出售过程中增加溯源信息的接口
//只有Retailer能调用
//traceNumber 食品溯源id
//traceName 当前用户名称
//quality 当前食品质量
function addTraceInfoByRetailer(uint256 traceNumber, string traceName, uint8 quality)
public onlyRetailer returns(bool) {
    require(foods[traceNumber] != address(0), "traceNumber does not exist");
    return FoodInfoItem(foods[traceNumber]).addTraceInfoByRetailer(traceName, msg.sender, quality);
}

```

④获取食品溯源的信息接口。其中 string[]用于保存食品流转过程中的各个阶段的相关信息，address[]为保存食品流转过程中各个阶段的用户地址信息，uint8[]保存食品流转过程中各个阶段的状态变化。根据以下步骤实现编写代码，具体如下图所示。

- a、getTraceInfo()方法是获取食品溯源信息，首先需要检查溯源码是否存在，存在即不可以继续添加；
- b、用 FoodInfoItem 方法获取溯源码对应的食品合约地址，然后用递归获取食品溯源完整信息；

- c、getFood()方法是获取食品信息的方法，首先需要检查溯源码是否存在，存在即不可以继续添加；
- d、用 FoodInfoItem 方法获取溯源码对应的食品合约地址，然后用递归获取所有的食品信息；
- e、getAllFood 方法是获取所有的食品信息，直接返回食品列表即可。

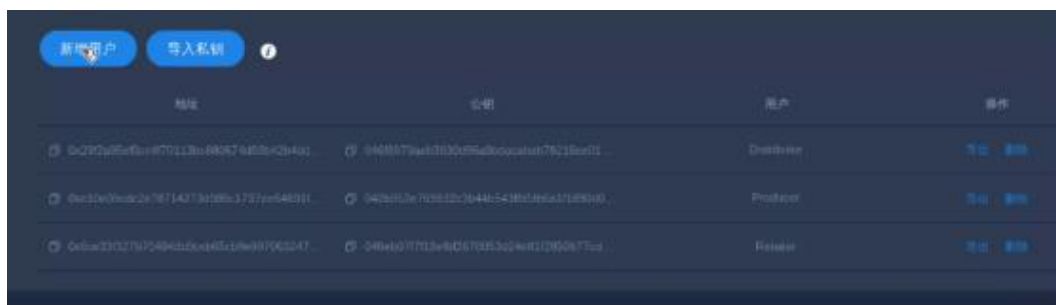
```
//获取食品溯源信息接口
//string[] 保存食品流转过程中各个阶段的相关信息
//address[] 保存食品流转过程各个阶段的用户地址信息（和用户——对应）
//uint8[] 保存食品流转过程中各个阶段的状态变化
function getTraceInfo(uint256 traceNumber) public constant returns(uint[], string[], address[], uint8[]) {
    require(foods[traceNumber] != address(0), "traceNumber does not exist");
    return FoodInfoItem(foods[traceNumber]).getTraceInfo();
}

function getFood(uint256 traceNumber) public constant returns(uint, string, string, string, address, uint8) {
    require(foods[traceNumber] != address(0), "traceNumber does not exist");
    return FoodInfoItem(foods[traceNumber]).getFood();
}

function getAllFood() public constant returns (uint[]) {
    return foodList;
}
```

至此，完整的溯源合约代码就都编写完了。依赖的智能合约实现感兴趣的同学可以课后继续学习，不再赘述。

接下来需要创建三个用户，分别是 **Producer**，**Distributor**，以及 **Retailer**。如下图所示。并记录相应的账户地址。【合约部署时需要】



地址	用户名	用户	操作
0x297a95efb0e870113b080670403e020401	04485779a13030d56d30cc0a0a76218e011	Distributor	查看 删除
0xc12e09e0c2a76714273a10b03727e064031	042b0f3e703022c3d44b54380f0a0a371890a0	Producer	查看 删除
0xc0e3332797549e0220e0405c19e007062247	040e0077713e0d570353c02e0012950e770a	Retailer	查看 删除

### 3.编译和部署智能合约

点击合约右上角的编译按钮对合约进行编译。如下图所示，编译生成的合约地址以及 ABI 等信息如下，需要记录 **contractAddress**，**contractName**，以及 **abi**，需要将此三项内容替换为合约代码中的相关信息。



接着点击部署，将合约部署到本地的 fisco 联盟链中。其中特别要注意用户选择的是哪一个，此处选择 **producer**。如下图所示。



将之前创建的生产商、中间商及超市这三个账户地址依次对照的填入下图的参数中。



选择用户地址

用户: 9cdc2e78714273d986c1737ee64691ffb31 (Produ...)

CNS:

参数:

❗ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如：['aaa','bbb']和[100,101]；如果数组参数包含双引号，需转义，例如：['aa a','bbb','ccc']。

取消 确认

至此，智能合约的部署就全部完成了。

## 实验 3 区块链食品溯源平台前后端开发

### 实验概述：

本实验主要编写区块链溯源平台的后端代码。前端代码不作为本次实验的重点内容，可自行探索研究即可。没有采用 MVC 架构，全部的代码都集成在 `IndexController` 中。

### 任务步骤：

#### 1.修改代码中与智能合约交互相关的初始信息

拉取初始化的代码，打开 IDEA，加载项目 `backend`，依次打开项目中的文件夹 `src`—>`main`—>`java`—>`com.zgxt.demo`—>`controller`—>`IndexController.java`。此项目没有使用 MVC 设计模式，所有的逻辑都集成在了 `IndexController.java` 文件中。

依次将三个账户地址，以及合约地址，合约 ABI 的信息填入代码对应的参数中。本代码没有注册与登录功能，直接获取用户的地址，需要在代码中手动添加账户地址。如下图所示。



```

private static final String CONTRACT_NAME = "Trace"; 合约地址
private static final String CONTRACT_ADDRESS = "0xa01c89b9c110c3e9d5872dd8ca759b95790c7fb8"; abi
private static final String CONTRACT_ABI = "[{\"constant\":true,\"inputs\":[{\"name\":\"traceNumber\",\"type\":\"uint256\"}],\"outputs\":[{\"name\":\"\",\"type\":\"string\"}],\"payable\":false,\"stateMutability\":\"view\"}]";

//填写用户地址信息
private static final String PRODUCER_ADDRESS = "0xe10e09c0c2e78714273d996c1737ee64691fb31";
private static final String DISTRIBUTOR_ADDRESS = "0x29f2a95ef0cc4f78113bc880674d03b42b4dd18e";
private static final String RETAILER_ADDRESS = "0x6ce33f3327a70494dc8ce065cb8e9970632474483";

```

除此之外，通过 HTTP 访问合约的 URL 为下图所示的地址（IP 和端口要根据部署的联盟链环境的地址修改）。

```

24 @Controller
25 public class IndexController {
26     //填写WeBASE-Front地址，用于后续交互
27     private static final String URL = "http://127.0.0.1:5802/WeBASE-Front/trans/handle";
28
29     private static final String CONTRACT_NAME = "Trace";
30     private static final String CONTRACT_ADDRESS = "0xd1b887874e120b9fb6b19b0208dfa8903279c781";
31     private static final String CONTRACT_ABI = "[{\"constant\":true,\"inputs\":[{\"name\":\"traceNumber\",\"type\":\"uint256\"}],\"outputs\":[{\"name\":\"\",\"type\":\"string\"}],\"payable\":false,\"stateMutability\":\"view\"}]";
32
33     //填写用户地址信息
34     private static final String PRODUCER_ADDRESS = "0x949bc825b874c257c0be2bfab9683d3c33e24b74";
35     private static final String DISTRIBUTOR_ADDRESS = "0x330b6c766c1e40249348e9538d735b99226f7213";
36     private static final String RETAILER_ADDRESS = "0xab4c3e39b3aef8cad196c4519a1954ed47532bf";

```

## 2.各角色的用户添加食品信息

### （1）农场添加食品的生产信息

函数 **produce** 用来添加食品的生产信息。参数包括 **traceNumber**，即食品的溯源 ID，食品溯源过程中的标识符；**foodName**：食品名称；**traceName**：用户名，食品流转过程中各个阶段的用户名；**quality**：当前的食品质量（0-优秀，1-合格，2-不合格）。

- 通过 **jsonParam** 获取输入的食品信息，即以上四个关于食品生产信息的值；
- 然后通过访问 **webase-front** 的智能合约中函数 **newfood**，传入以上食品生产信息的参数来添加一个新食品；

```

97 int trace_number = (int) jsonParam.get("traceNumber");
98 String food_name = (String) jsonParam.get("foodName");
99 String trace_name = (String) jsonParam.get("traceName");
100 int quality = (int) jsonParam.get("quality");
101
102 JSONArray params = JSONArray.parseArray("{\""+food_name+"\",\""+trace_number+"\",\""+trace_name+"\",\""+quality+"}");
103 JSONObject _jsonObj = new JSONObject();
104 _jsonObj.put("contractName", CONTRACT_NAME);
105 _jsonObj.put("contractAddress", CONTRACT_ADDRESS);
106 _jsonObj.put("contractAbi", JSONArray.parseArray(CONTRACT_ABI));
107 _jsonObj.put("user", PRODUCER_ADDRESS);
108 _jsonObj.put("funcName", "newFood");
109 _jsonObj.put("funcParam", params);

```

### （2）中间商添加食品流转的信息

函数 `add_trace_by_distributor` 用来在中间商处添加相应食品的流转信息。参数包括 `traceNumber`，即食品的溯源 ID，`traceName`：用户名，食品流转过程中各个阶段的用户名（可以取不同的名字）；`quality`：当前的食品质量（0-优秀，1-合格，2-不合格）。

- 通过 `jsonParam` 获取输入的食品信息，即以上三个关于食品流转信息的值；
- 然后通过访问 `webase-front` 智能合约中函数 `addTraceInfoByDistributor`，传入以上食品生产信息的参数在中间商处添加一个食品流转的信息；



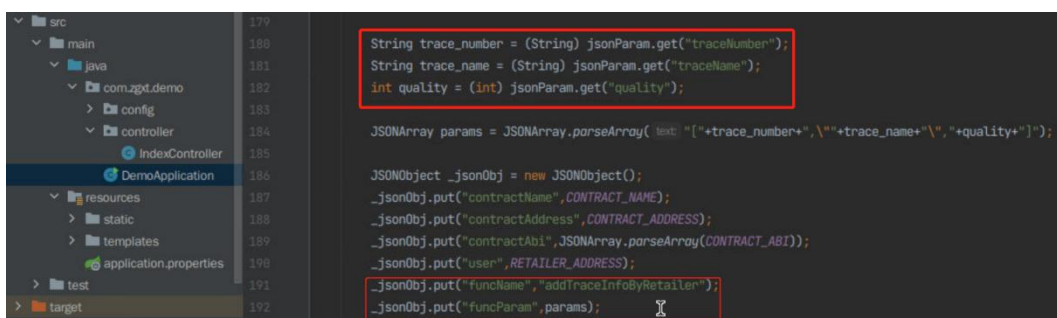
```
String trace_number = (String) jsonParam.get("traceNumber");
String trace_name = (String) jsonParam.get("traceName");
int quality = (int) jsonParam.get("quality");

JSONArray params = JSONArray.parseArray("{\""+trace_number+"\",\""+trace_name+"\", \""+quality+"\"}");
JSONObject _jsonObj = new JSONObject();
_jsonObj.put("contractName", CONTRACT_NAME);
_jsonObj.put("contractAddress", CONTRACT_ADDRESS);
_jsonObj.put("contractAbi", JSONArray.parseArray(CONTRACT_ABI));
_jsonObj.put("user", DISTRIBUTOR_ADDRESS);
_jsonObj.put("funcName", "addTraceInfoByDistributor");
_jsonObj.put("funcParam", params);
```

### （3）超市添加食品流转的信息

函数 `add_trace_by_retailer` 用来在超市处添加相应食品的流转信息。参数包括 `traceNumber`，即食品的溯源 ID，`traceName`：用户名，食品流转过程中各个阶段的用户名（可以取不同的名字）；`quality`：当前的食品质量（0-优秀，1-合格，2-不合格）。

- 通过 `jsonParam` 获取输入的食品信息，即以上三个关于食品流转信息的值；
- 然后通过访问 `webase-front` 智能合约中函数 `addTraceInfoByRetailer`，传入以上食品生产信息的参数在中间商处添加一个食品流转的信息；



```
String trace_number = (String) jsonParam.get("traceNumber");
String trace_name = (String) jsonParam.get("traceName");
int quality = (int) jsonParam.get("quality");

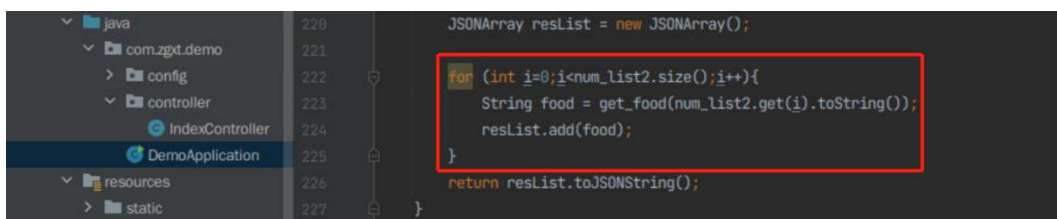
JSONArray params = JSONArray.parseArray("{\""+trace_number+"\",\""+trace_name+"\", \""+quality+"\"}");

JSONObject _jsonObj = new JSONObject();
_jsonObj.put("contractName", CONTRACT_NAME);
_jsonObj.put("contractAddress", CONTRACT_ADDRESS);
_jsonObj.put("contractAbi", JSONArray.parseArray(CONTRACT_ABI));
_jsonObj.put("user", RETAILER_ADDRESS);
_jsonObj.put("funcName", "addTraceInfoByRetailer");
_jsonObj.put("funcParam", params);
```

## 3. 获取食品的相关信息

### （1）获取所有食品信息

函数 `getlist` 用于获取所有的食品信息。首先需要从链上获取所有食品信息（用 `get_food_list` 函数实现）。然后对其进行 `for` 循环迭代查找所有的食品，从而获取所有食品信息。如下图所示。



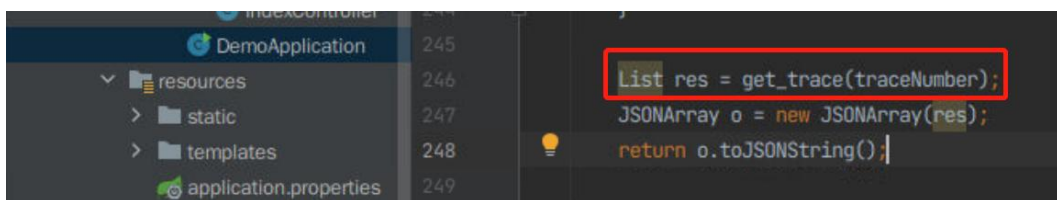
## (2) 获取某个食品的当前信息

`food` 实际是利用 `get_food`（从链上获取某个食品的基本信息）来实现的。在下面进行讲解。

## 4. 获取食品的溯源信息

### (1) 获取某个食品的溯源信息

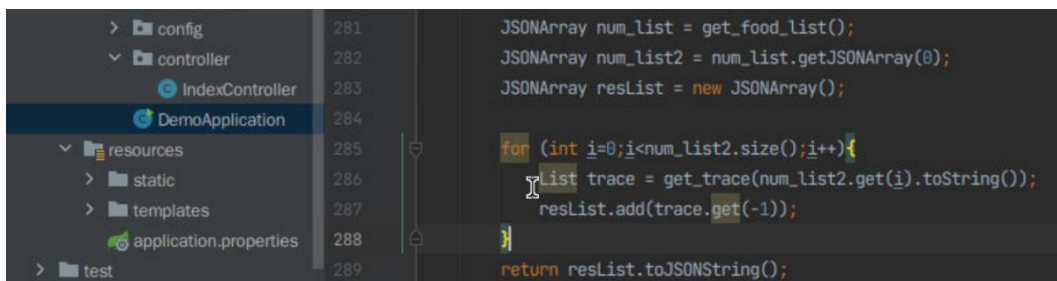
函数 `trace` 用来获取某个食品的溯源信息，根据 `traceNumber` 溯源码获取食品信息。首先需要用 `get_trace`（从链上获取某个食品的溯源信息）函数获取食品的完整信息，将得到的食品信息转换为一个对象输出。如下图所示补全代码。



### (2) 获取所有食品的最新溯源信息

函数 `get_latest` 用来获取所有食品的最新溯源信息，首先需要从链上获取所有食品信息 `get_food_list`；接着迭代对象 `num_list2` 从而获取食品的所有信息，如下图所示。

**注意：**此部分是从链上获取信息，即需要调用 `get_trace`，与 `get_food` 方法加以区别。



### (3) 从链上获取所有食品信息

从链上获取所有食品信息的函数是 `get_food_list`，即通过与 `webase-front` 交互，调用合约 `getAllFood` 获取所有的食品信息。如下图所示为从链上获取食品信息的方法。



#### (4) 从链上获取某个食品的基本信息

函数 `get_food` 从链上获取特定 `traceNumber` 溯源码的食品信息，如下图所示的方法。

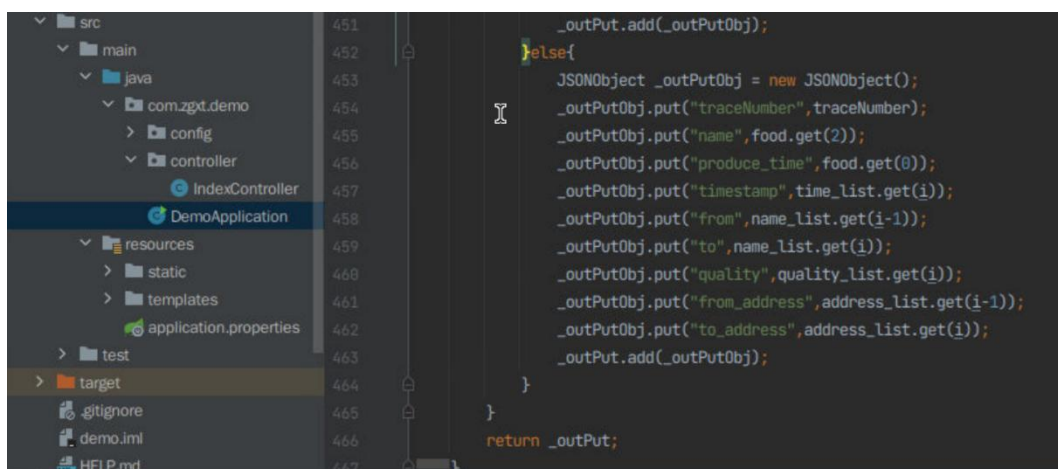
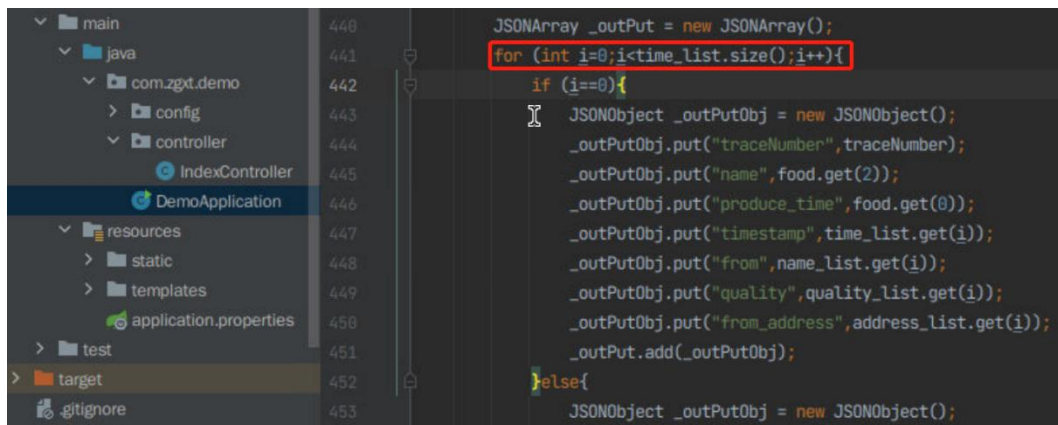


#### (5) 从链上获取某个食品的溯源信息

函数 `get_trace` 是通过首先用上一步的方法获取某个食品的基本信息。获取溯源信息的方法是 `getTraceInfo`，然后需要提取食品溯源中的时间信息 `time_list`，循环迭代得到溯源信息。







至此，关键的后端代码就都完成了，其它未涉及的代码都已经在拉取的初始化代码中实现了，请自行研究探索，不再赘述。

## 5.实现食品溯源的前端代码

前端采用 Vue.js+axios+elementUI 组件实现，没有采用前后端分离技术，所有的前端文件都放在 resources/static/目录下。

### (1) 完善 index.js 代码

此文件是对页面组件的定义，其中使用了动态组件，以及通过状态判断 login 组件的显示。

```
const vue = new Vue({
  // 注册 Header、Login 子组件
  components: {
    Header,
    Login,
  },
});
```

---

```

// 实例绑定的元素
el: '#app',
data: {
  currPage: 1, // 当前页
  pageSize: 10, // 每页显示数据
  total: 0, // 总数据
  showComponent: Farm, // 当前展示的动态组件
  login: false, // 登录状态
  user: "", // 当前用户
},
template: `
  <el-container>

    <!-- 头部导航组件 -->
    <el-header class="header"><Header :user="user" :login="login" @logout="(val)
=> {this.login = val;}" /></el-header>

    <!-- 主要内容组件 -->
    <el-container>
      <el-main>

        <!-- 登录组件 -->
        <Login v-if="!login" @login="(val) => {this.login = true; this.showComponent =
val.component; this.user = val.user;}" />

        <!-- 动态组件，根据 showComponent 动态渲染当前用户 -->
        <component v-else :is="showComponent"
          @logout="(val) => {this.login = val;}"
          @total="(val) => {this.total = val;}"
          :curr-page.sync="currPage" :page-size.sync="pageSize">
        </component>

        <!-- 分页组件 -->
        <el-pagination
          style="text-align: center; margin-top: 10px;"
          v-if="login && showComponent.name !== 'Consumer'"
          background
          :hide-on-single-page="false"
          @size-change="handleSizeChange"
          @current-change="handleCurrentChange"
          :current-page.sync="currPage"
          :page-sizes="[10, 20, 50]"
          :page-size.sync="pageSize"

```

```

        layout="total, sizes, prev, pager, next, jumper"
        :total="total">
      </el-pagination>
    </el-main>
  </el-container>
</el-container>
`,
methods: {
  // 处理每页数据渲染数
  handleSizeChange(val) {
    this.pageSize = val;
  },
  // 处理翻页
  handleCurrentChange(val) {
    this.currPage = val;
  },
}
})

```

参照以上完整的代码，对照下图红色框内的代码完善。在 **methods** 中补全处理每页数据渲染数和处理翻页。



```

56   methods: {
57     // 处理每页数据渲染数
58     handleSizeChange(val) {
59       [red box]
60     },
61     // 处理翻页
62     handleCurrentChange(val) {
63       [red box]
64     },
65   }
66 }

```

## (2) 完善 login.js 代码

此文件主要实现了登录组件，点击各个角色的按钮可以快速登录到角色功能页面。

```

// 登录组件
const Login = {
  data() {
    return {
      // 用户身份
      users: [
        {

```



---

```

    name: '农场',
    userName: 'producer',
    component: Farm,
  },
  {
    name: '中间商',
    userName: 'distributor',
    component: Agent,
  },
  {
    name: '超市',
    userName: 'retailer',
    component: Mall,
  },
  {
    name: '消费者',
    userName: 'consumer',
    component: Consumer,
  },
],
currentUser: null, // 当前用户
address: "", // 角色地址
countdown: 5, // 倒计时时间
loginItem: "", // 登录用户信息
timer: null, // 处理倒计时的定时器
}
},
template: `
<div class="login">
  <!-- 角色选择 -->
  <h3 v-if="currentUser === null">请选择您的角色</h3>
  <el-row :gutter="80" v-if="currentUser === null">
    <el-col :span="6" v-for="(item, index) in users" :key="index">
      <div @click="handleClick(item)">{{ item.name }}</div>
    </el-col>
  </el-row>

  <!-- 角色登录 -->
  <div v-else class="is-login">
    <h3>登录中.....(倒计时: {{ countdown }} 秒)</h3>
    <div>角色:
      <span>{{ currentUser }}</span>
    </div>
  </div>
`

```

---

```

    <!-- 非消费者则显示角色地址 -->
    <div v-if="currentUser !== '消费者'">角色地址:
      <span>{{ address }}</span>
    </div>

    <!-- 直接登录按钮 -->
    <el-button type="primary" @click="clearTimer">直接登录</el-button>
  </div>
</div>
`,
methods: {
  // 登录时有个 5 秒的倒计时，这里是在点击直接登录时，清楚倒计时，直接跳到相
  关页面
  clearTimer() {
    clearInterval(this.timer);
    this.$emit('login', {
      component: this.loginItem.component,
      user: this.loginItem.name,
    });
  },
  // 倒计时
  countdownInterval({ component, name: user }) {
    this.timer = setInterval(() => {
      if(this.countdown <= 0){
        this.clearTimer();
      }
      this.countdown -= 1;
    }, 1000);
  },
  // 点击用户登录，获取用户地址
  handleClick(item) {
    this.loginItem = item;
    // 处理消费者角色，其他三个角色都有一个角色地址
    if (item.userName !== 'consumer') {
      axios({
        method: 'get',
        url: `/userinfo?userName=${item.userName}`,
      })
      .then(ret => {
        this.address = ret.data.address;
        this.currentUser = item.name;
        this.countdownInterval(item);
      })
      .catch(err => {

```

```

        console.log(err)
      })
    } else {
      this.currentUser = item.name;
      this.countdownInterval(item);
    }
  }
}
}
}
}

```

参照以上代码，对照下图中的红色框缺失的代码，补全 click 的内容。

```

37 | <!-- 角色选择 -->
38 | <h3 v-if="currentUser === null">请选择您的角色</h3>
39 | <el-row :gutter="80" v-if="currentUser === null">
40 |   <el-col :span="6" v-for="(item, index) in users" :key="index">
41 |     <div @click="">{{ item.name }}</div>
42 |   </el-col>
43 | </el-row>
44 |

```

```

56 |
57 | <!-- 直接登录按钮 -->
58 | <el-button type="" @click="">直接登录</el-button>
59 | </div>
60 | </div>

```

参照以上的完整代码，完善下图中红色框缺失的代码，即处理查询溯源信息的消费者角色的方法。

JS login.js

JS main.js

JS index.js

> style

① README.md

> templates

J application.properties

```

78 |   }, 1000);
79 | },
80 | // 点击用户登录，获取用户地址
81 | handleClick(item) {
82 |   this.loginItem = item;
83 |   // 处理消费者角色，其他三个角色都有一个角色地址
84 |   //
85 | }
86 |
87 | }

```

### (3) 完善 main.js 与 components.js 文件

main.js 和 componets.js 实现了各个页面功能的组件，可以使用搜索功能进行接口关键字搜索。完整代码如下。

// 农场组件

---

```

const Farm = {
  // 接受分页属性，用来控制翻页
  props: ['currPage', 'pageSize'],
  data() {
    return {
      popup: false, // 创建蔬菜弹窗
      foodList: [], // 食品信息
    },
  },
  computed: {
    // 当前页渲染数据
    renderList() {
      return this.foodList.slice((this.currPage - 1) * this.pageSize, this.currPage *
this.pageSize)
    },
  },
  // 注册创建蔬菜子组件
  components: {
    CreateVegetable
  },
  template: `
    <div>
      <!-- 创建蔬菜按钮 -->
      <el-button type="primary" @click="popup = true;">新建蔬菜</el-button>

      <!-- 退出按钮 -->
      <el-button type="danger" @click="$emit('logout', false)" class="button-logout">退
出</el-button>

      <!-- 食品信息列表 -->
      <el-table
        :data="renderList"
        style="width: 100%"
        :default-sort = "{prop: 'date', order: 'descending'}"
      >
        <el-table-column
          prop="traceNumber"
          label="溯源码"
        >
        </el-table-column>
        <el-table-column
          prop="foodName"
          label="食品名称"
        >

```

---

```

        </el-table-column>
        <el-table-column
          prop="traceName"
          label="生产商"
        >
        </el-table-column>
        <el-table-column
          prop="date"
          label="采摘时间"
          sortable
        >
        </el-table-column>
      </el-table>

      <!-- 创建蔬菜子组件 -->
      <CreateVegetable :dialogFormVisible="popup" @popup="handlePopup"
        @confirmPopup="formSubmit" />
    </div>
  `,
  mounted() {
    // 挂载时获取农场数据
    this.getData();
  },
  methods: {
    // 处理数据请求
    getData() {
      axios({
        method: 'get',
        url: `/producing`
      })
        .then(ret => {
          // 对数据进行处理
          this.foodList = ret.data
            .map(item => ({
              ...item,
              date: dateTime(item.timestamp),
              traceNumber: item.traceNumber,
              traceName: item.from,
              foodName: item.name
            }))
            .reverse();
          this.$emit('total', this.foodList.length);
        })
        .catch(err => {

```

---

```
        console.log(err)
    })
},
// 添加蔬菜弹窗
handlePopup(val) {
    this.popup = val;
},
// 添加蔬菜提交
formSubmit(val) {
    axios({
        method: 'post',
        url: '/produce',
        data: {
            ...val,
            quality: parseInt(val.quality),
            user: 0
        }
    })
    .then(ret => {
        if (ret.data.ret !== 1) {
            // 已有溯源码校验
            if (ret.data.ret === 0 && ret.data.msg === 'traceNumber already exist') {
                this.$message({
                    message: '该溯源码已存在，请重新创建',
                    type: 'error',
                    center: true
                });
            } else {
                this.$message({
                    message: '提交失败',
                    type: 'error',
                    center: true
                });
            }
            return
        }
        this.$message({
            message: '提交成功',
            type: 'success',
            center: true
        });
        // 提交成功后重新获取数据
        this.getData();
    })
}
```

---

```
    .catch(err => {
      console.log(err)
    })
    this.popup = false;
  }
}
```

// 中间商组件

```
const Agent = {
  // 和超市的职能差不多，放在 mixin 里面
  mixins: [mixin],
  data() {
    return {
      user: 1, // user: 1 为中间商
    }
  }
}
```

// 超市组件

```
const Mall = {
  // 和中间商的职能差不多，放在 mixin 里面
  mixins: [mixin],
  data() {
    return {
      user: 2, // user: 2 为超市
    }
  }
}
```

// 消费者组件

```
const Consumer = {
  // 溯源详情子组件注册
  components: {
    TraceDetail
  },
  data() {
    return {
      form: {
        traceNumber: "", // 输入的溯源码
      },
      foodDetail: "", // 溯源详细信息
      onSearch: false, // 搜索情况
    }
  }
}
```



```

},
template: `
<el-row style="height: 100%;" class="consumer">
  <!-- 左边为溯源查询 -->
  <el-col :span="7" style="height: 100%;">
    <div class="grid-content bg-purple">
      <el-form :model="form" ref="form">
        <h4>溯源码搜索</h4>
        <el-divider></el-divider>
        <el-form-item
          label="请输入您想要查询的溯源码"
          prop="traceNumber"
        >
          <el-input v-model.number="form.traceNumber" type="textarea" :rows="3"
            autocomplete="off" @clear="onSearch = false;"></el-input>
        </el-form-item>
        <el-button type="primary" @click="onSubmit">查询</el-button>
      </el-form>
    </div>
  </el-col>

  <!-- 右边为溯源查询结果 -->
  <el-col :span="16">
    <div class="grid-content bg-purple-light">
      <div v-if ="!onSearch" class="consumer-tip">请在左侧查询栏中输入溯源码进行
        查询</div>
      <div v-if ="onSearch && foodDetail.length === 0" class="consumer-tip">该溯源码
        无对应信息，请确认后重新查询</div>

      <!-- 溯源查询成功后才展示 -->
      <div v-if ="onSearch && foodDetail.length">
        <!-- 溯源详情组件， user: 3 为消费者 -->
        <TraceDetail :food-detail="foodDetail" :user="3" />
      </div>
    </div>
  </el-col>
  <el-col :span="1">
    <!--退出按钮 -->
    <el-button type="danger" @click="$emit('logout', false)" class="button-logout">
      退出</el-button>
  </el-col>
</el-row>
`
,
methods: {

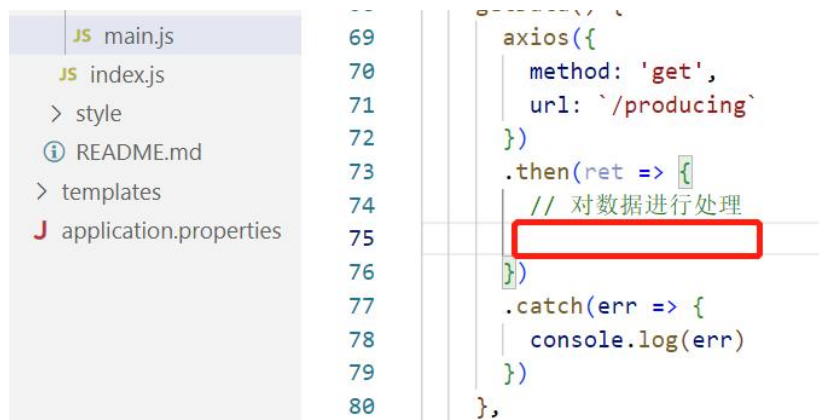
```

```

// 处理按溯源码查询
onSubmit() {
  if (!this.form.traceNumber) {
    this.$message.error('请输入溯源码');
    return;
  }
  this.onSearch = true;
  axios({
    method: 'get',
    url: `/trace?traceNumber=${this.form.traceNumber}`
  })
  .then(ret => {
    this.foodDetail = ret.data || []
  })
  .catch(err => {
    console.log(err);
  })
}
}
}

```

参照以上完整的 `main.js` 代码，补全以下对数据进行处理代码。



```
JS components.js 105
JS login.js 106
JS main.js 107
JS index.js 108
> style 109
① README.md 110
> templates 111
J application.properties 112

113 .then(ret => {
114   this.$message({
115     message: '提交成功',
116     type: 'success',
117     center: true
118   });
119   // 提交成功后重新获取数据
120   this.getData();
121 });
122 .catch(err => {
123   console.log(err)
124 })
125 this.popup = false;
126 }
```

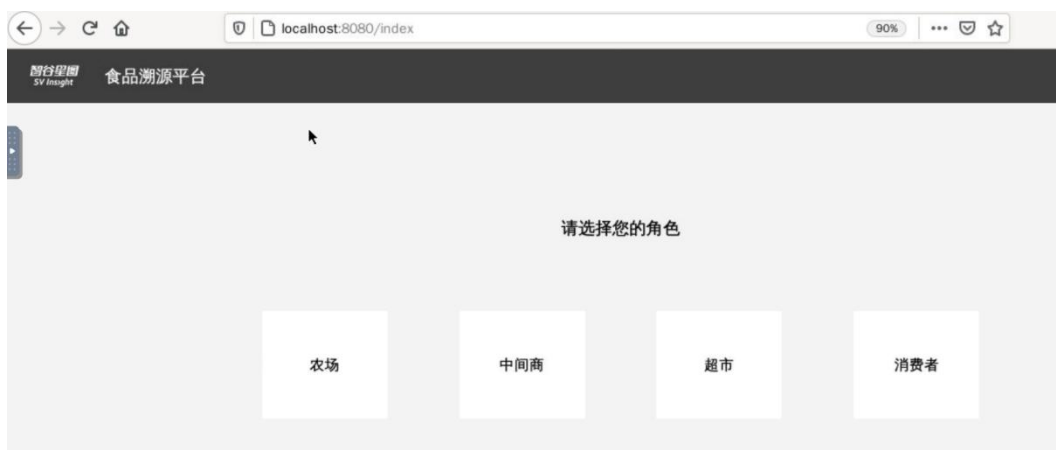
```
JS main.js 213
JS index.js 214
> style 215
① README.md 216
> templates 217
J application.properties 218

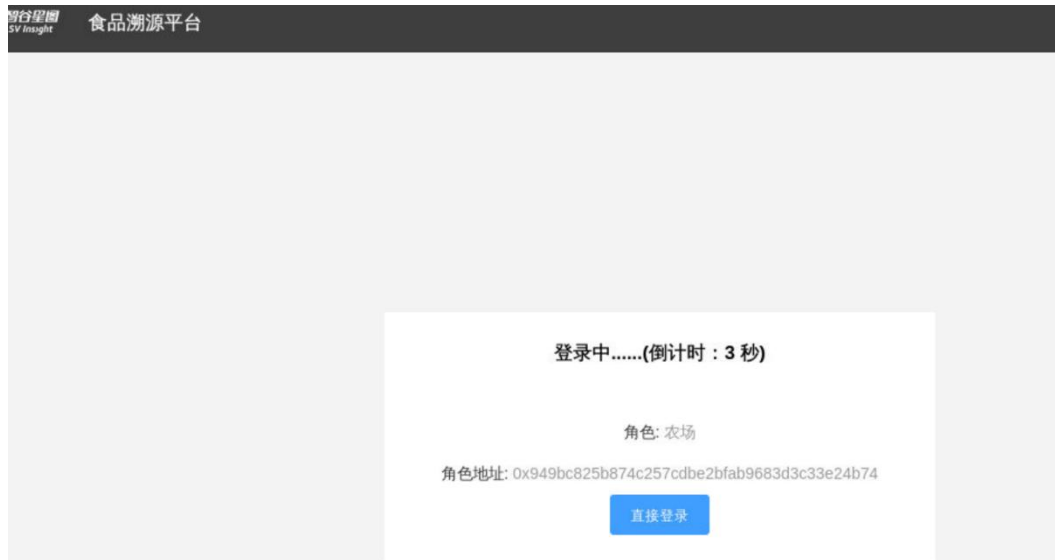
219 methods: {
220   // 处理按溯源码查询
221   onSubmit() {
222   }
223 }
```

完整的 `component.js` 代码留给同学们课后继续探索和研究。

## 6.测试

完成前后端代码后，编译运行程序 run“DemoApplication”，我们打开火狐浏览器输入地址 `localhost:8080/index`。如果能显示如下的界面表示主页加载成功，接下来我们选择一个角色，农场（生产商）登录，会有倒计时显示强制登录。





接下来新建一个蔬菜及相关信息，如下图所示为新建的溯源码，名称，生产商，质量。

### 新建蔬菜

* 溯源码	<input type="text" value="123"/>
* 食品名称	<input type="text" value="tomato"/>
* 生产商	<input type="text" value="ZGXT"/>
* 质检情况	<input checked="" type="radio"/> 优质 <input type="radio"/> 合格 <input type="radio"/> 不合格



之后等一段时间，退出此角色，登录到中间商，如下图所示，输入溯源码添加在中间商处的食品流转信息。

123

添加蔬菜信息

溯源码	食品名称	上架时间
-----	------	------

添加蔬菜信息

\* 质检情况

☐ 优质

☒ 合格

☐ 不合格

\* 收货单位(本单位)

TB

取消

确定

以下是溯源码为123的溯源信息

蔬菜基本信息

农场地址

0x949bc825b874c257cd  
be2bfab9683d3c33e24b7

溯源码

123

食品名称

tomato

生产商

ZGXT

采摘时间

2022-08-17 14:34:22

食品溯源平台

提交成功

123

添加蔬菜信息

共 1 条

10条/页

< 1 >

前往 1 页

接着再等待一段时间，登出此账户，登录到超市，如下图所示继续添加在超市处的食品流转信息。



添加蔬菜信息

\* 质检情况 ☐ 优质 ☒ 合格 ☐ 不合格

\* 收货单位(本单位)

以下是溯源码为123的溯源信息

蔬菜基本信息

农场地址	0x949bc825b874c257cd be2bfab9683d3c33e24b7
溯源码	123
食品名称	tomato
生产商	ZGXT
采摘时间	2022-08-17 14:34:22



最后我们再等待一段时间后登出超市，登录到消费者账户查询完整的溯源信息，如下图所示。



至此，整个测试的流程就全部完成了，尝试添加新的食品信息及流转信息进行溯源验证。