

QuTao ——智能区块链交易平台



1 项目介绍

1.1 项目背景

日常生活中，人们常有进行交易的需求，由此催生了跳蚤市场；在互联网上，虚拟商品的交易也逐渐兴盛。然而，基于安全性等方面的考虑，互联网上的交易往往需要通过管理平台，出现“中间商赚差价”的现象。本项目针对这一痛点，利用区块链去中心化、安全性高的特点，实现了一个智能区块链交易平台，以帮助用户进行高效交易。我们将项目命名为“区淘QuTao”，其中“Qu”代指区块链，“Tao”代指淘宝，顾名思义，展示了应用的底层架构和目标功能。

1.2 项目解决的问题

1. 提供线上交易平台，解决了用户交易虚拟商品的需求。例如，买卖游戏软件、影视资源等等。
2. 通过区块链去中心化的特性，使交易不再依赖中间商，解决了“差价”问题。

3. 通过区块链安全性高的特点，解决了用户对交易信任的需求。

1.3 已有功能

- 注册用户
- 用户登录
- 修改用户信息
- 添加商品
- 修改商品信息
- 购买商品
- 搜索商品
- 查看个人商品

功能呈现请见第四部分 **4 最终成果展示**

2 技术开发方案

2.1 链码层 —— fabric / go-sdk

2.1.1 需求

1. go-sdk作为链码层与后端交互的桥梁。
2. 实现链码层简单接口，并用go-sdk包装接口转发给后端。

2.1.2 实现

2.1.2.1 fabric

(1) 基本结构定义

```
1 // SmartContract provides functions for managing a car
2 type SmartContract struct {
3     contractapi.Contract
4 }
5
6 // User describes basic details of a user
7 type User struct {
8     Id      uint   `json:"id"`
9     Name    string `json:"name"`
10    Password string `json:"password"`
11    Balance  uint   `json:"balance"`
12    Goodslist string `json:"goodslist"`
```

```

13 }
14
15 // Product describes basic details of a product
16 type Product struct {
17     Id          uint   `json:"id"`
18     Url          string `json:"url"`
19     Price        uint   `json:"price"`
20     Name         string `json:"name"`
21     Description  string `json:"description"`
22     Owner        string `json:"owner"`
23     Allowance    uint   `json:"allowance"`
24 }
25
26 // QueryResult structure used for handling result of query
27 type QueryUserResult struct {
28     Key    string `json:"key"`
29     Record *User  `json:"record"`
30 }
31 type QueryProductResult struct {
32     Key    string `json:"key"`
33     Record *Product `json:"record"`
34 }

```

(2) 函数定义

```

1 func (s *SmartContract) CreateUser(ctx
    contractapi.TransactionContextInterface, id uint, name string, password
    string, balance uint) error {}
2
3 func (s *SmartContract) QueryUser(ctx contractapi.TransactionContextInterface,
    name string) (*User, error) {}
4
5 func (s *SmartContract) QueryAllUsers(ctx
    contractapi.TransactionContextInterface) ([]QueryUserResult, error) {}
6
7 func (s *SmartContract) UpdateUser(ctx
    contractapi.TransactionContextInterface, name string, password string, balance
    uint, sel string) error {}
8
9 func (s *SmartContract) UpdateProduct(ctx
    contractapi.TransactionContextInterface, id uint, url string, price uint,
    allowance uint, name string, description string, sel string) error {}
10

```

```

11 func (s *SmartContract) CreateProduct(ctx
    contractapi.TransactionContextInterface, id uint, url string, price uint,
    owner string, allowance uint, name string, description string) error {}
12
13 func (s *SmartContract) QueryProduct(ctx
    contractapi.TransactionContextInterface, id uint) (*Product, error) {}
14
15 func (s *SmartContract) QueryAllProducts(ctx
    contractapi.TransactionContextInterface) ([]QueryProductResult, error) {}
16
17 func (s *SmartContract) BuyProduct(ctx
    contractapi.TransactionContextInterface, buyer string, product_id uint, times
    uint) error {}
18
19 func (s *SmartContract) ClearState(ctx
    contractapi.TransactionContextInterface) error {}
20

```

(3) 链码启动

```

1 func main() {
2
3     chaincode, err := contractapi.NewChaincode(new(SmartContract))
4
5     if err != nil {
6         fmt.Printf("Error create fabcar chaincode: %s", err.Error())
7         return
8     }
9
10    if err := chaincode.Start(); err != nil {
11        fmt.Printf("Error starting fabcar chaincode: %s", err.Error())
12    }
13 }
14

```

2.1.2.2 go-sdk

(1) 结构定义

```

1 // Number of total users
2 var UserNum uint

```

```
3
4 // Number of total products
5 var ProductNum uint
6
7 // User describes basic details of a user
8 type User struct {
9     Id          uint   `json:"id"`
10    Name         string  `json:"name"`
11    Password     string  `json:"password"`
12    Balance      uint    `json:"balance"`
13    Goodslist    string  `json:"goodslist"`
14 }
15
16 // Product describes basic details of a product
17 type Product struct {
18     Id          uint   `json:"id"`
19     Url         string  `json:"url"`
20     Price       uint    `json:"price"`
21     Name        string  `json:"name"`
22     Description string  `json:"description"`
23     Owner       string  `json:"owner"`
24     Allowance   uint    `json:"allowance"`
25 }
26
27 type UpdateUserRequest struct {
28     Name        string  `json:"name"`
29     Password    string  `json:"password"`
30     Balance     uint    `json:"balance"`
31     Select      string  `json:"select"`
32 }
33
34 type UpdateProductRequest struct {
35     Id          uint   `json:"id"`
36     Url         string  `json:"url"`
37     Price       uint    `json:"price"`
38     Allowance   uint    `json:"allowance"`
39     Name        string  `json:"name"`
40     Description string  `json:"description"`
41     Select      string  `json:"select"`
42 }
43
44 //buy product struct
45 type BuyProductRequest struct {
46     Buyer       string  `json:"buyer"`
47     Product_id  uint    `json:"product_id"`
48     Times       uint    `json:"times"`
49 }
```

```

50
51 var (
52     SDK          *fabsdk.FabricSDK
53     channelClient *channel.Client
54     channelName   = "mychannel"
55     chaincodeName = "QuTao"
56     orgName       = "Org1"
57     orgAdmin      = "Admin"
58     org1Peer0     = "peer0.org1.example.com"
59     org2Peer0     = "peer0.org2.example.com"
60 )

```

(2) 接口实现

其中两个接口使用GET请求，其余接口使用POST请求

链码层API及部署方案：[📖 区块链层API LIST及部署方法](#)

```

1 func RunGin() {
2     InitState()
3     r := gin.Default()
4
5     r.GET("/QueryAllUsers", func(c *gin.Context) {
6         var result channel.Response
7         result, err := ChannelExecute("QueryAllUsers", [][]byte{})
8         fmt.Println(result)
9         if err != nil {
10             //fmt.Printf("Failed to evaluate transaction: %s\n", err)
11             c.JSON(http.StatusBadRequest, gin.H{
12                 "code":    "400",
13                 "message": "Failure",
14                 "result":  string(err.Error()),
15             })
16         } else {
17             c.JSON(http.StatusOK, gin.H{
18                 "code":    "200",
19                 "message": "Success",
20                 "result":  string(result.Payload),
21             })
22         }
23     })
24
25     r.POST("/CreateUser", func(c *gin.Context) {
26         var user User
27         c.BindJSON(&user)

```

```

28     var result channel.Response
29     result, err := ChannelExecute("CreateUser", [][]byte{[]byte(strconv.Itoa
30     fmt.Println(result)
31     if err != nil {
32         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
33         c.JSON(http.StatusBadRequest, gin.H{
34             "code":    "400",
35             "message": "Failure",
36             "result":  string(err.Error()),
37         })
38     } else {
39         UserNum++
40         c.JSON(http.StatusOK, gin.H{
41             "code":    "200",
42             "message": "Success",
43             "result":  "{\"id\":\"" + strconv.Itoa(int(UserNum-1)) + "\"}",
44         })
45     }
46 })
47
48 r.POST("/QueryUser", func(c *gin.Context) {
49     var user User
50     c.BindJSON(&user)
51     var result channel.Response
52     result, err := ChannelExecute("QueryUser", [][]byte{[]byte(user.Name)})
53     fmt.Println(result)
54     if err != nil {
55         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
56         c.JSON(http.StatusBadRequest, gin.H{
57             "code":    "400",
58             "message": "Failure",
59             "result":  string(err.Error()),
60         })
61     } else {
62         c.JSON(http.StatusOK, gin.H{
63             "code":    "200",
64             "message": "Success",
65             "result":  string(result.Payload),
66         })
67     }
68 })
69
70 r.POST("/UpdateUser", func(c *gin.Context) {
71     var user UpdateUserRequest
72     c.BindJSON(&user)
73     var result channel.Response
74     result, err := ChannelExecute("UpdateUser", [][]byte{[]byte(user.Name),

```

```

75     fmt.Println(result)
76     if err != nil {
77         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
78         c.JSON(http.StatusBadRequest, gin.H{
79             "code":    "400",
80             "message": "Failure",
81             "result":  string(err.Error()),
82         })
83     } else {
84         c.JSON(http.StatusOK, gin.H{
85             "code":    "200",
86             "message": "Success",
87             "result":  string(result.Payload),
88         })
89     }
90 })
91
92 r.POST("/UpdateProduct", func(c *gin.Context) {
93     var product UpdateProductRequest
94     c.BindJSON(&product)
95     var result channel.Response
96     result, err := ChannelExecute("UpdateProduct", [][]byte{[]byte(strconv.I
97     fmt.Println(result)
98     if err != nil {
99         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
100        c.JSON(http.StatusBadRequest, gin.H{
101            "code":    "400",
102            "message": "Failure",
103            "result":  string(err.Error()),
104        })
105    } else {
106        ProductNum++
107        c.JSON(http.StatusOK, gin.H{
108            "code":    "200",
109            "message": "Success",
110            "result":  string(result.Payload),
111        })
112    }
113 })
114
115 r.GET("/QueryAllProducts", func(c *gin.Context) {
116     var result channel.Response
117     result, err := ChannelExecute("QueryAllProducts", [][]byte{})
118     fmt.Println(result)
119     if err != nil {
120         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
121         c.JSON(http.StatusBadRequest, gin.H{

```



```

122         "code":    "400",
123         "message": "Failure",
124         "result":  string(err.Error()),
125     })
126 } else {
127     c.JSON(http.StatusOK, gin.H{
128         "code":    "200",
129         "message": "Success",
130         "result":  string(result.Payload),
131     })
132 }
133 })
134
135 r.POST("/CreateProduct", func(c *gin.Context) {
136     var product Product
137     c.BindJSON(&product)
138     var result channel.Response
139     result, err := ChannelExecute("CreateProduct", [][]byte{[]byte(strconv.I
140     fmt.Println(result)
141     if err != nil {
142         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
143         c.JSON(http.StatusBadRequest, gin.H{
144             "code":    "400",
145             "message": "Failure",
146             "result":  string(err.Error()),
147         })
148     } else {
149         ProductNum++
150         c.JSON(http.StatusOK, gin.H{
151             "code":    "200",
152             "message": "Success",
153             "result":  "{ \"id\": \" + strconv.Itoa(int(ProductNum-1)) + \"}",
154         })
155     }
156 })
157
158 r.POST("/QueryProduct", func(c *gin.Context) {
159     var product Product
160     c.BindJSON(&product)
161     var result channel.Response
162     result, err := ChannelExecute("QueryProduct", [][]byte{[]byte(strconv.It
163     fmt.Println(result)
164     if err != nil {
165         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
166         c.JSON(http.StatusBadRequest, gin.H{
167             "code":    "400",
168             "message": "Failure",

```

```

169         "result": string(err.Error()),
170     })
171 } else {
172     c.JSON(http.StatusOK, gin.H{
173         "code":    "200",
174         "message": "Success",
175         "result":  string(result.Payload),
176     })
177 }
178 })
179
180 r.POST("/BuyProduct", func(c *gin.Context) {
181     req := BuyProductRequest{}
182     c.BindJSON(&req)
183     var result channel.Response
184     result, err := ChannelExecute("BuyProduct", [][]byte{[]byte(req.Buyer)},
185     fmt.Println(result)
186     if err != nil {
187         //fmt.Printf("Failed to evaluate transaction: %s\n", err)
188         c.JSON(http.StatusBadRequest, gin.H{
189             "code":    "400",
190             "message": "Failure",
191             "result":  string(err.Error()),
192         })
193     } else {
194         c.JSON(http.StatusOK, gin.H{
195             "code":    "200",
196             "message": "Success",
197             "result":  string(result.Payload),
198         })
199     }
200 })
201
202 r.Run(":9099")
203 }

```

2.2 后端层 —— java

2.2.1 需求

1. 将基础业务转发至链码层
2. 处理复杂业务（如登录、搜索），将其拆解为基础业务发送至区块链层
3. 后台信息监控与指令处理

4. 对接前端

2.2.2 技术栈

1. 使用 `CloseableHttpResponse` 连接链码层
2. 使用 `Springboot` 连接前端
3. 使用 `slf4j` 输出日志信息
4. 使用 `md5` 加密用户登录信息

2.2.3 实现

1. 对链码的实体（User和Product）进行封装

User结构如下：

```
1 public class User {
2     private final int id;
3     private String name;
4     private String password;
5     private int balance;
6     //goods的未序列化的版本，用于懒加载
7     private String goodslist;
8     private List<Integer> goods;
9     //details omitted
10 }
```

Product结构如下：

```
1 public class Product {
2     private final int id;
3     private String url;
4     private int price;
5     private String owner;
6     private String name;
7     private String description;
8     private int allowance;
9     //details omitted
10 }
```

2. 与链码对接的接口

```
1 public Result<Integer> createUser(String name, String password, double initialBa
```

```

2 public Result<User> getUser(String name);
3 public Result<List<User>> getUsers();
4 public Result<?> updateUser(User user, String select);
5 public Result<Integer> createProduct(String url, int price, String owner, int al
6 public Result<Product> getProduct(int id);
7 public Result<List<Product>> getProducts();
8 public Result<?> buyProduct(String buyer, int productId, int times);
9 public Result<?> updateProduct(Request.ModifyProduct request , Product product);
10 public record QueryResult(int code, String message, String result){
11     private static final Gson gson = new Gson();
12     public static QueryResult fromJson(String json){
13         return gson.fromJson(json,QueryResult.class);
14     }
15     public <T> T getResult(Class<T> clazz){
16         String json = result.replaceAll("\\\\", "");
17         return gson.fromJson(json,clazz);
18     }
19     public boolean isSuccess(){
20         return "Success".equalsIgnoreCase(message);
21     }
22 }
23 public record QueryAllUsersResult(String key, User record){}
24 public record QueryAllProductsResult(String key, Product record){}
25 public record CreateResult(int id){}

```

3. 与前端对接的接口

RequestBody定义如下：

```

1 public class Request {
2     public record Register(String username, String password){}
3     public record Login(String username, String password){}
4     public record ChangePassword(String username, String oldPassword, String new
5     public record Recharge(String username, int amount){}
6     public record Buy(String username, int productId, int times){}
7     public record CreateProduct(String username, String url, int price, int allo
8     public record ModifyProduct(String username, int productId, @Nullable String
9         @Nullable Integer allowance, @Nullable String na
10     public record ListProduct(String message){}
11     public record ListMyProduct(String username){}
12 }

```

返回值定义如下：

```

1 public record Result<R>(boolean success, R payload, String message) {
2     public static <R> Result<R> of(boolean success, R payload, String message){
3         return new Result<>(success, payload, message);
4     }
5     public static <R> Result<R> of(boolean success, R payload){
6         return of(success, payload, null);
7     }
8     public static <R> Result<R> of(boolean success, String message){
9         return of(success, null, message);
10    }
11    public static <R> Result<R> of(boolean success){
12        return of(success,null, null);
13    }
14    @Override
15    public String toString() {
16        return "Result{" +
17            "success=" + success +
18            ", payload=" + payload +
19            ", message='" + message + '\'' +
20            '}';
21    }
22 }

```

4. 提供的后台命令（尖括号要去掉）

```

1 create-user -username=<用户名> -password=<密码>
2 query-user -username=<用户名>
3 query-all-users
4 create-product -url=<url> -price=<价格> -owner=<用户名> -allowance=<限量> -name=<名称>
5 query-product -id=<商品id>
6 query-all-products
7 buy (或者buy-product) -buyer=<用户名> -id=<商品id> -times=<购买数量>
8 quit

```

2.3 前端层 —— react

2.3.1 需求

1. 展示用户信息，处理用户修改密码、货币兑换
2. 展示商品数据，处理商品购买、添加个人商品
3. 处理登录、注册信息
4. 对接后端

2.3.2 技术栈

1. Ant Design Pro

该项目主要使用 `ProTable` 进行表格的展示

通过不同的参数，`ProTable` 分别用做表单填写，表格展示，表格查询等功能

2. React

React 是一个用于构建用户界面的 JavaScript 库，React 主要用于构建UI。

React 通过组件的方式构建整个页面，通过组件的嵌套，可以构建出复杂的页面。

该项目主要使用 React 处理前端页面的展示逻辑

3. Redux

Redux 是 JavaScript 状态容器，提供可预测化的状态管理。

Redux 可以让应用的状态变化变得可预测，易于调试。

该项目主要使用 Redux 管理前端的状态，提高代码的可靠性和可维护性。

2.3.3 实现

1. API

```
1 declare namespace API {
2
3   interface registerInfo {
4     username: string;
5     password: string;
6   }
7
8   interface changePasswordInfo {
9     username: string;
10    oldPassword: string;
11    newPassword: string;
12  }
13
14  interface loginInfo {
15    username: string;
16    password: string;
17  }
18
19  interface createProductInfo {
20    username: string;
21    url: string;
22    price: number;
23    allowance: number;
```

```
24     name: string;
25     description: string;
26 }
27
28 interface modifyProductInfo {
29     username: string;
30     productId: number;
31     url: string;
32     price: number;
33     allowance: number;
34     name: string;
35     description: string;
36 }
37
38 interface rechargeInfo {
39     username: string;
40     amount: number;
41 }
42 interface listProductInfo {
43     message: string;
44 }
45
46 interface listMyProductInfo {
47     username: string;
48 }
49
50 interface buyProductInfo {
51     username: string;
52     productId: number;
53     times: number;
54 }
55
56 interface productInfo {
57     id: number;
58     url: string;
59     price: number;
60     owner: string;
61     name: string;
62     description: string;
63     allowance: number;
64 }
65 }
```

2. 商品展示

```

1 <ProTable<API.productInfo>
2   headerTitle="商品列表"
3   actionRef={actionRef}
4   rowKey="cardId"
5   search={{ labelWidth: 'auto' }}
6   toolBarRender={() => [
7   ]}
8   request={async (
9     params,
10    sorter,
11    filter,
12  ) => {
13    const { payload, success } = await listProduct({
14      message:params?.description,
15    });
16    return {
17      data: payload || [],
18      success,
19    };
20  }}
21  columns={columns}
22 />

```

3. 登录界面

```

1 <ProConfigProvider hashed={false}>
2   <div style={{}}>
3     <LoginForm
4       logo={<img src={logoimg} />}
5       title="QuTao"
6       subTitle="垃圾区块链平台"
7       actions={
8         <></>
9       }
10      onFinish={async (values) => {
11        await handleSubmit(values as API.loginInfo,setName);
12      }}
13    >
14      <ProFormText
15        name="username"
16        fieldProps={{
17          size: 'large',
18        }}
19        placeholder={'用户名'}
20        rules={[

```



```
21      {
22        required: true,
23        message: '请输入用户名!',
24      },
25    ]}
26  />
27  <ProFormText.Password
28    name="password"
29    fieldProps={{
30      size: 'large',
31    }}
32    placeholder={'密码'}
33    rules={[
34      {
35        required: true,
36        message: '请输入密码! ',
37      },
38    ]}
39  />
40 </LoginForm>
41 <div
42   style={{
43     marginBlockEnd: 24,
44   }}
45 >
46   <a
47     style={{
48       display: 'block',
49       textAlign: 'center',
50     }}
51     onClick={() => {
52       window.location.href = '/register'
53     }}
54   >
55     去注册
56   </a>
57 </div>
58 <div
59   style={{
60     marginBlockEnd: 24,
61   }}
62 >
63   <a
64     style={{
65       display: 'block',
66       textAlign: 'center',
67     }}
```

```
68     onClick={() => {
69       window.location.href = '/product'
70     }}
71   >
72     游客模式查看商品
73   </a>
74 </div>
75 </div>
76 </ProConfigProvider>
```

4. 对接后端

```
1 export async function register(body: API.registerInfo) {
2   console.log("注册",body)
3   return request<any>('/api/register', {
4     method: 'POST',
5     headers: {
6       'Content-Type': 'application/json',
7     },
8     data: body,
9   });
10 }
11
12 export async function login(body: API.loginInfo) {
13   console.log("登录",body)
14   return request<any>('/api/login', {
15     method: 'POST',
16     headers: {
17       'Content-Type': 'application/json',
18     },
19     data: body,
20   });
21 }
22
23 export async function logout() {
24   console.log("登出")
25   message.success('退出成功')
26 }
27
28 export async function changePassword(body: API.changePasswordInfo) {
29   console.log("修改密码",body)
30   return request<any>('/api/changePassword', {
31     method: 'POST',
32     headers: {
33       'Content-Type': 'application/json',
```

```
34     },
35     data: body,
36   });
37 }
38
39 export async function createProduct(body: API.createProductInfo) {
40   console.log("创建商品",body)
41   return request<any>('/api/createProduct', {
42     method: 'POST',
43     headers: {
44       'Content-Type': 'application/json',
45     },
46     data: body,
47   });
48 }
49
50 export async function modifyProduct(body: API.modifyProductInfo) {
51   console.log("修改商品",body)
52   return request<any>('/api/modifyProduct', {
53     method: 'POST',
54     headers: {
55       'Content-Type': 'application/json',
56     },
57     data: body,
58   });
59 }
60
61 export async function recharge(body: API.rechargeInfo) {
62   console.log("充值",body)
63   return request<any>('/api/recharge', {
64     method: 'POST',
65     headers: {
66       'Content-Type': 'application/json',
67     },
68     data: body,
69   });
70 }
71
72 export async function listProduct(body: API.listProductInfo) {
73   console.log("商品列表",body)
74   return request<any>('/api/listProduct', {
75     method: 'POST',
76     headers: {
77       'Content-Type': 'application/json',
78     },
79     data: body,
80   });
```

```
81 }
82
83 export async function buyProduct(body: API.buyProductInfo) {
84   console.log("购买商品",body)
85   return request<any>('/api/buyProduct', {
86     method: 'POST',
87     headers: {
88       'Content-Type': 'application/json',
89     },
90     data: body,
91   });
92 }
93
94 export async function listMyProduct(body: API.listMyProductInfo) {
95   console.log("我的商品",body)
96   return request<any>('/api/listMyProduct', {
97     method: 'POST',
98     headers: {
99       'Content-Type': 'application/json',
100     },
101     data: body,
102   });
103 }
```

3 团队组成与分工

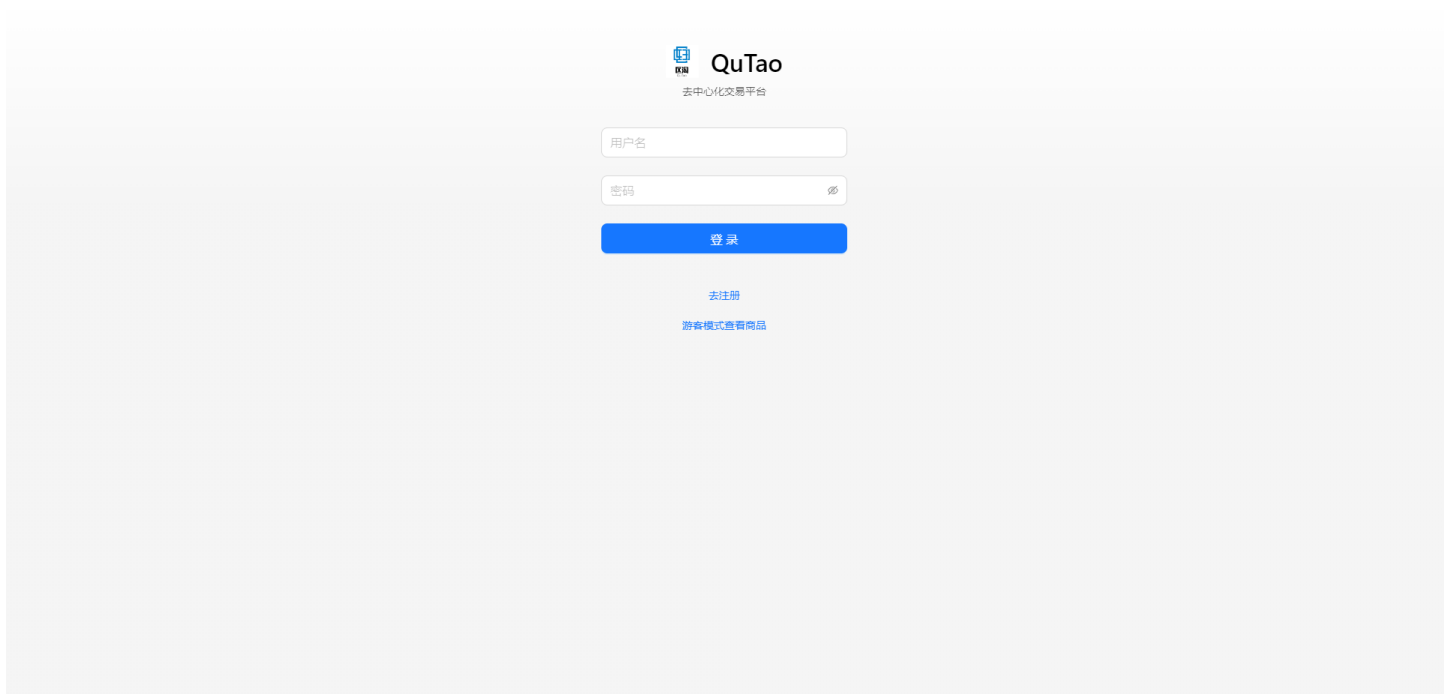
- 卢峰杰：链码层的实现、书写部分报告
- 许若一：后端的实现、书写部分报告
- 邓铭辉：前端的实现、书写部分报告
- 陶天骋：制作PPT及展示、书写部分报告

4 最终成果展示

4.1 首页

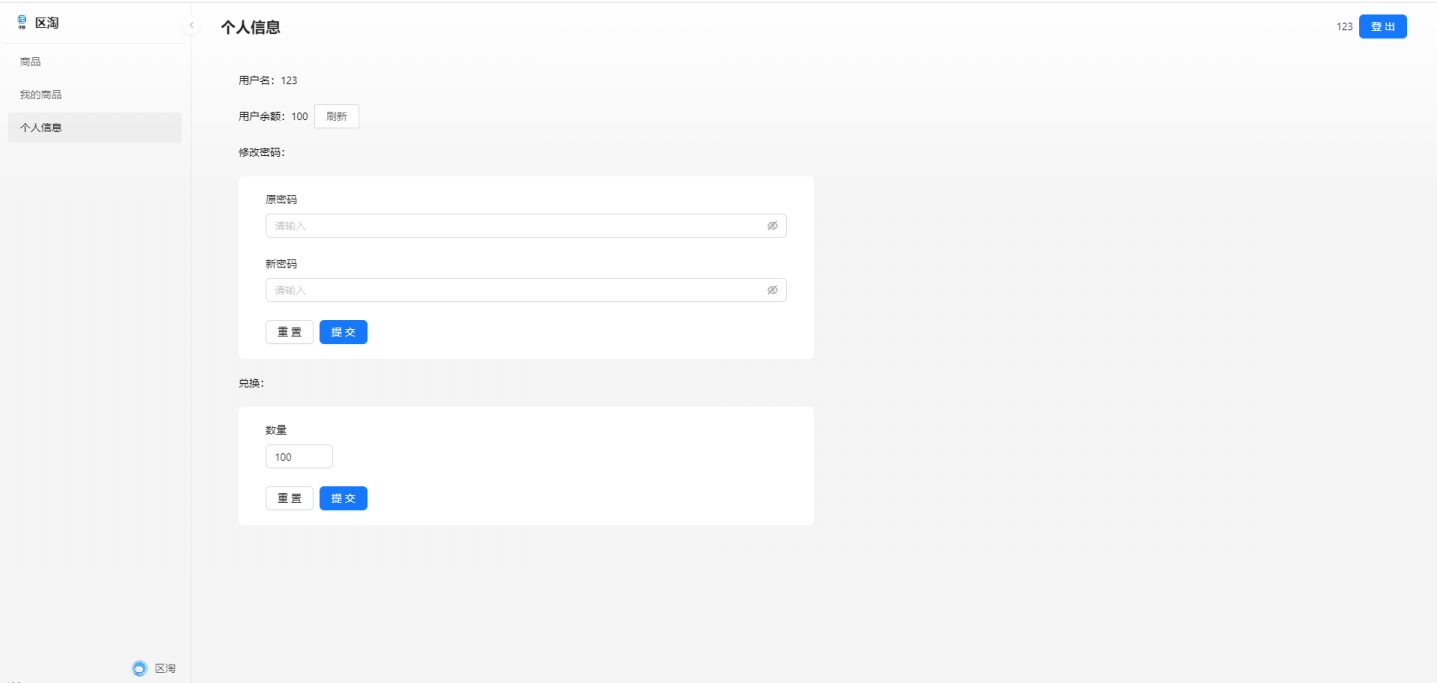


4.2 登录/注册



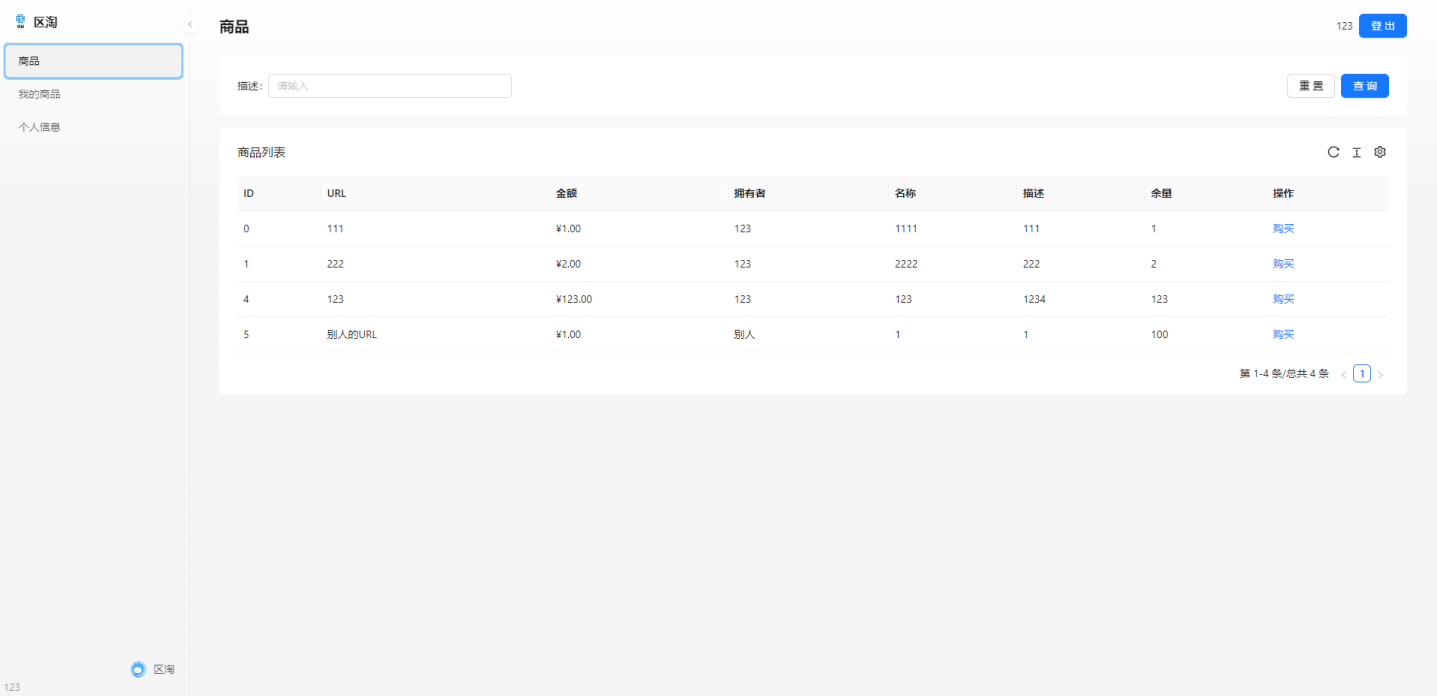
4.3 修改个人信息

可以修改自己的个人信息



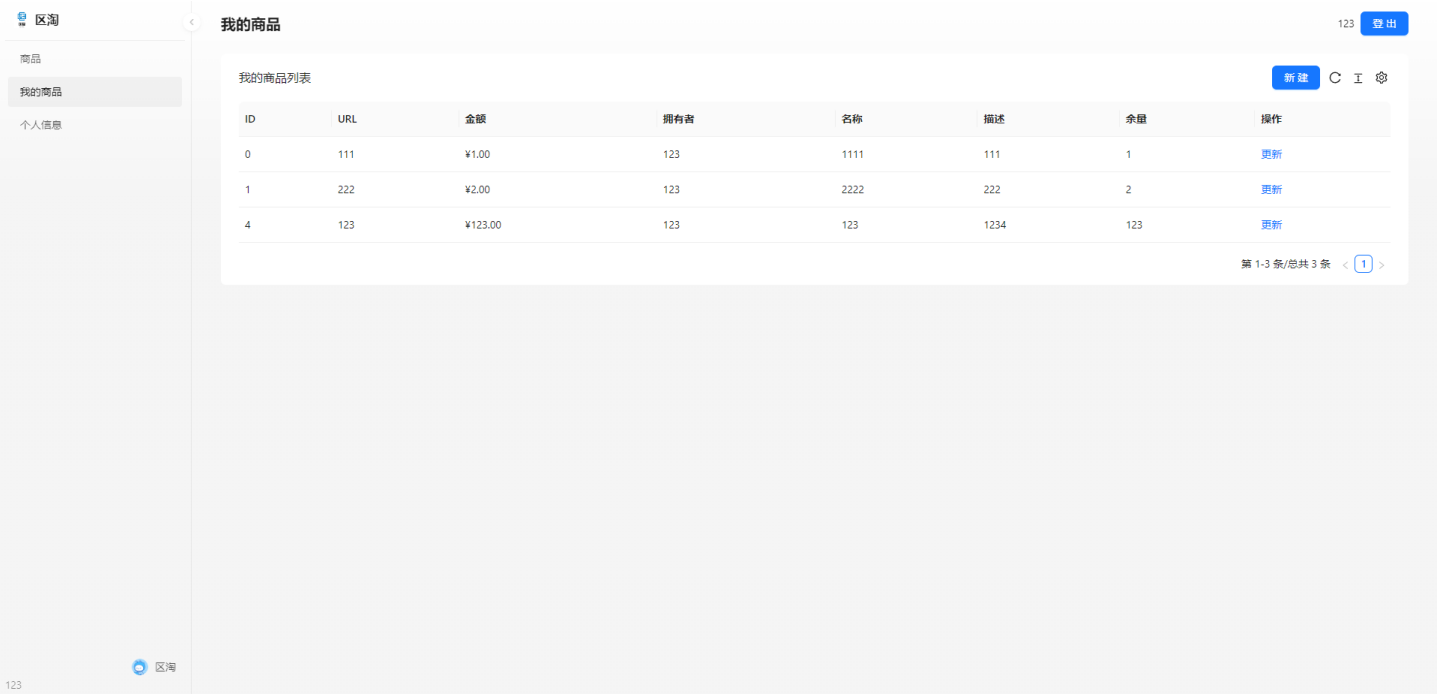
4.4 所有商品列表

查看所有平台上的商品



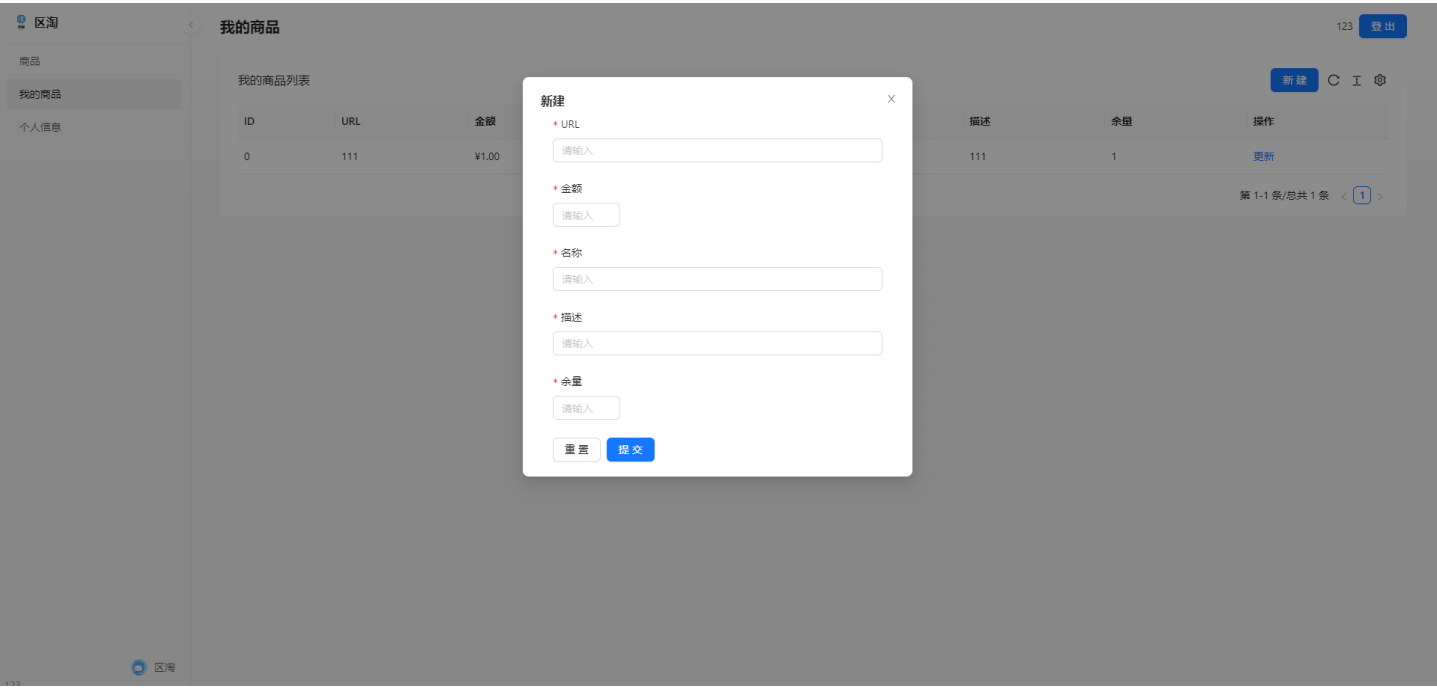
4.5 我的商品

查看自己所有已添加的商品

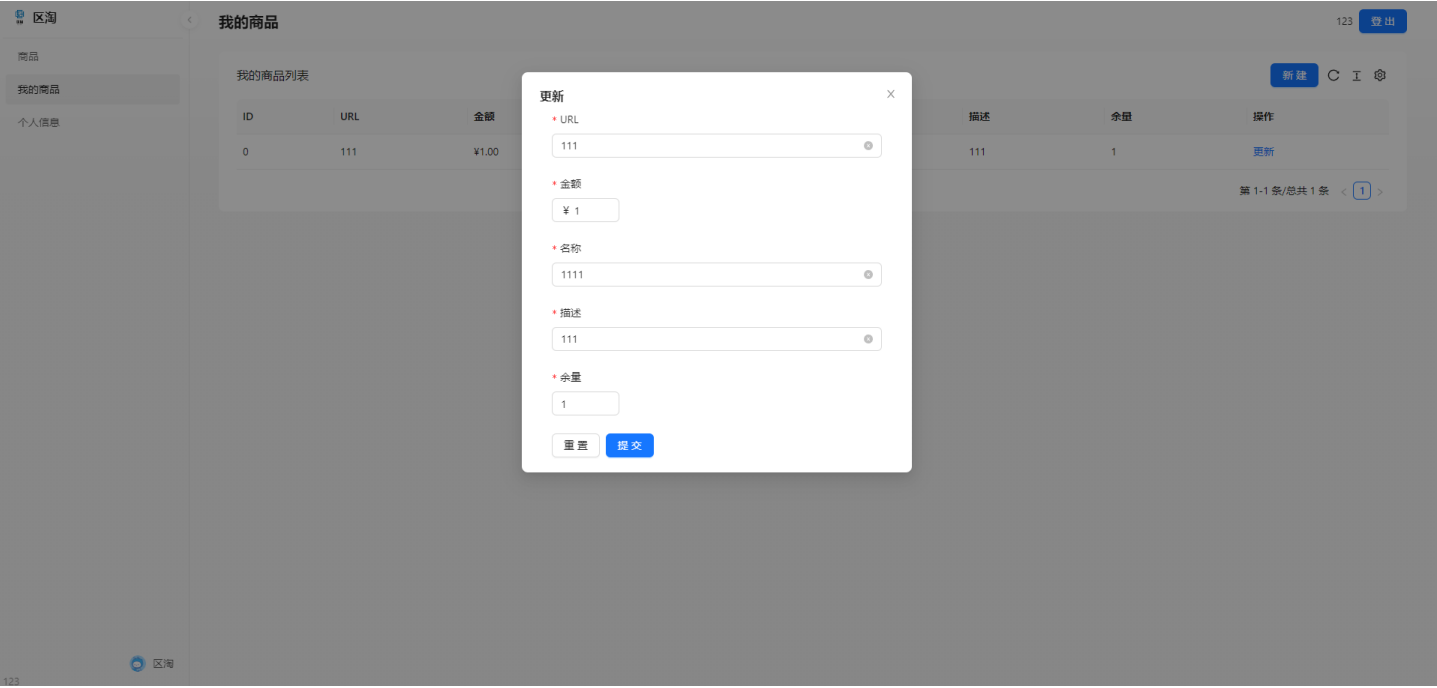


4.6 添加商品

提供商品的相关信息，然后即可添加成功



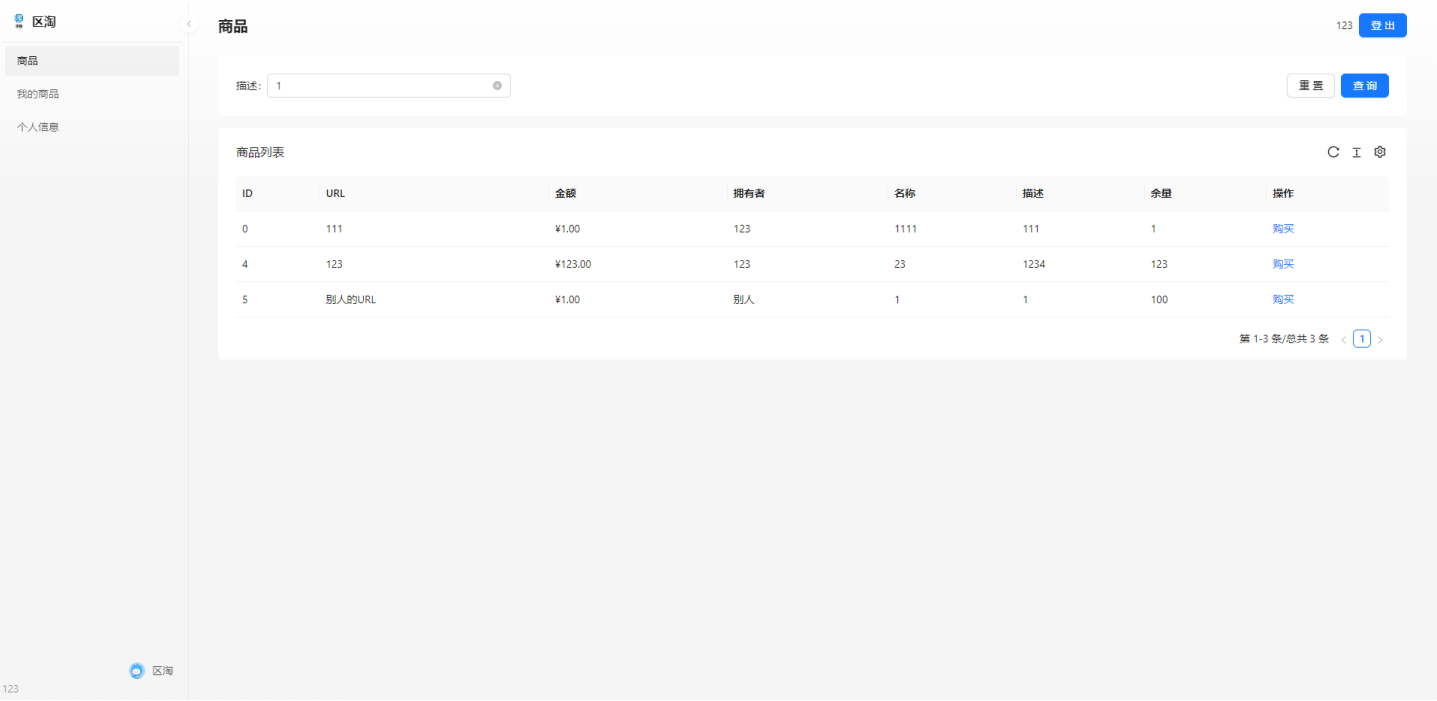
4.7 修改商品信息



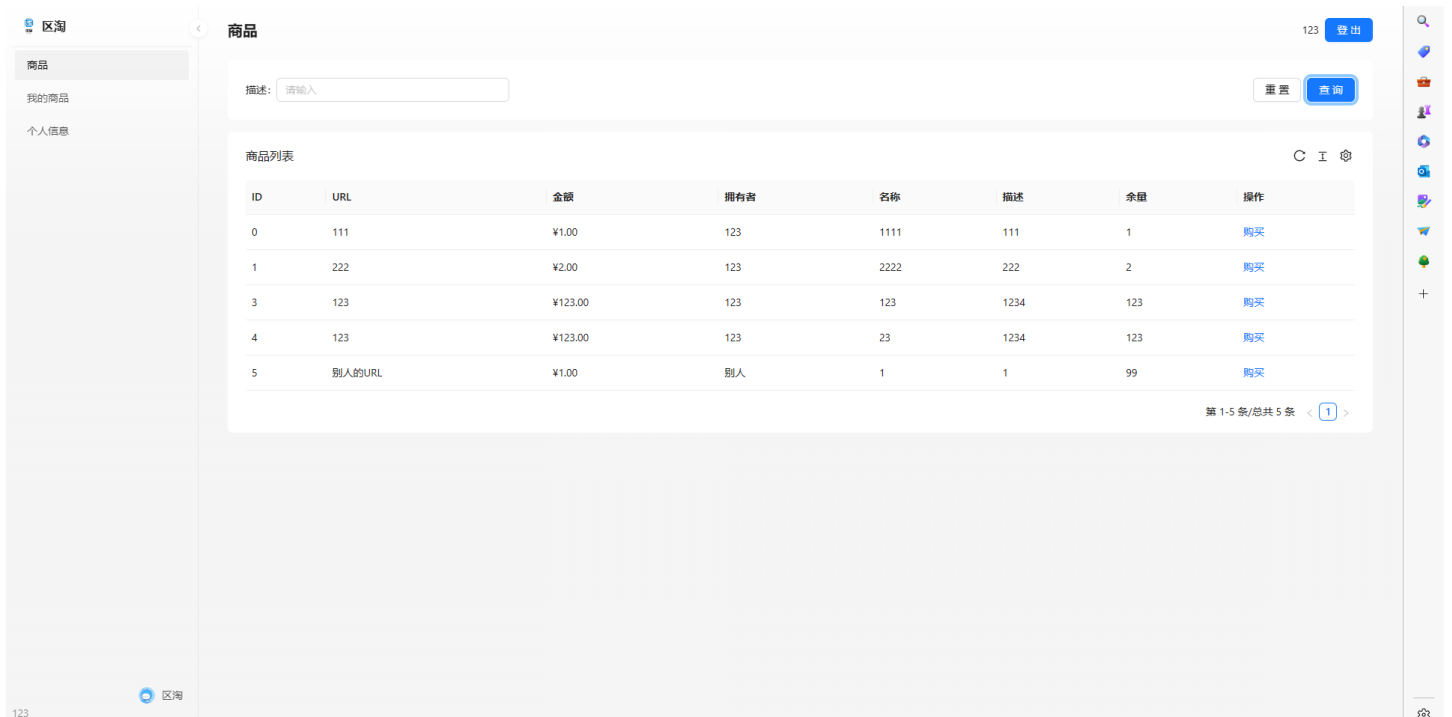
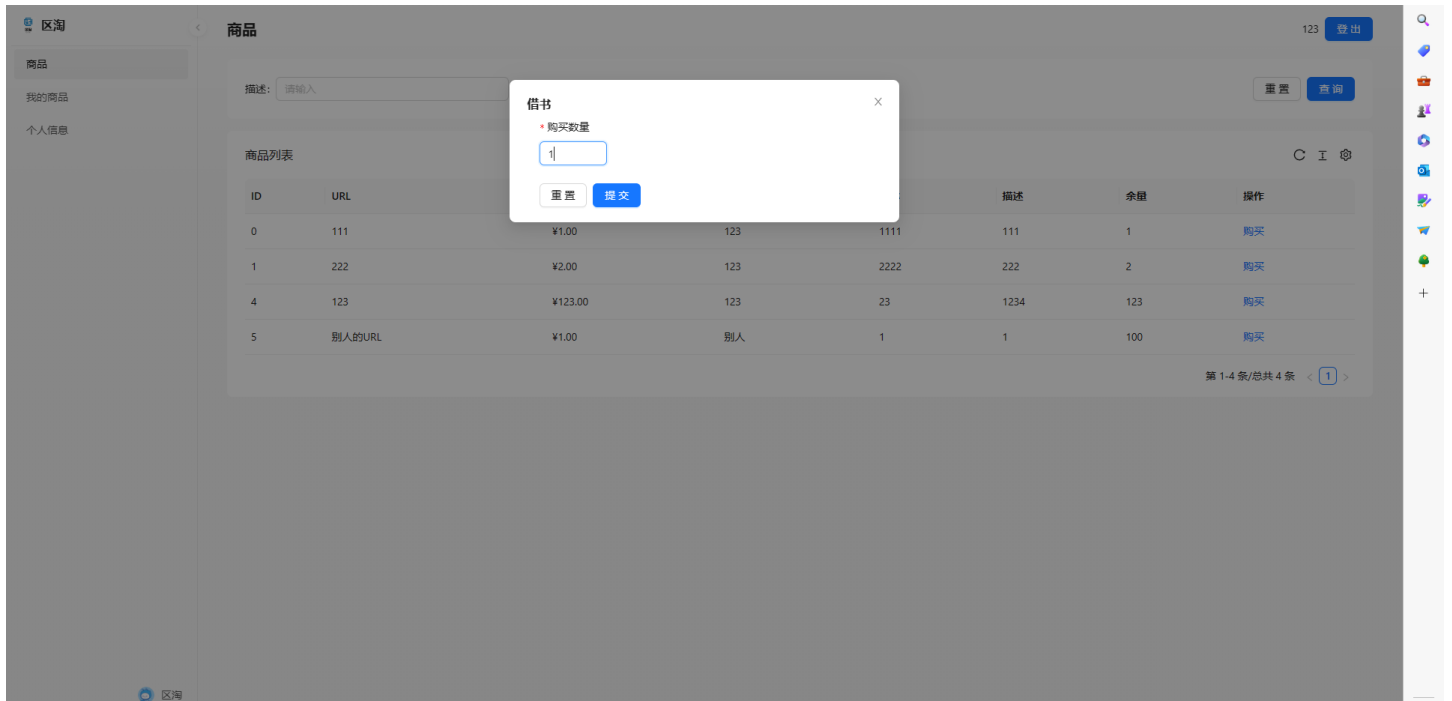
4.8 搜索商品

可以根据描述进行搜索

输入描述1，我们匹配到了部分商品符合条件



4.9 购买商品



5 后期改进思路

1. 更高的安全性：商品信息从原来的一层变成两层，一些属性可以在购买后才能查看
2. 更丰富的功能：消费记录查询功能的实现、退款功能的实现
3. 更优的并发：优化并发性能，支持更高的并发数
4. 更好的部署策略：我们用免费的内网穿透套餐尝试了部署，可以考虑之后部署在自己的服务器上

6 仓库地址



<https://github.com/Crer-lu/QuTao>

欢迎大家给我们一个Star，谢谢！