# 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**

```go
// SmartContract provides functions for managing a car
type SmartContract struct {
    contractapi.Contract
}

// User describes basic details of a user
type User struct {
    Id        uint   `json:"id"`
    Name      string `json:"name"`
    Password  string `json:"password"`
    Balance   uint   `json:"balance"`
    Goodslist string `json:"goodslist"`
```

```
}

// Product describes basic details of a product
type Product struct {
    Id          uint   `json:"id"`
    Url         string `json:"url"`
    Price       uint   `json:"price"`
    Name        string `json:"name"`
    Description string `json:"description"`
    Owner       string `json:"owner"`
    Allowance   uint   `json:"allowance"`
}

// QueryResult structure used for handling result of query
type QueryUserResult struct {
    Key    string `json:"key"`
    Record *User  `json:"record"`
}
type QueryProductResult struct {
    Key    string   `json:"key"`
    Record *Product `json:"record"`
}
```

**（2）函数定义**

```
func (s *SmartContract) CreateUser(ctx contractapi.TransactionContextInterface, id uint, name string, password string, balance uint) error {}

func (s *SmartContract) QueryUser(ctx contractapi.TransactionContextInterface, name string) (*User, error) {}

func (s *SmartContract) QueryAllUsers(ctx contractapi.TransactionContextInterface) ([]QueryUserResult, error) {}

func (s *SmartContract) UpdateUser(ctx contractapi.TransactionContextInterface, name string, password string, balance uint, sel string) error {}

func (s *SmartContract) UpdateProduct(ctx contractapi.TransactionContextInterface, id uint, url string, price uint, allowance uint, name string, description string, sel string) error {}

```

```
func (s *SmartContract) CreateProduct(ctx contractapi.TransactionContextInterface, id uint, url string, price uint, owner string, allowance uint, name string, description string) error {}

func (s *SmartContract) QueryProduct(ctx contractapi.TransactionContextInterface, id uint) (*Product, error) {}

func (s *SmartContract) QueryAllProducts(ctx contractapi.TransactionContextInterface) ([]QueryProductResult, error) {}

func (s *SmartContract) BuyProduct(ctx contractapi.TransactionContextInterface, buyer string, product_id uint, times uint) error {}

func (s *SmartContract) ClearState(ctx contractapi.TransactionContextInterface) error {}

```

**（3）链码启动**

```
func main() {

    chaincode, err := contractapi.NewChaincode(new(SmartContract))

    if err != nil {
        fmt.Printf("Error create fabcar chaincode: %s", err.Error())
        return
    }

    if err := chaincode.Start(); err != nil {
        fmt.Printf("Error starting fabcar chaincode: %s", err.Error())
    }
}

```

### 2.1.2.2 go-sdk

**（1）结构定义**

```
// Number of total users
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 }
```

```
var (
    SDK           *fabsdk.FabricSDK
    channelClient *channel.Client
    channelName   = "mychannel"
    chaincodeName = "QuTao"
    orgName       = "Org1"
    orgAdmin      = "Admin"
    org1Peer0     = "peer0.org1.example.com"
    org2Peer0     = "peer0.org2.example.com"
)
```

### （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

```
func RunGin() {
    InitState()
    r := gin.Default()

    r.GET("/QueryAllUsers", func(c *gin.Context) {
        var result channel.Response
        result, err := ChannelExecute("QueryAllUsers", [][]byte{})
        fmt.Println(result)
        if err != nil {
            //fmt.Printf("Failed to evaluate transaction: %s\n", err)
            c.JSON(http.StatusBadRequest, gin.H{
                "code":    "400",
                "message": "Failure",
                "result":  string(err.Error()),
            })
        } else {
            c.JSON(http.StatusOK, gin.H{
                "code":    "200",
                "message": "Success",
                "result":  string(result.Payload),
            })
        }
    })

    r.POST("/CreateUser", func(c *gin.Context) {
        var user User
        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：

```
public class User {
    private final int id;
    private String name;
    private String password;
    private int balance;
    //goods的未序列化的版本，用于懒加载
    private String goodslist;
    private List<Integer> goods;
    //details omitted
}
```

Product结构如下：

```
public class Product {
    private final int id;
    private String url;
    private int price;
    private String owner;
    private String name;
    private String description;
    private int allowance;
    //details omitted
}
```

2. 与链码对接的接口

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

```
public Result<User> getUser(String name);
public Result<List<User>> getUsers();
public Result<?> updateUser(User user, String select);
public Result<Integer> createProduct(String url, int price, String owner, int al
public Result<Product> getProduct(int id);
public Result<List<Product>> getProducts();
public Result<?> buyProduct(String buyer, int productId, int times);
public Result<?> updateProduct(Request.ModifyProduct request , Product product);
public record QueryResult(int code, String message, String result){
    private static final Gson gson = new Gson();
    public static QueryResult fromJson(String json){
        return gson.fromJson(json,QueryResult.class);
    }
    public <T> T getResult(Class<T> clazz){
        String json = result.replaceAll("\\\\","");
        return gson.fromJson(json,clazz);
    }
    public boolean isSuccess(){
        return "Success".equalsIgnoreCase(message);
    }
}
public record QueryAllUsersResult(String key, User record){}
public record QueryAllProductsResult(String key, Product record){}
public record CreateResult(int id){}
```

3. 与前端对接的接口

RequestBody定义如下：

```
public class Request {
    public record Register(String username, String password){}
    public record Login(String username, String password){}
    public record ChangePassword(String username, String oldPassword, String new
    public record Recharge(String username, int amount){}
    public record Buy(String username, int productId, int times){}
    public record CreateProduct(String username, String url, int price, int allo
    public record ModifyProduct(String username, int productId, @Nullable String
                                @Nullable Integer allowance, @Nullable String na
    public record ListProduct(String message){}
    public record ListMyProduct(String username){}
}
```

返回值定义如下：

```java
public record Result<R>(boolean success, R payload, String message) {
    public static <R> Result<R> of(boolean success, R payload, String message){
        return new Result<>(success, payload, message);
    }
    public static <R> Result<R> of(boolean success, R payload){
        return of(success, payload, null);
    }
    public static <R> Result<R> of(boolean success, String message){
        return of(success, null, message);
    }
    public static <R> Result<R> of(boolean success){
        return of(success,null, null);
    }
    @Override
    public String toString() {
        return "Result{" +
                "success=" + success +
                ", payload=" + payload +
                ", message='" + message + '\'' +
                '}';
    }
}
```

4. 提供的后台命令（尖括号要去掉）

```
create-user -username=<用户名> -password=<密码>
query-user -username=<用户名>
query-all-users
create-product -url=<url> -price=<价格> -owner=<用户名> -allowance=<限量> -name=<
query-product -id=<商品id>
query-all-products
buy（或者buy-product） -buyer=<用户名> -id=<商品id> -times=<购买数量>
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

```
declare namespace API {

  interface registerInfo {
    username: string;
    password: string;
  }

  interface changePasswordInfo {
    username: string;
    oldPassword: string;
    newPassword: string;
  }

  interface loginInfo {
    username: string;
    password: string;
  }

  interface createProductInfo {
    username: string;
    url: string;
    price: number;
    allowance: number;
```

```
    name: string;
    description: string;
  }

  interface modifyProductInfo {
    username: string;
    productId: number;
    url: string;
    price: number;
    allowance: number;
    name: string;
    description: string;
  }

  interface rechargeInfo {
    username: string;
    amount: number;
  }
  interface listProductInfo {
    message: string;
  }

  interface listMyProductInfo {
    username: string;
  }

  interface buyProductInfo {
    username: string;
    productId: number;
    times: number;
  }

  interface productInfo {
    id: number;
    url: string;
    price: number;
    owner: string;
    name: string;
    description: string;
    allowance: number;
  }
}
```

2. 商品展示

```
<ProTable<API.productInfo>
    headerTitle="商品列表"
    actionRef={actionRef}
    rowKey="cardId"
    search={{ labelWidth: 'auto' }}
    toolBarRender={() => [
    ]}
    request={async (
      params,
      sorter,
      filter,
    ) => {
      const { payload, success } = await listProduct({
        message:params?.description,
      });
      return {
        data: payload || [],
        success,
      };
    }}
    columns={columns}
  />
```

3. 登录界面

```
<ProConfigProvider hashed={false}>
  <div style={{}}>
    <LoginForm
      logo={<img src={logoimg} />}
      title="QuTao"
      subTitle="垃圾区块链平台"
      actions={
        <></>
      }
      onFinish={async (values) => {
        await handleSubmit(values as API.loginInfo,setName);
      }}
    >
      <ProFormText
        name="username"
        fieldProps={{
          size: 'large',
        }}
        placeholder={'用户名'}
        rules={[
```

```
          {
            required: true,
            message: '请输入用户名!',
          },
        ]}
      />
      <ProFormText.Password
        name="password"
        fieldProps={{
          size: 'large',
        }}
        placeholder={'密码'}
        rules={[
          {
            required: true,
            message: '请输入密码! ',
          },
        ]}
      />
    </LoginForm>
    <div
      style={{
        marginBlockEnd: 24,
      }}
    >
      <a
        style={{
          display: 'block',
          textAlign: 'center',
        }}
        onClick={() => {
          window.location.href = '/register'
        }}
      >
        去注册
      </a>
    </div>
    <div
      style={{
        marginBlockEnd: 24,
      }}
    >
      <a
        style={{
          display: 'block',
          textAlign: 'center',
        }}
```

```
        onClick={() => {
          window.location.href = '/product'
        }}
      >
        游客模式查看商品
      </a>
    </div>
  </div>
</ProConfigProvider>
```

4. 对接后端

```
export async function register(body: API.registerInfo) {
  console.log("注册",body)
  return request<any>('/api/register', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function login(body: API.loginInfo) {
  console.log("登录",body)
  return request<any>('/api/login', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function logout() {
  console.log("登出")
  message.success('退出成功')
}

export async function changePassword(body: API.changePasswordInfo) {
  console.log("修改密码",body)
  return request<any>('/api/changePassword', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
```

```
    },
    data: body,
  });
}

export async function createProduct(body: API.createProductInfo) {
  console.log("创建商品",body)
  return request<any>('/api/createProduct', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function modifyProduct(body: API.modifyProductInfo) {
  console.log("修改商品",body)
  return request<any>('/api/modifyProduct', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function recharge(body: API.rechargeInfo) {
  console.log("充值",body)
  return request<any>('/api/recharge', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function listProduct(body: API.listProductInfo) {
  console.log("商品列表",body)
  return request<any>('/api/listProduct', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
```

```ts
}

export async function buyProduct(body: API.buyProductInfo) {
  console.log("购买商品",body)
  return request<any>('/api/buyProduct', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}

export async function listMyProduct(body: API.listMyProductInfo) {
  console.log("我的商品",body)
  return request<any>('/api/listMyProduct', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    data: body,
  });
}
```

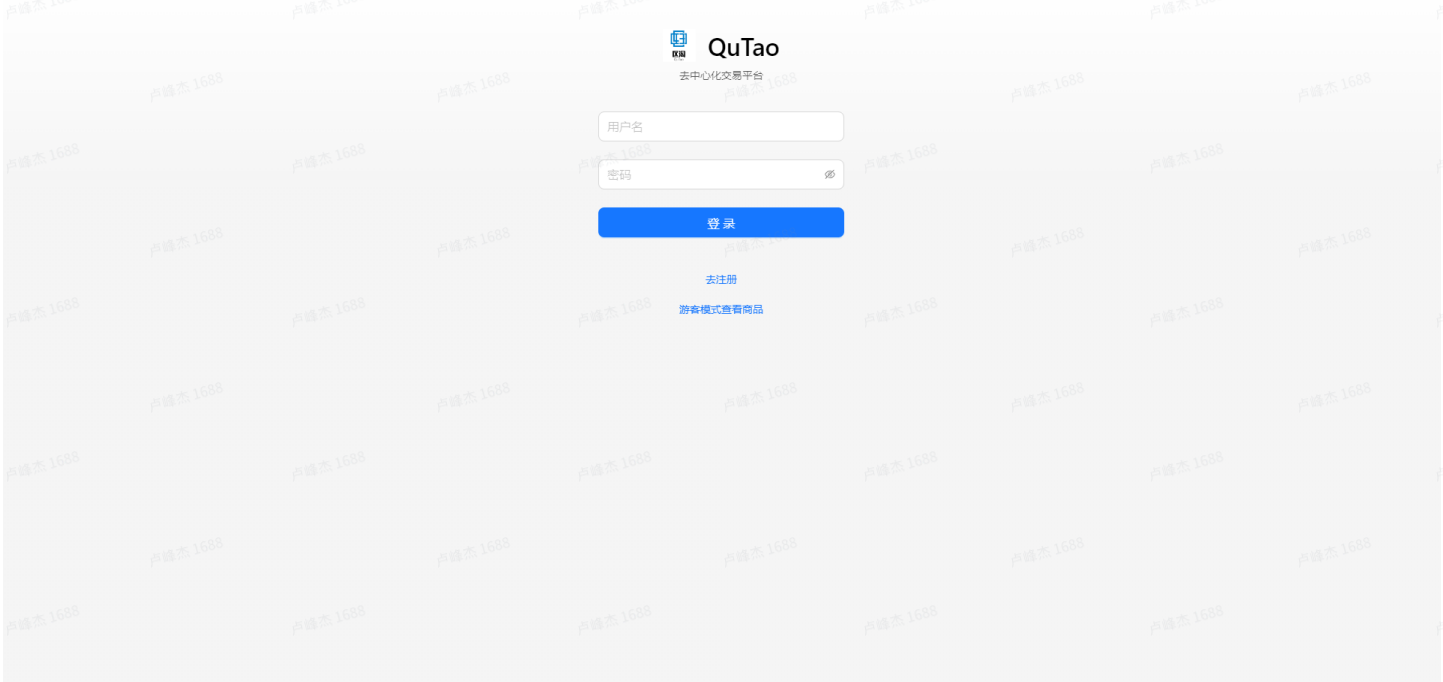
## 3 团队组成与分工

- 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

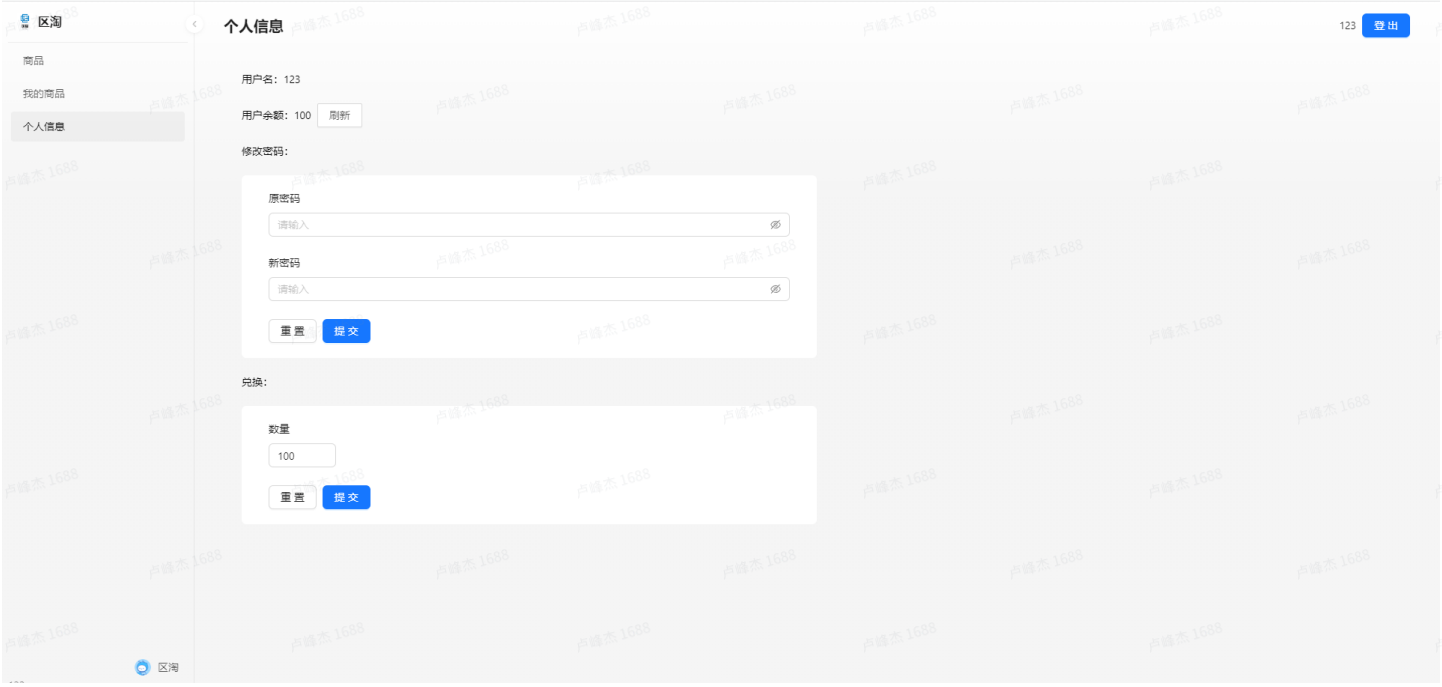
### 4.1 首页


## 4.2 登录/注册



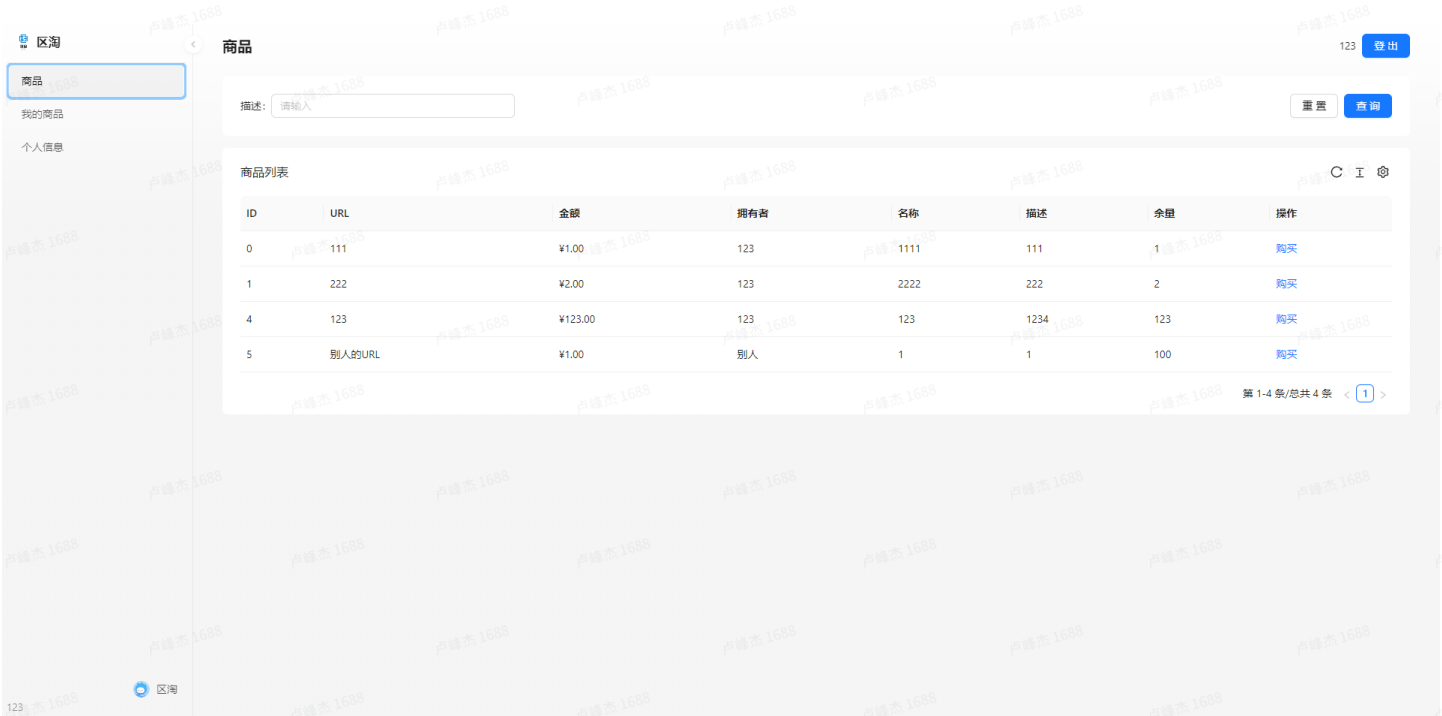

## 4.3 修改个人信息

可以修改自己的个人信息



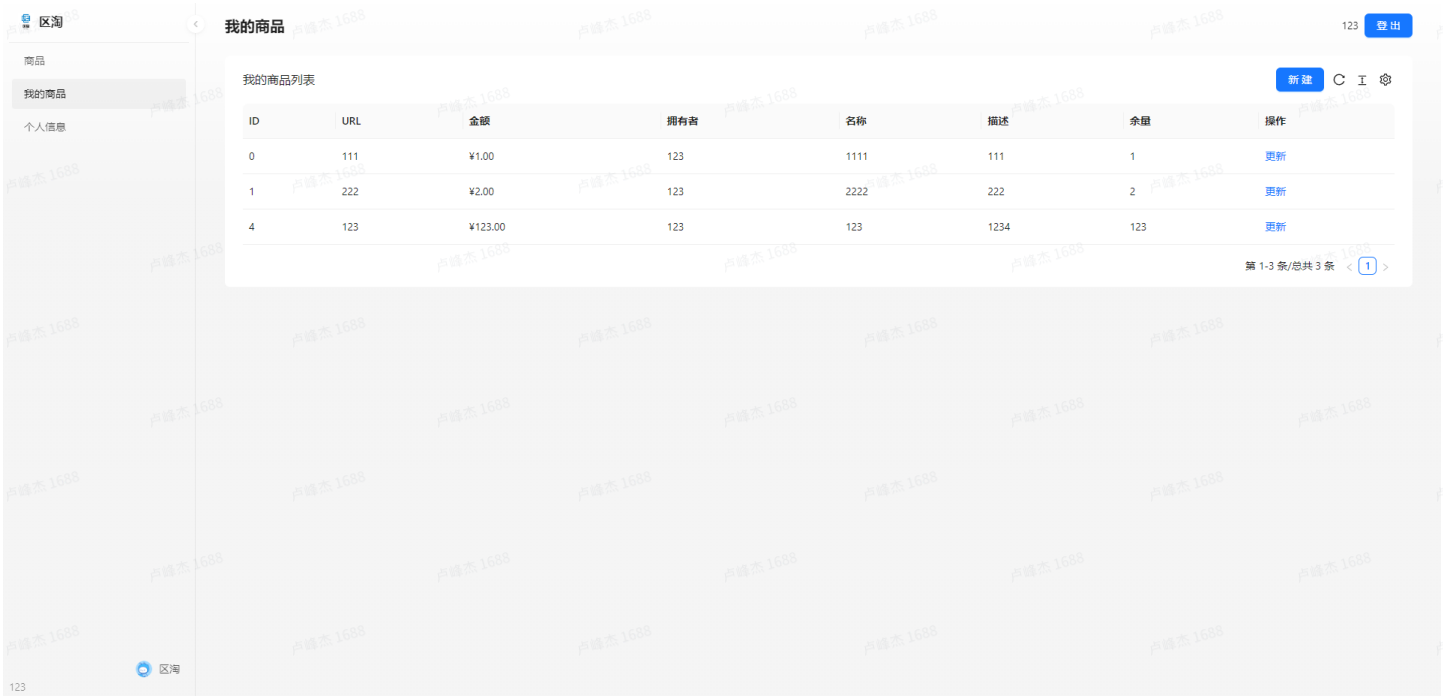

## 4.4 所有商品列表

查看所有平台上的商品



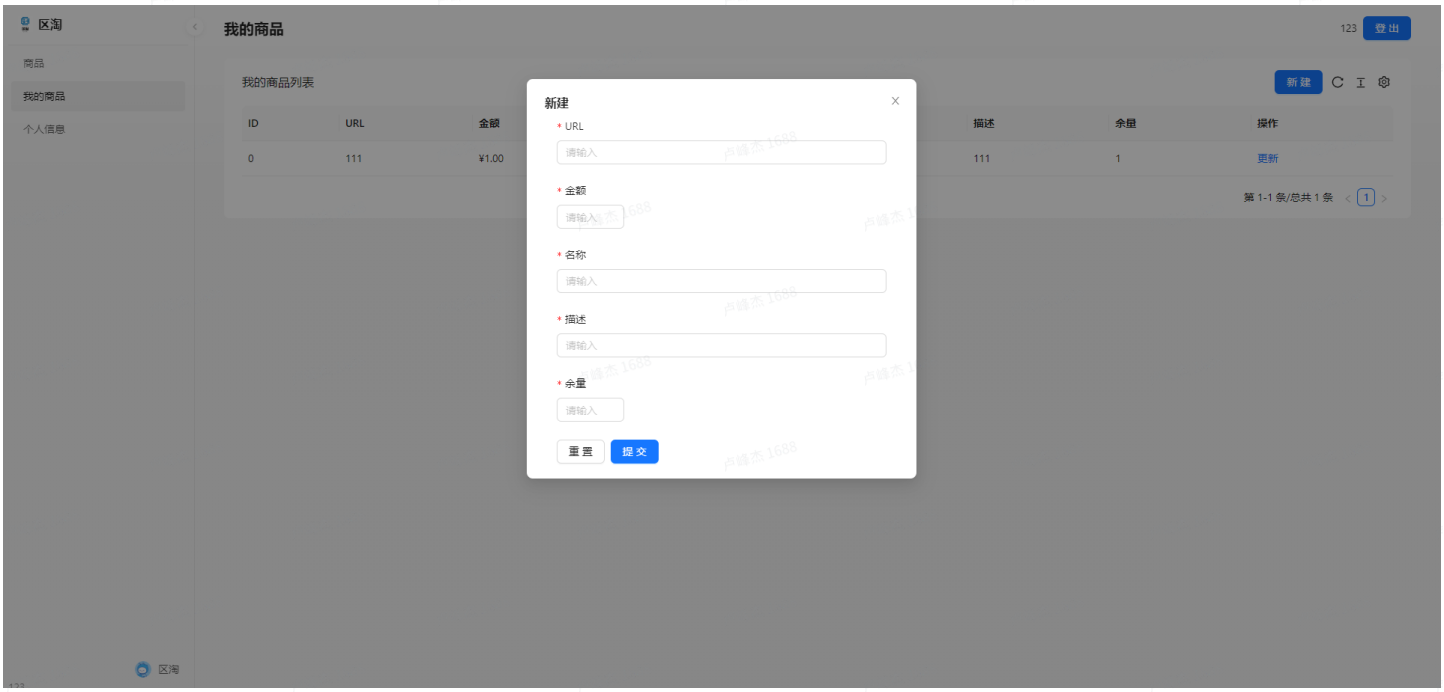

## 4.5 我的商品

查看自己所有已添加的商品

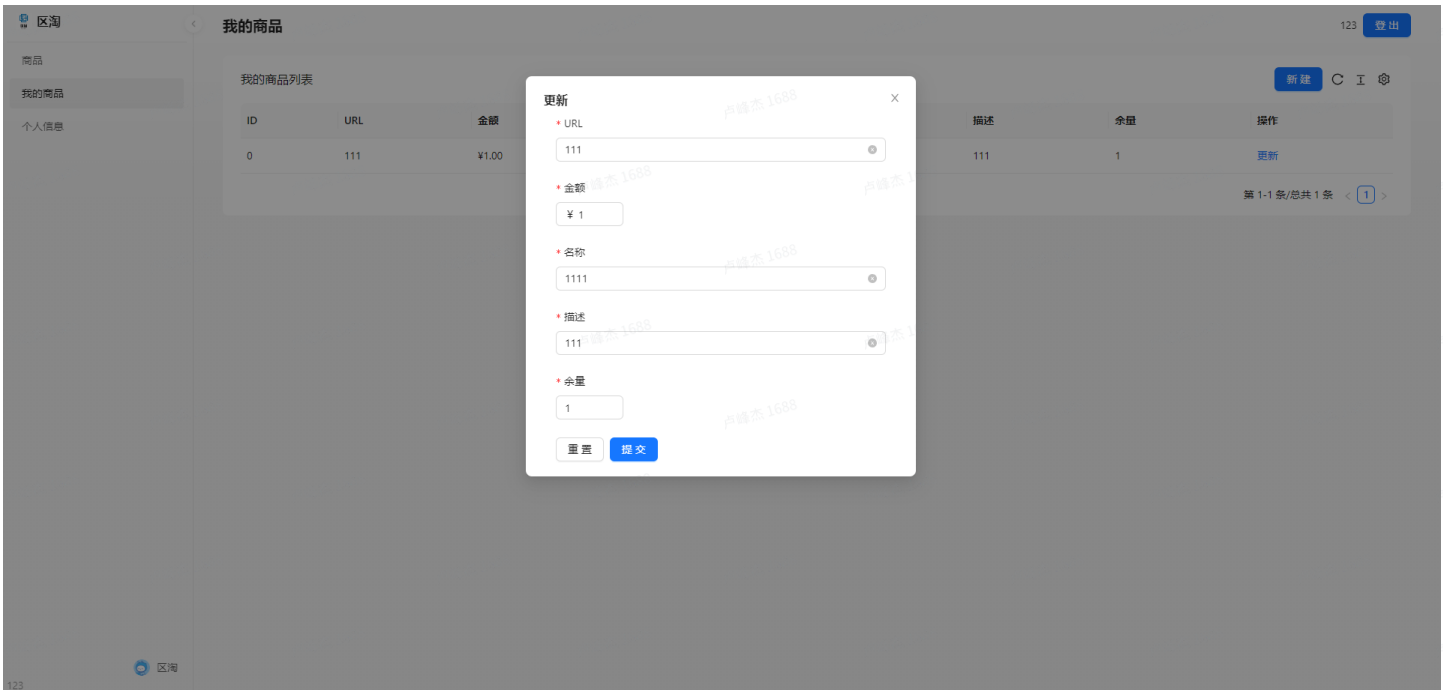



## 4.6 添加商品

提供商品的相关信息，然后即可添加成功



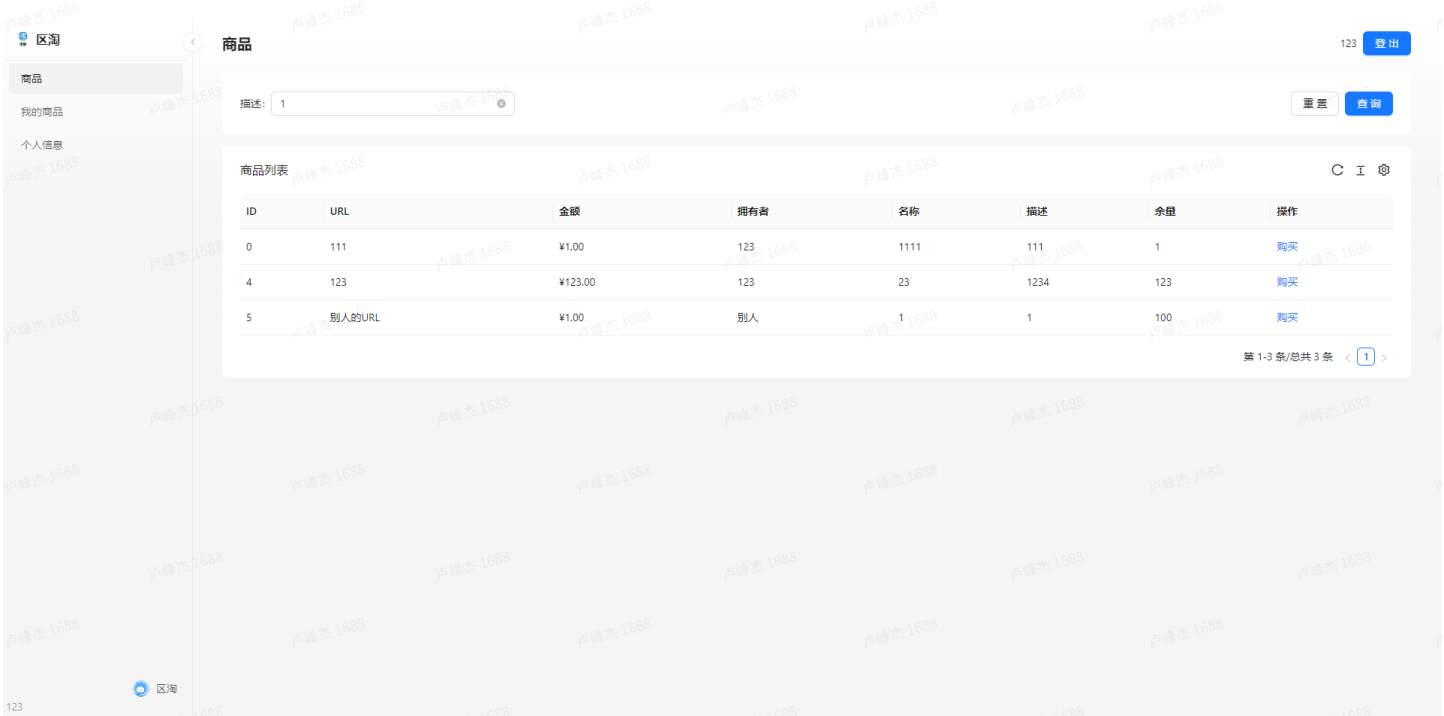

## 4.7 修改商品信息



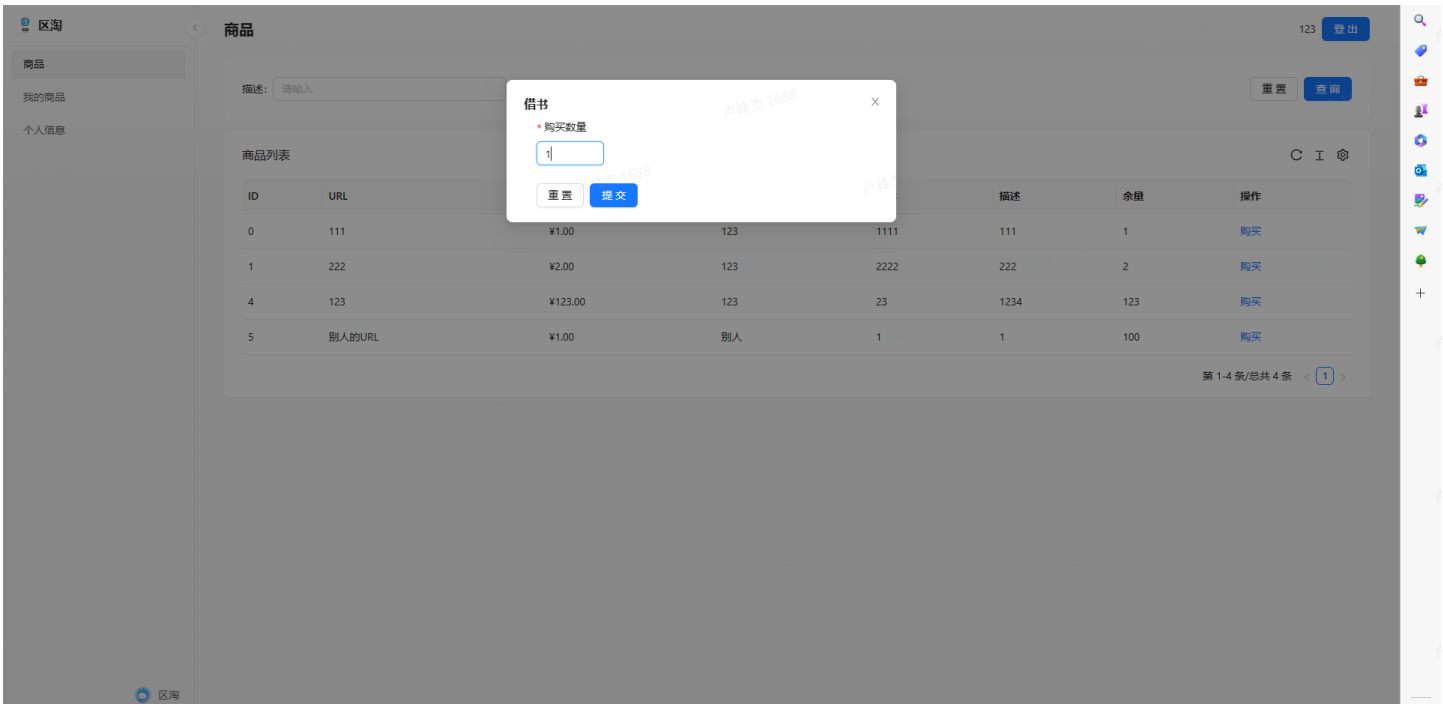

## 4.8 搜索商品

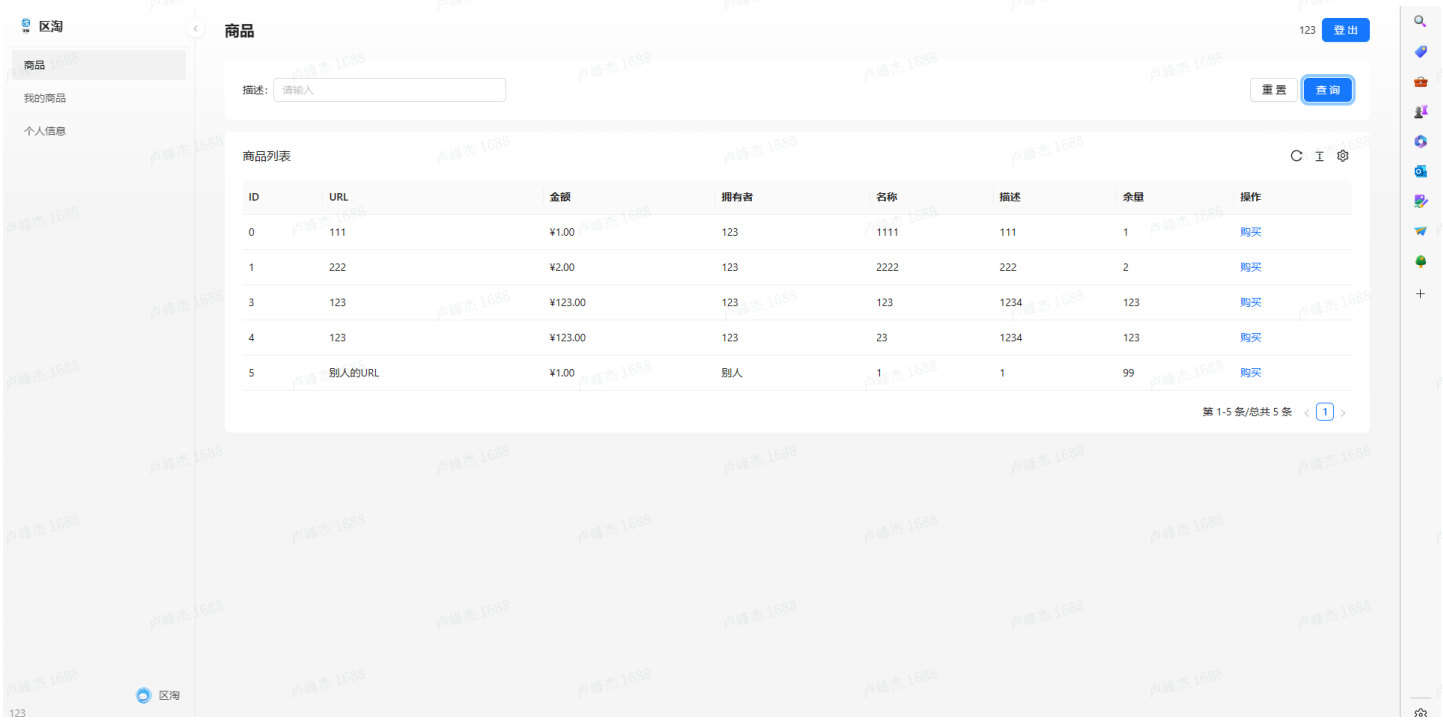
可以根据描述进行搜索

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！