接口设计说明

Tomato

2018 年 1 月

1 引言

1.1 系统概述

接口主要分为两大类的接口，一类是 Admin 的接口，另一类是 Client 的接口。Client 有两种接口，一种是 web 端的接口，另一类是微信端的接口。Admin 本身是一个 package，两种 Client 公用一个 package，名为 client。Web 端的接口包括了正常 HttpRequest 和 Web 端的 Websocket。Web 端的 Websocket 使用 StompClient 协议，在后端使用 Spring Boot 自带的 Controller，即 @MessageMapping 和 @SendTo 来完成对 socket 内容的转发。而微信端在 HttpRequest 上和 Web 端共用一个 API，并使用裸的 socket 来回复 websocket。

1.2 文档概述

文档首先介绍了使用的文献，然后对于每个特定的接口进行了具体的说明。

2 引用文件

本文档引用了我们在开发过程中撰写的《RestAPI》文档，详情可查看RestAPI.tex。

3 Admin 接口

3.1 Login Admin

管理员登录。

这里 Controller 的实现是查询数据库，看是否 username 和 password 匹配，如果不匹配则返回错误，如果匹配则返回登陆成功，并附上一个 token。

Request

```
POST /api/admin/login

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json

{
    "username": "admin",
    "password": "admin",
}
```

Returns

```
HTTP 200 OK
{
    "username": "admin",
    "token": "1283091828021803120",
}
```

Error

```
HTTP 401 NOTAUTHORIZED
{
    "error": "Admin with username admin doesn\'t exist or password is wrong."
}
```

3.2 Change Admin Password

更改管理员密码。

Controller 这里的实现是先 query 数据库，对拍原用户名和密码是否正确，如果不正确则直接返回错误，如果正确则用新密码更新数据库中的表项。

Request

```
POST /api/admin/update

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json

{
    "username": "admin",
```

```
    "prePassword": "admin",
    "newPassword": "newpwd",
}
```

Returns

```
HTTP 200 OK
```

Error

```
HTTP 401 NOTAUTHORIZED
{
    "error": "Wrong password of admin."
}
```

3.3 Get All Competitions

列出全部比赛。

Status 是”not_start”, ”auction_not_record”, ”auction_recorded”, ”trade”, ”rest”, ”end”之一。

服务器 query 数据库获得所有比赛的信息。并把它按照 API 格式返回给前端。

Request

```
GET /api/admin/competition/getall

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK

[
    {
        "id": "competiton1_id",
        "username": "competition1",
        "status": "auction"
    },
    {
        "id": "competiton2_id",
        "username": "competition2",
```

```
        "status": "end"
    }
]
```

Error

```
HTTP 204 NO CONTENT
```

3.4 Create Competition

新建一场比赛。注意，底层也要生成机器的 id。注意每场比赛的基本配置（比赛名称，参赛人数）只能创建一次，不能修改。

后端把前端发来的 json Parse 完之后新建所有的 round，把 round 信息填写完之后用 round 和其他信息组成比赛。在创建的过程中还要先创建 team，再喝 competition 链接上。如果创建的信息有问题则返回有问题。

Request

```
POST /api/admin/competition/new

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json

{
    "username": "competition_username",
    "round": 2,
    "startWealth": 1000,

    "teamNum": 2,
    "participantNum": 3,
    "team":
    [
        {
            "username": "team1",
            "participant": ["mem11", "mem12", "mem13"],
            "password": "111111",
        },
        {
            "username": "team2",
            "participant": ["mem21", "mem22", "mem23"],
            "password": "222222",
        }
```

```
    ]

    "roundParameter":
    [
        {
            "machineStartPrice": [300, 350, 400],
            "machineNum": [1, 1, 1],
            "materialProduceCost": [10, 20, 30],
            "time": 900,
        },
        {
            "machineStartPrice": [300, 350, 400],
            "machineNum": [1, 1, 1],
            "materialProduceCost": [10, 20, 30],
            "time": 900,
        }
    ]
}
```

Returns

```
HTTP 201 CREATED
```

Error

```
HTTP 404 NOT FOUND
{
    "error":"Unable to delete. Competition with id xxx not found."
}
```

3.5 Delete Competition By ID

通过 ID 删除比赛。

Controller 首先检查是否存在这个 ID，如果不存在直接返回错误，否则就删除数据库中的这个比赛和比赛相关的 round 和 team，然后返回删除成功。

Request

```
DELETE /api/admin/competiton/id={competition_id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
[
   {
      "id": "competiton2_id",
      "username": "competition2",
      "status": "end"
   }
]
```

Error

```
{
   "error":"Unable to delete. Competition with id xxx not found."
}
```

3.6 Update Competition Status

需要进入下一环节时，管理员端会向服务器发送更新比赛状态的请求，服务器返回当前比赛信息以便管理员端更新到最新的比赛状态。

Controller 根据发送的 status 和时间来更新后台数据库。这是一个 websocket，所以还需要把它转发给所有的 team，然后 team 来判断 competition 的 id 是否相同来选择更新。

Web Socket

```
MessageMapping: /api/admin/status/update/id=3

SendTo: /api/admin/status/id=3

EndPoint: http://127.0.0.1:8090/competitionStatus

Send JSON Pattern:
{
   "status": "auction"("not_start", "auction_not_record", "auction_recorded", "trade", "rest", "end")
   "round": 0/1/2/3
   "timeLeft":227(s)
 }

Get JSON Pattern:
{
   "status": "auction"("not_start", "auction_not_record", "auction_recorded", "trade", "rest", "end")
   "round": 0/1/2/3
}
```

Error

```
HTTP 404 NOT FOUND
{
   "error":"Unable to update. Competition with id xxx not found"
}
```

3.7 Get Auction Machine

获得某场比赛某一轮拍卖机器的初始信息。

Controller 向数据库 query 比赛数据，主要是某一轮拍卖的机器有哪些，有多少个之类的。然后返回给前端。

Request

```
GET /api/admin/competition/auction/id={id}/round={round}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
[
   {
      "machineId": "machine1",
      "type": "Wood",
      "startPrice": 200,
   },
   {
      "machineId": "machine2",
      "type": "Brick",
      "startPrice": 300,
   },
   {
      "machineId": "machine3",
      "type": "Cement",
      "startPrice": 400,
   }
]
```

Error

```
HTTP 404 NOT FOUND
```

```
{
   "error": "Competition with id xxx not found." (or Competition with id xxx does not have round xxx)
}
```

3.8 Record Auction Result

登记某场比赛某一轮的拍卖结果。

Controller 根据前端发给后端的 auction 得到的结果，来更新后端数据库的 machine 的拥有情况。主要是 update machine 的 owner 和 set 一个队伍有的 machine，并扣除相应的钱。如果没钱则返回错误。

Request

```
POST /api/admin/competition/record/id={id}/round={round}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
[
   {
      "machineId": "machine1",
      "teamId": "team1",
      "price": 2000,
   },
   {
      "machineId": "machine2",
      "teamId": "brick",
      "price": 3000,
   },
   {
      "machineId": "machine3",
      "teamId": "cement",
      "price": 4000,
   }
]
```

Error

```
HTTP 404 NOT FOUND
```

```
{
   "error": "Competition with id xxx not found." (or Competition with id xxx does not have round xxx)
}
```

3.9 Get Competition Property

从服务器按 id 获取某一比赛的各种属性。如果该比赛的属性尚未被设置，则该项为空。属性包括名称、比赛轮数（如果比赛已开始，则不能删除已开始或结束的轮）、比赛各项参数（不能修改已开始或结束的轮的参数）、机器的 id 等等。

Controller 根据 id 把后端关于这个比赛的信息打包，发给前端。

Request

```
GET /api/admin/competition/property/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
{
   "id": "competition_id",
   "username": "competition_username",
   "status": "not started",
   "teamNum": 1,
   "participantNum": 2,
   "team":
   [
      {
         "username": "team1",
         "participant": ["member1", "member2", "member2"],
         "password": "password",
      }
   ]
   "round": 1,
   "startWealth": 1000,
   "roundParameter":
   [
      {
         "machineStartPrice": [300, 350, 400],
         "machineNum": [1, 1, 1],
         "materialProduceCost": [10, 20, 30],
```

```
            "time": 900,
        }
    ]
}
```

Error

```
HTTP 404 NOT FOUND
{
    "error": "Competition with id xxx not found."
}
```

3.10 Update Competition Property

更新比赛的各种属性。属性包括名称、比赛轮数（如果比赛已开始，则不能更改）、比赛各项参数（不能修改已开始或结束的轮的参数）。

Controller 首先向创建比赛一样看比赛内容是否有格式错误等问题，如果有直接返回错误，否则根据发来的数据更新数据库中的表项。

Request

```
PUT /api/admin/competition/property/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json

{
    "round": 2,
    "startWealth": 1000,
    "round_parameter":
    [
        {
            "machineStartPrice": [300, 350, 400],
            "machineNum": [1, 1, 1],
            "materialProduceCost": [10, 20, 30],
            "time": 900,
        },
        {
            "machineStartPrice": [300, 350, 400],
            "machineNum": [1, 1, 1],
            "materialProduceCost": [10, 20, 30],
            "time": 900,
        }
```

```
    ]
}
```

Returns

```
HTTP 201 CREATED
```

Error

```
HTTP 404 NOT FOUND
{
    "error": "Competition with id xxx not found."
}
```

```
HTTP 400 INVALID REQUEST
{
    "error": "Cannot update competition id xxx with given changes."
}
```

3.11 Get Competition Information

获取当前比赛信息，包括队伍的数量、资产、交易记录、机器的使用情况等。Controller 根据比赛 ID 获取当前所有的比赛信息返回给前端。

Request

```
GET /api/admin/competition/info/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
{
    "id": "competition_id",
    "username": "competition_username",
    "status": "not_start",
    "round": 2,
    "presentRound": 0,
    "teamInfo":
    [
        {
```

```
        "id": "id1",
        "wealth": 100,
        "material": [30, 40, 50],
        "machine":
        [
            {
               "id": "machine1_id",
               "type": "type1",
               "left": 3
            },
            {
               "id": "machine2_id",
               "type": "type2",
               "left": 2
            }
        ]
    },
    {
        "id": "id2",
        "wealth": 100,
        "material": [30, 40, 50],
        "machine":
        [
            {
               "id": "machine1_id",
               "type": "type1",
               "left": 3
            },
            {
               "id": "machine2_id",
               "type": "type2",
               "left": 2
            }
        ]
    }
],
"trade_history":
[
    {
        "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
        "sell": "team_id1",
        "buy": "team_id2",
        "content": {"wood": 1},
        "price": 10
    },
     {
        "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
```

```
            "sell": "team_id1",
            "buy": "team_id2",
            "content": {"machine_wood": 1},
            "price": 20
        }
    ]
}
```

content 中是 wood，brick，cement，machine_wood，machine_brick，machine_cement 中的一个。

Error

```
HTTP 404 NOT FOUND
{
    "error": "Competition with id 1 not found."
}
```

4 Client 端接口

首先是 HttpRequest 的接口。

4.1 Login Client

用户登录。

Controller 的实现和 Admin 的登陆类似，这里就不赘述了。

Request

```
POST /api/client/login

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json

{
    "username": "client",
    "password": "client",
}
```

Returns

```
HTTP 200 OK
{
   "username": "client",
    "id": "3",
   "token": "1283091828021803120",
}
```

Error

```
HTTP 401 NOTAUTHORIZED
{
   "error": "Client with userusername admin doesn\'t exist or password is wrong."
}
```

4.2 Get Information

这个接口在用户登录的过程中被使用，当用户登录之后，用户将其 id 发送给服务器，服务器返回用户当前的状态信息，包括队伍中有哪些人，当前比赛状态，队伍排名等信息。

注：若比赛未开始，则 rank 为 0。

Controller 的实现为通过 id 查找队伍，然后计算队伍排名。排名的计算方法是先换算房子，房子多的排名高，如果房子数相同，则钱多的队伍排名高。最后把这些信息返回给前端。

Request

```
GET /api/client/info/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns »»»> 20eb624d3e96976ee93450dcd011364bb4cf34ae

```
HTTP 200 OK

{
  "memberList":
   [
      "member1":"wangmz",
      "member2":"songsh"
   ],
   "username": "team1",
   "id":3,
```

```
    "rank": 1,

    "gameStatus": "auction"( "not_start", "auction_not_record", "auction_recorded", "trade", "rest", "end" )
    "round": 0/1/2/3 (第 ( round+1 ) 轮)
    "timeLeft":300(s)
}
```

Error

```
HTTP 404 NOT FOUND
```

4.3 Get Property

输入 ID，获得与这一 ID 相关的用户的财产信息。包括机器的使用情况和材料的价格。Controller 的实现是直接把机器的使用情况和 competition 中材料的价格返回给前端。

Request

```
GET /api/client/property/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK

{
  "wealth": 3000,
  "machine":
   [
      {
         "id": 0073,
         "type": "Cement",
         "left": 3
         "lock":false
      },
      {
         "id": 0793,
         "type": "Brick",
         "left": 0
         "lock":true(正处于出售中的机器和材料 lock == true)
      },
```

```
        {
            "id": 8765,
            "type": "Wood",
            "left": 2
            "lock":false
        }
    ],
    "material":
    [
        {
            "type": "Wood",
            "price": 10,
            "number": 20,
            "lock":true
        },
        {
           "type": "Brick",
            "price": 20,
            "number": 0,
            "lock":false
        },
        {
          "type": "Cement",
            "price": 80,
            "number": 150,
            "lock":false
        }
    ]
}
```

Error

```
HTTP 404 NOT FOUND
```

4.4 Get All User

get 所有队伍，(除了发送消息的队伍)，用来发 sell Request 时进行选择。

Controller 的实现是通过发来的 id 查找比赛，然后找到比赛里的所有 team，然后返回给前端。

Request

```
GET /api/client/getAllUser/id={id}

Host: localhost:8080
```

```
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK

{
   [
      {
      "teamId": 889,
         "username": "dddd",
      },
      {
            "teamId": 999,
         "username": "ddfadf",
      },
   ]
}
```

Error

```
HTTP 404 NOT FOUND
```

4.5 Get Trade History

交易历史信息。在发订单的时候客户端手动更新 History。

Controller 在每次交易创建的时候都会把交易的 ID 存在 team 的表中，在每次 HttpRequest 请求 trade history 的时候就通过 id 找到 trade，并把 trade 的主要信息返回给前端。

Request

```
GET /api/client/tradehHistory/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
      [
         {
```

```
        "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
        "target": "team1",
        "action": "sell",
        "content": "wood",
        "price": 10,
        "number": 2
        "status": 1,(完成)
        "tradeId":44,
        "buyerId":9
        },

        {
        "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
        "target": "team1",
         "action": "buy",
        "content": "1234" (machine.id ==1234)
        "price": 10,
        "number": 1 (只能是1)
        "status": 0, (正在进行)
        "tradeId":44,
        "buyerId":9
        },

        {
        "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
        "target": "team1",
        "action": "buy",
        "content": "6666" (machine.id ==1234)
        "price": 10,
        "number": 1 (只能是1)
        "status": -1, (失败)
        "tradeId":44,
        "buyerId":9,
        }
    ]
```

Error

```
HTTP 404 NOT FOUND
```

4.6 Get Produce History

生产历史信息。

Controller 的实现是每一次生产的时候都记录一下 Produce 的 id 到 team 的 ProduceList 里面。然后 query 的时候直接通过 id 找到 Produce 并把信息返回回去。

Request

```
GET /api/client/produceHistory/id={id}

Host: localhost:8080
Auth:
Content-type: application/json
Accept: application/json
```

Returns

```
HTTP 200 OK
      [
         {
         "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
         "machineId":9987
         "content": "Brick",
         "price": 10,
         "number": 2
         },
         {
         "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
         "machineId":3457
         "content": "Wood",
         "price": 10,
         "number": 2
         },
         {
         "time": "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",
         "machineId":5777
         "content": "Cement",
         "price": 10,
         "number": 2
         },
      ]
```

Error

```
HTTP 404 NOT FOUND
```

然后是 websocket 的接口。

4.7 交易：发出出售请求，buyer 监听

Controller 的实现是先把出售请求存下来，然后把它原样转发给买方。

```
MessageMapping: /api/client/property/sellerId={sellerId}/buyerId={buyerId}
```

```
SendTo: /api/client/property/buyerId={buyerId}

EndPoint: http://127.0.0.1:8090/trade

Get JSon Pattern:( 卖方发送 )
{
   "tradeId": '',
   "sellerId":sellerId,
    "buyerId":buyerId,
    "buyer":"TOMATO"
    "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
    "price":300,
    "number":7
    "seller":"Rua"
}
Send JSon Pattern:( 发给买方 )
{
   "tradeId": tradeId,
   "sellerId": sellerId,
    "buyerId": buyerId,
    "buyer": "TOMATO"
    "typeOrMachineID": "9987" ("Wood" "Cement" "Brick" OR machineID)
    "price": 300,
    "number": 7
    "seller": "Rua"
}
```

4.8 交易结束给卖家转发账单和现有资产

Controller 的实现是根据交易的内容修改交易的状态，然后把账单转发给卖家。

```
MessageMapping: /api/client/tradeFinish/sellerId={sellerId}

SendTo: /api/client/tradeFinish/id={sellerId}

EndPoint: http://127.0.0.1:8090/tradeFinish

Get JSon Pattern:
{
   "tradeId":tradeId,
   "sellerId":sellerId,
   "buyerId":buyerId,
   "buyer":"TOMATO"
   "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
   "price":300,
   "number":7
```

```
    "seller":"Rua"

     "isAccept":true
}

Send JSon Pattern:
{
    "reply":
    {
    "tradeId":tradeId,
    "sellerId":sellerId,
    "buyerId":buyerId,
    "buyer":"TOMATO"
    "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
    "price":300,
    "number":7
    "seller":"Rua"

    "isAccept":true
     }


   "propertyList":
   {
    "wealth":1000,
    "machineList":
        [
        {
            "id": 0073,
            "type": "Wood",
            "left": 3
            "lock":false
        },
        ],
        "materialList":
        [
        {
            "type": "Brick",
            "price": 20,
            "number": 0,
            "lock":false
        },
        ]
    }
}
```

4.9 交易结束给队友转发账单和现有资产

Controller 的实现是根据交易的内容修改交易的状态，然后把账单转发给卖家。

```
MessageMapping: /api/client/tradeFinish/buyerId={buyerId}

SendTo: /api/client/tradeFinish/id={buyerId}

EndPoint: http://127.0.0.1:8090/tradeFinish

Get JSon Pattern:
{
   "tradeId":tradeId,
   "sellerId":sellerId,
   "buyerId":buyerId,
   "buyer":"TOMATO"
   "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
   "price":300,
   "number":7
   "seller":"Rua"

    "isAccept":true
}

Send JSon Pattern:
{
   "reply":
   {
   "tradeId":tradeId,
   "sellerId":sellerId,
   "buyerId":buyerId,
   "buyer":"TOMATO"
   "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
   "price":300,
   "number":7
   "seller":"Rua"

   "isAccept":true
    }


  "propertyList":
  {
   "wealth":1000,
   "machineList":
      [
      {
         "id": 0073,
```

```
            "type": "Wood",
            "left": 3
            "lock":false
        },
        ],
        "materialList":
        [
        {
            "type": "Brick",
            "price": 20,
            "number": 0,
            "lock":false
        },
        ]
    }
}
```

4.10 监听比赛状态改变

Controller 的实现是当 Admin 更换比赛状态的时候, 把比赛状态的具体信息转发给 client。

```
MessageMapping: /api/client/competition/status/id={id}

SendTo: /api/client/competitionStatus/id={id}

EndPoint: http://127.0.0.1:8090/hhh

Send JSon Pattern:
{
      "gameStatus": "auction"( "not_start", "auction_not_record", "auction_recorded", "trade", "rest", "
          end" )
      "round": 0/1/2/3 (第( round+1 )轮)
      "timeLeft":227(s)

}

Get JSON Pattern:
{
   "gameStatus": "auction"( "not_start", "auction_not_record", "auction_recorded", "trade", "rest",
   "round": 0/1/2/3 (从0开始)
}
```

4.11 监听 produce 后资产的改变

Controller 的实现是把 produce 看成一个 websocket，像 HttpRequest 一样处理这个 produce 请求，然后所有的 client 都监听这个 websocket，最后把生产完的账单转发给所有的队友。

```
MessageMapping: /api/client/ListenProperty/id=3

SendTo: /api/client/ListenProperty/receive/id=3

EndPoint: http://127.0.0.1:8090/listenProperty

Send JSon Pattern:
{
  "wealth":1000,
  "machine":
   [
      {
         "id": 0073,
         "type": "Wood",
         "left": 3
         "lock":false
      },
      {
         "id": 0793,
         "type": "Brick",
         "left": 0
         "lock":true
      },
      {
         "id": 8765,
         "type": "Cement",
         "left": 2
         "lock":false
      }
   ],
   "material":
   [
      {
        "type": "Wood",
         "price": 10,
         "number": 20,
         "lock":false
      },
      {
        "type": "Brick",
         "price": 20,
```

```
          "number": 0,
          "lock":false
      },
      {
         "type": "Cement",
          "price": 80,
          "number": 150,
          "lock":false
      }
   ]
}

Get JSON Pattern:
{
   "id":2,
   "times":1,
}
}
```

4.12 撤销，卖方监听

Controller 的实现是修改订单的状态，然后把回执转发给卖方。

```
MessageMapping: /api/client/undo/sendToSeller/sellerId={sellerId}/buyerId={buyerId}

SendTo: /api/client/receiveUndo/id={sellerId}

EndPoint: http://127.0.0.1:8090/undo

Get JSon Pattern:(卖方发送)
{
   "tradeId": 77,
}
Send JSon Pattern:(发给卖方)
{
   "request":
   {
   "tradeId":77,
   "sellerId":sellerId,
   "buyerId":buyerId,
   "buyer":"TOMATO"
   "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
   "price":300,
   "number":7
   "seller":"Rua"
    }
```

```
  "propertyList":
  {
   "wealth":1000,
   "machineList":
      [
      {
         "id": 0073,
         "type": "Wood",
         "left": 3
         "lock":false
      },
      ],
      "materialList":
      [
      {
         "type": "Brick",
         "price": 20,
         "number": 0,
         "lock":false
      },
      ]
   }
}
```

4.13 撤销，买方监听

Controller 的实现是修改订单的状态，然后把回执转发给买方。

```
MessageMapping: /api/client/undo/sendToBuyer/sellerId={sellerId}/buyerId={buyerId}

SendTo: /api/client/receiveUndo/id={buyerId}

EndPoint: http://127.0.0.1:8090/undo

Get JSon Pattern:(卖方发送)
{
   "tradeId": 77,
}
Send JSon Pattern:(发给买方)
{
   "request":
   {
   "tradeId":77,
   "sellerId":sellerId,
   "buyerId":buyerId,
```

```
  "buyer":"TOMATO"
  "typeOrMachineID":"9987" ("Wood" "Cement" "Brick" OR machineID)
  "price":300,
  "number":7
  "seller":"Rua"
   }

 "propertyList":
 {
  "wealth":1000,
  "machineList":
     [
     {
        "id": 0073,
        "type": "Wood",
        "left": 3
        "lock":false
     },
     ],
     "materialList":
     [
     {
        "type": "Brick",
        "price": 20,
        "number": 0,
        "lock":false
     },
     ]
  }
}
```