

# 软件工程大作业文档

Tomato

2018 年 1 月

## 1 系统/子系统设计（结构设计）说明（后台部分）

### 1.1 引言

#### 1.1.1 系统概述

后台分为两个部分，上层是 **Controller**，是用来处理业务逻辑的，下层是数据库，给上层的 **Controller** 提供相应的服务。下面会分两块来具体介绍这两部分。详见2和3。

#### 1.1.2 文档概述

本文档将围绕 **Spring Boot** 架构来介绍后台部分。

### 1.2 系统级设计决策

本系统后台主要使用 **Spring Boot** 架构，它支持在开启服务的时候，接受 **HttpRequest** 或 **websocket** 并进行后台处理，然后返回相应的结果。系统接受的输入是 **HttpRequest** 或 **websocket**，详见2，对于每一个输入的响应，被 **Controller** 定义，不同的 **request** 会有不同的 **Controller** 对其进行处理并返回相应的结果。系统处理的过程中，依赖于两部分的内容：用户的 **request** 中的具体内容和底层数据库已有的信息。详见??。该系统的数据库存在服务器上，服务器的管理人员可以通过 **database asdan** 来访问数据库并查看当前数据库中的所有信息。该系统对服务器的要求是要求服务器能通过 **maven** 运行项目，并配有 **mysql** 数据库。

### 1.3 系统体系结构设计

#### 1.3.1 系统总体设计

**1.3.1.1 概述** 本系统要实现的功能详见《prj9ASDAN 模拟商业竞拍交易大赛系统》。本系统要实现的性能：响应较快、支持并发、有一定的安全性、可在各个平台上运行、可移植等。本系统支

持在 Windows、Unix、Linux 环境下运行。

**1.3.1.2 设计思想** 本系统顶层核心要处理的两个问题是对于 `HttpRequest` 的处理和对于 `websocket` 的处理，而底层核心要处理的问题是数据库的维护等问题。如果直接从头开始开发一套框架，则要实现的内容可能比我们当前写的多得多。所以我们后台根据这个需求找到了 `Spring Boot` 框架。`Spring Boot` 很好地通过了 `Annotations` 来实现我们需要的对于前端 `web` 端的响应，也实现了我们需要的和数据库的连接、并发的响应等等。本系统并不需要很高深的算法，但是对逻辑上的要求十分的多和细致。

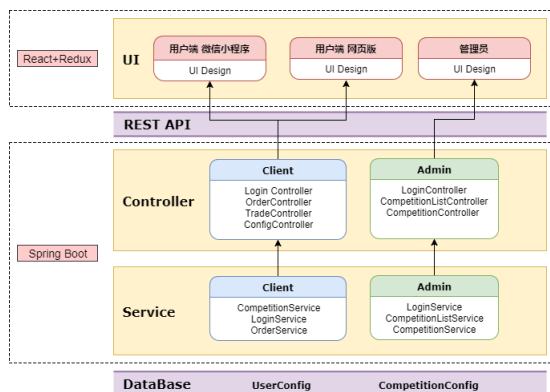
### 1.3.1.3 基本处理流程

### 1.3.1.4 系统体系结构

### 1.3.1.5 功能需求与系统配置项的关系

## 1.3.2 执行概念

## 1.3.3 接口设计



## 1.4 系统出错处理设计

系统出错后，后台数据库的内容不会被清空，并且我们是在处理 `request` 流程中对数据库是实时更新的，宕机之后数据库的内容还是会保留，当下次启动服务器的时候它还能从原数据库中获得原来的数据，并根据这些数据进行处理。

当系统出错后，建议重启服务器。

## 1.5 尚待解决的问题

微信后端的 websocket 的测试问题。

## 1.6 需求的可追踪性

## 1.7 注解

# 2 接口设计说明

## 2.1 引言

### 2.1.1 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。

### 2.1.2 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。

## 2.2 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](#)。

## 2.3 Admin 接口

### 2.3.1 Login Admin

管理员登录。

```
POST /api/admin/login
```

```
Host: localhost:8080
```

```
Auth:
```

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

Accept: application/json

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

HTTP 200 OK

```
{  
  "username": "admin",  
  "token": "1283091828021803120",  
}
```

HTTP 401 NOTAUTHORIZED

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

### 2.3.2 Change Admin Password

更改管理员密码。

POST /api/admin/update

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

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

```
HTTP 200 OK
```

```
HTTP 401 NOTAUTHORIZED
```

```
{  
  "error": "Wrong password of admin."  
}
```

### 2.3.3 Get All Competitions

列出全部比赛。

Status 是"not\_start", "auction\_not\_record", "auction\_recorded", "trade", "rest", "end" 之一。

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

```
Host: localhost:8080
```

```
Auth:
```

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

```
Accept: application/json
```

```
HTTP 200 OK
```

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

#### 2.3.4 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。

```
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,
      }
    ]
  }
}
```

HTTP 201 CREATED

HTTP 404 NOT FOUND

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

### 2.3.5 Delete Competition By ID

通过 ID 删除比赛。

**DELETE** /api/admin/competiton/id={competition\_id}

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

HTTP 200 OK

```
[
  {
    "id": "competiton2_id",
```

```

    "username": "competition2",
    "status": "end"
  }
]

```

```

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

```

### 2.3.6 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。

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
}

```



HTTP 404 NOT FOUND

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

### 2.3.7 Get Auction Machine

获得某场比赛某一轮拍卖机器的初始信息。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

HTTP 200 OK

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

HTTP 404 NOT FOUND

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

### 2.3.8 Record Auction Result

登记某场比赛某一轮的拍卖结果。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

HTTP 200 OK

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

HTTP 404 NOT FOUND

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

### 2.3.9 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。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

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,
    }
  ]
}

```

HTTP 404 NOT FOUND

```

{
  "error": "Competition with id xxx not found."
}

```

### 2.3.10 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）。

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,
  }
]
```

HTTP 201 CREATED

HTTP 404 NOT FOUND

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

HTTP 400 INVALID REQUEST

```
{
  "error": "Cannot update competition id xxx with given changes."
}
```

### 2.3.11 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。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

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。

```

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

```

### 2.3.12 Record machine owner

向服务器发送对比赛的更新信息。增加机器、分配财产之类的。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

```

{
  "round": 2,
  "present_round": 0,
  "team_info":
  [
    {
      "id": "id1",
      "wealth": "100",
      "machine":
      [
        {
          "id": "machine1_id",
          "left": 3
        },
        {
          "id": "machine2_id",
          "left": 2
        }
      ]
    },
    {
      "id": "id2",
      "wealth": "100",
      "machine":
      [
        {
          "id": "machine1_id",
          "left": 3
        },
        {
          "id": "machine2_id",
          "left": 2
        }
      ]
    }
  ]
}

```

HTTP 200 OK

```

{
  "id": "competition_id",
  "username": "competition_username",
  "status": "not started",

```



```

"round": "2",
"present_round": "0",
"team_info":
[
  {
    "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": "hh:MM:ss",

```

```

        "sell": "team_id1",
        "buy": "team_id2",
        "content": {"wood": 1},
        "price": 10
    },
    {
        "time": "hh:MM:ss",
        "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。

HTTP 404 NOT FOUND

```

{
  "error": "Competition with id xxx not found."
}

```

HTTP 400 INVALID REQUEST

```

{
  "error": "Cannot update competition id xxx with given information."
}

```

## 2.4 Client 端接口

首先是 HttpRequest 的接口。

### 2.4.1 Login Client

用户登录。

```
POST /api/client/login

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

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

```
HTTP 200 OK

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

```
HTTP 401 NOTAUTHORIZED

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

#### 2.4.2 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**。

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

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

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) 轮+1)
  "timeLeft":300(s)
}
```

HTTP 404 NOT FOUND

### 2.4.3 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。

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

Host: localhost:8080

Auth:

Content-type: application/json

Accept: application/json

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
    }
]
}

```

HTTP 404 NOT FOUND

#### 2.4.4 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

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

```
Host: localhost:8080
```

```
Auth:
```

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

```
Accept: application/json
```

```
HTTP 200 OK
```

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

```
HTTP 404 NOT FOUND
```

#### 2.4.5 Get Trade History

交易历史信息。在发订单的时候客户端手动更新 History。

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

```
Host: localhost:8080
```

```
Auth:
```

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

HTTP 200 OK

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

HTTP 404 NOT FOUND

#### 2.4.6 Get Produce History

生产历史信息。

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

```
Host: localhost:8080
```

```
Auth:
```

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

```
Accept: application/json
```

```
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
  },
]
```

```
HTTP 404 NOT FOUND
```

然后是 websocket 的接口。



#### 2.4.7 交易：发出出售请求，buyer 监听

```
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"
}
```

#### 2.4.8 交易结束给卖家转发账单和现有资产

```
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
      },
    ],
  }
}

```

```

    ]
  }
}

```

#### 2.4.9 交易结束给队友转发账单和现有资产

```

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
    },
  ]
}

```

#### 2.4.10 监听比赛状态改变

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) 轮+1)
  "timeLeft":227(s)
}

```

Get JSON Pattern:

```

{
  "gameStatus": "auction"
    ("not_start", "auction_not_record", "auction_recorded", "trade", "rest",

```

```
"round": 0/1/2/3 从开始(0)
}
```

#### 2.4.11 监听 produce 后资产的改变

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,
}
}

```

#### 2.4.12 撤销，卖方监听

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
      },
    ]
  }
}

```

#### 2.4.13 撤销，买方监听

```

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
    },
  ]
}
}

```

## 2.5 需求的可追踪性

## 2.6 注解



## 3 数据库（顶层）设计说明

### 3.1 引言

#### 3.1.1 标识

数据库版本为 1.0.0。

#### 3.1.2 数据库概述

数据库的用途是存放必要的用户管理数据和比赛数据。项目的用户是 ASDAN 商业竞赛，开发方是 Tomato 团队。

#### 3.1.3 文档概述

本文档的用途是介绍数据库。

### 3.2 引用文件

没有引用别的文件。

### 3.3 数据库级测试决策

### 3.4 数据库详细设计

#### 3.4.1 概念设计

#### 3.4.2 内部设计

#### 3.4.3 逻辑设计

数据库的操作实体类共有 8 个，接下来将详细介绍每个类内部的成员变量。

##### 3.4.3.1 Admin 存放管理员的类。

**serialVersionUID (static final long)** 用于序列化不同版本的类。

**adminId (Long)** 主键，存放管理员 ID。

**name (String)** 管理员名称。

**username (String)** 管理员用户名。

**password (String)** 管理员密码。

**enabled (boolean)** 管理员权限是否被启用。

**role (String)** 当前账户的性质，管理员为"ADMIN"。

**competitionList (List<Competition>)** 当前管理员管理的全部比赛。

##### 3.4.3.2 Competition 存放比赛的类。

**serialVersionUID (static final long)** 用于序列化不同版本的类。

**competitionId (Long)** 主键，存放比赛 ID。

**name (String)** 比赛名称。

**round (int)** 比赛一共有多少轮。

**presentRound (int)** 比赛当前进行到了第几轮。

**roundList (List<Round>)** 比赛每一轮的信息。

**teamList (List<Team>)** 比赛中每支队伍的信息。

**produceList (List<Produce>)** 比赛中全部生产的信息。

**competitionTradeList (List<Trade>)** 比赛中全部交易的信息。

**initial (int)** 比赛队伍的初始资产。

**status (Competition::Status)** 比赛状态，分为 NOT\_START（比赛未开始）、AUCTION\_NOT\_RECORDED（拍卖结果未记录）、AUCTION\_RECORDED（拍卖结果已记录）、TRADE（交易中）、REST（休息中）和 END（比赛已结束）6 种。

#### 3.4.3.3 Competition::Round

**roundId (Long)** 主键，存放轮 ID。

**startTime (Date)** 本轮开始时间。

**time (Integer)** 本轮持续时间，以分钟为单位。

**machineNumberMap (Map<Material, Integer>)** 每种机器的个数。

**machineStartPriceMap (Map<Material, Integer>)** 每种机器的起拍价格。

**machineList (List<Machine>)** 本轮拍卖的全部机器。

**machineAuctionPriceMap (Map<Machine, Integer>)** 每个机器的拍卖价格。

#### 3.4.3.4 Machine

#### 3.4.3.5 Material

#### 3.4.3.6 Produce

#### 3.4.3.7 Team

#### 3.4.3.8 Trade

#### 3.4.4 物理设计

### 3.5 用于数据库访问或操纵的软件配置项的详细设计

### 3.6 需求的可追踪性

### 3.7 注解

## 4 软件测试计划

### 4.1 引言

对于后端我们做了详尽的测试，包括 `Controller` 正确性的测试，`Service` 的测试和底层数据库的测试。

#### 4.1.1 系统概述

整个 `test` 文件包含三种 `test`，分别是 `Controller` 的 `test`，`Model` 的 `test` 和 `Service` 的 `test`。`Controller` 中的 `test` 又分为 `Admin` 的 `test` 和 `Client` 的 `test`。由于没有找到微信 `websocket` 的 `test` 方法，所以微信的 `websocket` 没有写 `test`。

#### 4.1.2 文档概述

本文档将介绍测试环境，测试计划以及测试进度。

#### 4.1.3 与其他计划的关系

测试是在接口实现中逐步完善的。与接口的计划相辅相成，可以说是写一个接口就写了一个 `test`。

### 4.2 软件测试环境

#### 4.2.1 后端测试

**4.2.1.1 软件项** 我们所用的语言是 `Java`，使用的框架是 `Spring Boot`，所有的测试都是搭建在这个框架之上的，写测试的方法也是使用了这个框架集成的 `test annotations`。

**4.2.1.2 参与组织** `Tomato` 后端开发组。

**4.2.1.3 人员** 张慧盟、宋世虹。

**4.2.1.4 要执行的测试** 如上所述，需要测试 `Controller`，`Model` 和 `Service`。

## **4.3 计划**

## **4.4 总体设计**

### **4.4.1 测试过程**

#### **4.4.2 Controller**

##### **4.4.2.1 Admin**

- `AdminLoginControllerTest`: 测试了两种情况，`Admin` 登陆成功和登录失败。
- `AuctionControllerTest`: 测试了 `Auction` 的接口，包括当 `get auction` 的时候应该返回什么，如果发起了错误的 `get auction` 操作时会返回什么，`post auction` 的时候 `Service` 的更新，`post auction` 错误的时候返回什么等等。
- `ChangePasswordControllerTest`: 测试了如果 `password` 输入正确的更新准则和输入错误时的报错。
- `CompetitionCreatorControllerTest`: 测试了创建比赛，更新比赛和获得比赛具体信息时的返回值，以及上述情况 `request` 出错时的返回值。
- `CompetitionInfomationControllerTest`: 测试了请求比赛信息时返回信息的正确性，和请求失败的返回值。
- `CompetitionStatusAdminControllerTest`: 测试了不同比赛状态时的返回值。
- `DeleteCompetitionControllerTest`: 测试了删除比赛时：正常删除、没有比赛和获得全部比赛出错的情况。
- `GetAllCompetitionControllerTest`: 测试了返回全部比赛时：正常返回、没有比赛、获得全部比赛出错的情况。
- `UpdateCompetitionStatusControllerTest`: 测试了更新比赛状态时的返回值：正常返回、没有比赛、后端没法 `update` 比赛状态、非法状态等情况。

##### **4.4.2.2 Client**

- `BuyMaterialControllerTest`: 测试了买东西时：买材料和买机器的正在交易、已经交易完成和交易取消的情况。

- **ClientInfoControllerTest**: 测试了获取和更新 **team** 信息时成功和失败时的情况。
- **ClientLoginControllerTest**: 测试了登陆成功和失败的情况。
- **ClientPropertyControllerTest**: 测试了 **team** 获取 **property** 时成功、失败的情况，测试了 **team** 生产的时候成功和失败的情况，
- **CompetitionStatusControllerTest**: 测试了更新比赛状态时 **Client** 的返回情况。
- **GetAllUsersControllerTest**: 测试了获取所有 **team** 的成功和失败的情况。
- **GetProduceHistoryControllerTest**: 测试了获取生产历史的成功和失败（没有 **team**，没有 **produce** 记录）的情况。
- **GetTradeHistoryControllerTest**: 测试了获取交易历史的成功和失败（没有 **team**，没有 **produce** 记录）的情况。
- **ListenPropertyControllerTest**: 测试了在 **websocket** 转发中监听 **produce** 后资产的变化。
- **SellMaterialControllerTest**: 测试了卖资产时成功的情况，包括各种材料和机器。
- **SendToSellerControllerTest**: 测试了将售卖的账单转给买方和卖方的同 **team**，主要测试了不同账单状态的回复情况。
- **UndoTradeControllerTest**: 测试了撤销交易的成功情况，主要测试了发给买方和卖方的账单。

#### 4.4.3 Service

- **admin.CompetitionServiceTest**: 测试了创建比赛、查找比赛、更新比赛、删除比赛的接口，测试了创建队伍、更新队伍、查找队伍的接口，测试了创建机器、查找机器、更新机器的接口，测试了获得所有 **Produce** 信息和交易信息的接口，测试了更新 **Round** 的接口。
- **admin.LoginServiceTest**: 测试了创建 **Admin**、更新 **Admin**、查找 **Admin** 和删除 **Admin** 的接口。
- **user.CompetitionServiceTest**: 测试了查找 **Competition** 和更新 **Competition** 的成功和失败的情况。
- **user.LoginServiceTest**: 测试了查找 **team**、更新 **team**、创建 **team** 和删除 **team** 的 **Service** 的成功和失败情况。
- **user.OrderServiceTest**: 测试了创建 **Produce**、查找 **Produce**、更新 **Produce**、删除 **Produce** 的接口，测试了创建 **Trade**、查找 **Trade**、更新 **Trade**、删除 **Trade** 的接口。

#### 4.4.4 Model

- AdminTest: 测试了 Admin 表的各 field。
- CompetitionTest: 测试了 Competition 表的各 field, 包括 Round。
- MachineTest: 测试了 Machine 表的各 field。
- MaterialTest: 测试了 Material 的 struct 是否正确。
- ProduceTest: 测试了 Produce 表的各 field。
- TeamTest: 测试了 Team 表的各 field。
- TradeTest: 测试了 Trade 表的各 field。

### 4.5 计划执行的测试

#### 4.5.1 被测试项

Tomato 的整个后端。

### 4.6 测试进度表

如下图所示。

server (com.java.asdan.tomato)	18s 346ms
▶ ServerApplicationTest	1ms
▶ AdminLoginControllerTest	4s 76ms
▶ AuctionControllerTest	1s 264ms
▶ ChangePasswordControllerTest	122ms
▶ CompetitionCreatorControllerTest	1s 12ms
▶ CompetitionInformationControllerTest	1s 621ms
▶ CompetitionStatusAdminControllerTest	1s 691ms
▶ DeleteCompetitionControllerTest	134ms
▶ GetAllCompetitionControllerTest	135ms
▶ UpdateCompetitionStatusControllerTest	353ms
▶ AvatarControllerTest	169ms
▶ BuyMaterialControllerTest	1s 668ms
▶ ClientInfoControllerTest	151ms
▶ ClientLoginControllerTest	177ms
▶ ClientPropertyControllerTest	770ms
▶ CompetitionStatusControllerTest	747ms
▶ GetAllUsersControllerTest	56ms
▶ GetProduceHistoryControllerTest	236ms
▶ GetTradeHistoryControllerTest	194ms
▶ ListenPropertyControllerTest	655ms
▶ SellMaterialControllerTest	785ms
▶ SendToSellerControllerTest	1s 310ms
▶ UndoTradeControllerTest	499ms
▶ WebSocketPushHandlerTest	0ms
▶ AdminTest	0ms
▶ CompetitionTest	0ms
▶ MachineTest	0ms
▶ MaterialTest	0ms
▶ ProduceTest	0ms
▶ TeamTest	3ms
▶ TradeTest	2ms
▶ CompetitionServiceTest	105ms
▶ LoginServiceTest	58ms
▶ CompetitionServiceTest	36ms
▶ LoginServiceTest	171ms
▶ OrderServiceTest	145ms
▶ CustomErrorTypeTest	0ms

4.7 需求的可追踪性

本测试计划覆盖了所有后端（除微信 WebSocket 外）的需求（接口）。

4.8 评价

4.8.1 评价准则

Coverage。

4.8.2 数据处理

数据处理由 IntelliJ 自动完成，IntelliJ 将为我们生成完整的 Coverage 报告。

4.8.3 结论

本测试计划十分合理。

4.9 注解

5 软件测试报告

5.1 引言

5.1.1 系统概述

可以参见《4》中的系统概述。

5.1.2 文档概述

本文档将介绍测试的详细结果，主要是 Coverage。

5.2 测试结果概述

5.2.1 对被测试软件的总体评估

97% classes, 80% lines covered in package 'com.java.asdan.tomato.server'

Element	Class, %	Method, %	Line, %
configuration	83% (5/6)	63% (7/11)	86% (83/39)
controller	100% (55/55)	94% (324/342)	75% (1685/2461)
model	100% (13/13)	100% (184/184)	98% (372/373)
repository	100% (0/0)	100% (0/0)	100% (0/0)
security	0% (0/1)	0% (0/1)	0% (0/14)
service	100% (5/5)	98% (50/51)	98% (299/304)
util	100% (1/1)	100% (2/2)	100% (4/4)
ServerApplication	100% (1/1)	50% (1/2)	50% (2/4)

本测试覆盖率很高，可以说是十分到位了。



### 5.2.2 测试环境的影响

任何平台都可以进行测试，并没有什么影响。

### 5.2.3 改进建议

如果可以的话，可以绕出架构对微信的 `websocket` 进行测试。

## 5.3 详细的测试结果

以下是我们的测试覆盖率：

97% classes, 80% lines covered in package 'com.java.asdan.tomato.server'			
Element	Class, %	Method, %	Line, %
configuration	83% (5/6)	63% (7/11)	86% (33/38)
controller	100% (55/55)	94% (324/342)	75% (1885/2481)
model	100% (13/13)	100% (184/184)	99% (372/373)
repository	100% (0/0)	100% (0/0)	100% (0/0)
security	0% (0/1)	0% (0/1)	0% (0/14)
service	100% (5/5)	98% (50/51)	98% (299/304)
util	100% (1/1)	100% (2/2)	100% (4/4)
ServerApplication	100% (1/1)	50% (1/2)	50% (2/4)
83% classes, 86% lines covered in package 'configuration'			
Element	Class, %	Method, %	Line, %
BeanConfiguration	0% (0/1)	0% (0/2)	0% (0/3)
CorsConfigurator	100% (1/1)	100% (1/1)	100% (3/3)
MyWebSocketInt...	100% (1/1)	0% (0/2)	33% (1/3)
SecurityConfigur...	100% (1/1)	100% (2/2)	100% (13/13)
WebSocketConfig	100% (1/1)	100% (2/2)	100% (10/10)
wxWebSocketCo...	100% (1/1)	100% (2/2)	100% (6/6)
100% classes, 75% lines covered in package 'controller'			
Element	Class, %	Method, %	Line, %
admin	100% (24/24)	94% (142/150)	93% (909/971)
client	100% (31/31)	94% (182/192)	64% (956/1490)
100% classes, 93% lines covered in package 'admin'			
Element	Class, %	Method, %	Line, %
AdminLoginCont...	100% (2/2)	100% (4/4)	100% (24/24)
AuctionController	100% (3/3)	93% (14/15)	91% (136/148)
ChangePasswor...	100% (2/2)	85% (6/7)	93% (28/30)
CompetitionCrea...	100% (5/5)	98% (51/52)	99% (298/300)
CompetitionInfor...	100% (5/5)	97% (41/42)	96% (249/259)
CompetitionStat...	100% (3/3)	93% (15/16)	74% (89/119)
DeleteCompetitio...	100% (1/1)	66% (2/3)	93% (29/31)
GetAllCompetitio...	100% (1/1)	66% (2/3)	89% (17/19)
UpdateCompetiti...	100% (2/2)	87% (7/8)	95% (39/41)
100% classes, 64% lines covered in package 'client'			
Element	Class, %	Method, %	Line, %
AvatarController	100% (2/2)	83% (5/6)	59% (28/47)
BuyMaterialCont...	100% (2/2)	100% (24/24)	81% (119/146)
ClientInfoContro...	100% (3/3)	100% (24/24)	76% (87/114)
ClientLoginContro...	100% (2/2)	100% (6/6)	100% (28/28)
ClientPropertyCo...	100% (5/5)	100% (37/37)	96% (198/205)
CompetitionStat...	100% (3/3)	100% (16/16)	92% (48/52)
GetAllUsersCont...	100% (1/1)	100% (1/1)	100% (17/17)
GetProduceHisto...	100% (2/2)	100% (15/15)	100% (60/60)
GetTradeHistory...	100% (2/2)	100% (5/5)	100% (54/54)
ListenPropertyC...	100% (1/1)	100% (4/4)	64% (50/78)
SellMaterialContr...	100% (2/2)	100% (23/23)	84% (90/106)
SendToSellerCon...	100% (2/2)	100% (8/8)	76% (88/115)
UndoTradeContr...	100% (3/3)	100% (12/12)	78% (88/109)
WebSocketPush...	100% (1/1)	18% (2/11)	0% (3/359)
100% classes, 99% lines covered in package 'model'			
Element	Class, %	Method, %	Line, %
Admin	100% (1/1)	100% (17/17)	100% (33/33)
Competition	100% (3/3)	100% (50/50)	100% (104/104)
Machine	100% (1/1)	100% (13/13)	100% (24/24)
Material	100% (1/1)	100% (7/7)	100% (12/12)
Produce	100% (1/1)	100% (17/17)	100% (33/33)
Team	100% (2/2)	100% (49/49)	100% (108/108)
Trade	100% (4/4)	100% (31/31)	98% (58/59)
100% classes, 98% lines covered in package 'service'			
Element	Class, %	Method, %	Line, %
admin	100% (2/2)	100% (26/26)	100% (156/156)
user	100% (3/3)	96% (24/25)	96% (143/148)
100% classes, 100% lines covered in package 'admin'			
Element	Class, %	Method, %	Line, %
CompetitionServiceImpl	100% (1/1)	100% (19/19)	100% (119/119)
LoginServiceImpl	100% (1/1)	100% (7/7)	100% (37/37)

100% classes, 96% lines covered in package 'user'

Element	Class, %	Method, %	Line, %
CompetitionServiceImpl	100% (1/1)	100% (4/4)	100% (22/22)
LoginServiceImpl	100% (1/1)	100% (8/8)	100% (57/57)
OrderServiceImpl	100% (1/1)	92% (12/13)	92% (64/69)

## 5.4 测试记录

本次测试结果生成于 2018 年 1 月 19 日，测试平台是 Mac，集成开发环境是 IntelliJ。见证者是宋世虹和张慧盟。

## 5.5 评价

### 5.5.1 能力

该测试覆盖了几乎所有后台的接口，可以说是很鲁棒，给前端提供了十分稳定的服务。

### 5.5.2 缺陷和限制

由于没有找到微信的 websocket 的测试方法，所以我们没有对微信的 websocket 进行测试，这很遗憾。同时，由于时间的关系，还有极少一部分代码没有被测试到。

## 5.6 测试活动总结

人力消耗：张慧盟、宋世虹。