基于 WebSocket 的立体五子棋

模块 2 选题

Liu, Jiaxing

2020 年 4 月 27 日

目录

1 预览

1.1 介绍

多人多房间的在线五子棋已经部署到腾讯云服务器上。服务的访问地址为 http://49.233.221.184:8080/five/。由于服务器的性能不够稳定，如果助教在检查的时候发现出现无法连接服务器的情况，请通知我重启后台服务，谢谢。

目前多房间五子棋对战的设计为：玩家首先会进入首页，首页中可以选择加入房间还是创建房间。加入房间时如果输入的房间号不存在，系统会提示房间不存在。用户选择创建房间后就会直接进入房间，进入房间后，系统会立即告知玩家房间号。与此同时，页面中房间左侧的控制面板也会展示相应的房间号。目前，系统默认创建房间的人为黑棋。当另外一个人进入房间时，有两种情况：当他是房间中的第三个人的时候，他会作为参与者，系统和控制面板都会提示其身份；如果他是第二个进入房间的人，系统默认他为白棋，并通知房间中的人游戏已经开始。

随着游戏的进行，黑白棋手交替下棋。当出现一方获胜，平局，或者其中一方离开游戏时。系统会根据实际情况，提醒房间中的所有人游戏的进展并告知游戏已经结束，棋手或旁观者可以选择返回首页。

1.2 页面

1.2.1 首页

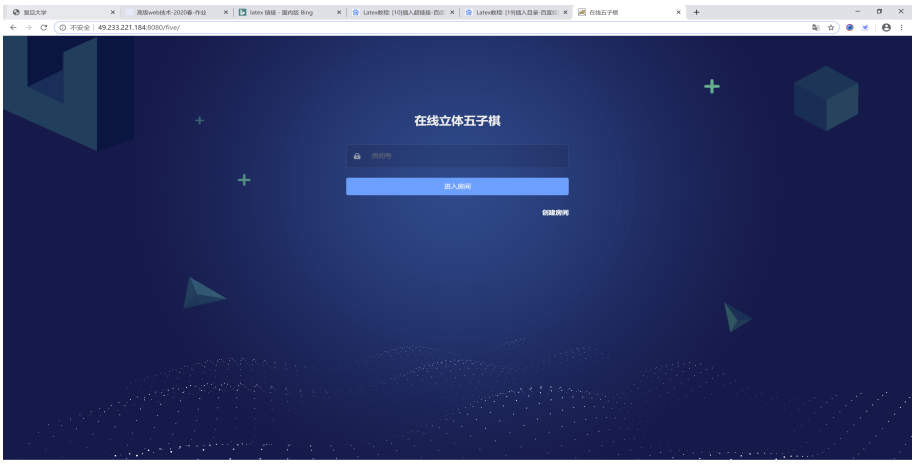


图 1: 首页效果.

首页的设计中利用了 Three.js 技术。在首页的下发，实现了一个波浪粒

子效果。波浪在页面上是一直运动着的。当鼠标在波浪上移动时波浪的形状也会跟随变化。除此以外，首页中间是一个表单，可以用来填入房间号并进入。

1.2.2 房间

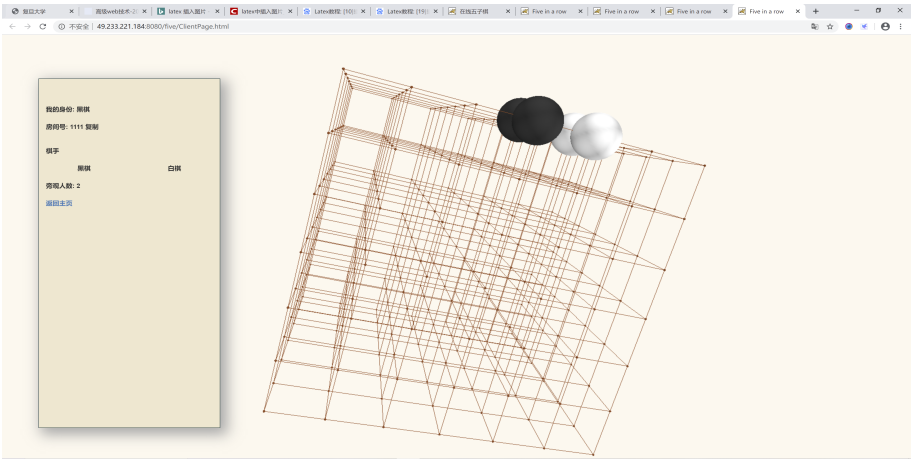


图 2: 房间效果.

房间页面主要分为两部分：控制面板和工作台。控制面板在页面左侧，上面有玩家身份信息，房间号，当前需要落子的棋手信息，旁观人数信息，返回主页的控制链接。

工作台是棋手落子的立体网格。当玩家将鼠标移动到立体网格中的某个点的时候，该点的颜色会变为红色以提示玩家可以在此落子。同时玩家，可以旋转立体网格，缩放立体网格的大小。

2 开发

作业已经提供了在线单房间五子棋的代码。为了实现多房间五子棋对战，开发的重点应该着重在后台的逻辑上。但是由于前端页面的设计，前端页面也花了不少时间。特别是首页的效果。

2.1 后台

2.1.1 设计思路

当房间和多房间的差别在于用户的管理。单房间中所有的用户都属于一个集体，那么多房间就应该根据用户选择的房间号加入用户集体。那么我

们可以设计一个数组来存储所有的房间对象。对于每个房间对象，它自己管理自己的棋盘，玩家，游戏部署，游戏状态。那么这样的一个分组管理的方式就可以实现多房间对战。

2.1.2 监听指令

图 3: 指令设计.

- create: 创建房间指令，当玩家选择创建房间时会执行，执行结果为返回房间号或者提示非法操作（非法指玩家已经在房间中就不执行）。
- enter: 进入房间指令，当玩家选择进入房间时会执行，执行结果为返回房间号或提示非法操作（非法操作同上）。
- play: 下棋指令，当玩家选择在某个位置落子时就会向服务器发送 play 指令。执行结果为通知房间中的玩家棋盘变动情况或者提示该玩家操作非法（非法指此轮不允许玩家落子）。
- check: 房间检查指令，当玩家在首页中选择进入房间时需要检查房间是否存在。这时客户端就会像服务器发送 check 指令，传入房间号，检查房间是否存在。执行结果是告知房间存在与否信息。

2.1.3 创建房间

当服务器监听到 create 指令时，首先根据服务器客户端的连接对象判断玩家是否已经在某个房间中。如果已经在某个房间中就拒绝创建房间操作。否则，后台开始执行创建房间工作。

创建房间工作分为两部分：初始化房间参数和将初始化后的房间推入房间数据库。初始化房间参数包括：通过已有的房间名信息 zoomNames 生成

```
function createZoom(zooms, client, zoomNames){
    let zoomName = getZoomName(zoomNames);
    let board=new Array();
    for (var i=0;i<=6;i++){
        board[i]=new Array();
        for (var j=0;j<=6;j++)
            board[i][j]=new Array();
    }
    boardInit(board);
    let zoom = {
        name: zoomName,
        clients: [client],
        board: board,
        go: 0, // 0表示改黑棋走，1表示改白棋走
        steps: 0,
        over: 0
    }
    zooms.push(zoom);
    zoomNames.push(zoomName);
    client.zoomName = zoomName;
    return zoom;
}
```

图 4: 创建房间.

一个随机并且与已有房间不重合的房间号。初始化棋盘为空棋盘对象。初始化房间的用户为只有创建者的数组。初始化游戏的步数、状态、玩家先手信息。新的房间对象初始化之后，就加入到房间数据库中 zooms.

2.1.4 加入房间

```
function addClient(zoomName, client, zooms){
    for(let i = 0; i < zooms.length; i++){
        if(zooms[i].name === zoomName){
            client.zoomName = zoomName;
            zooms[i].clients.push(client);
            return zooms[i];
        }
    }
    return false;
}
```

图 5: 加入房间 1.

相比于创建房间，加入房间相对简单。当服务器收到进入 enter 指令时，首先通过房间名信息检查房间是否存在。如果不存在，拒绝执行指令。否则，开始执行加入房间的工作。加入房间分为两步：先根据房间信息找到对应房间，然后将当前客户端加入该房间的客户端集体中。遍历房间数据库

zooms，通过房间名比较的方式可以找到相应的房间。然后将当前客户端推入房间的 clients 中。由于系统的设计，当房间的人数为 2 时，游戏会自动开始。所以当执行完进入房间后，会立即检查当前房间的客户端人数。如果恰好为两人，就会分别通知游戏已经开始并告知其身份。如果为第三人以及其他人加入，会告知加入者身份，通知房间中所有人当前房间的人数和棋盘信息。

```
case 'enter':
    if(client.zoomName){
        mySendUTF(client, 'ctrl', config.codes.allreadyInZoom);
        break;
    }
    console.log((new Date())+" Zoom : " + json.data + " Enter Zoom.");

    if(json.data === undefined || !zoomNames.includes(json.data)){
        mySendUTF(client, 'ctrl', config.codes.zoomNotFound);
        break;
    }
    let ret = addClient(json.data, client, zooms);
    if(!ret || ret.over){
        mySendUTF(client, 'ctrl', config.codes.zoomNotFound);
        break;
    }
    if(ret.clients.length >= 3){   // 第三个人加入的话，就直接告诉他棋盘
        mySendUTF(client,'ctrl', config.codes.view);
        mySendUTF(client, 'board', ret.board);
    } else if(ret.clients.length === 2){
        mySendUTF(ret.clients[0],'ctrl', config.codes.go);
        mySendUTF(ret.clients[0],'ctrl', config.codes.black);
        mySendUTF(ret.clients[1],'ctrl', config.codes.white);
    }
    for(let i = 0; i < ret.clients.length; i++){
        mySendUTF(ret.clients[i], 'number', ret.clients.length);
    }
    break;
```

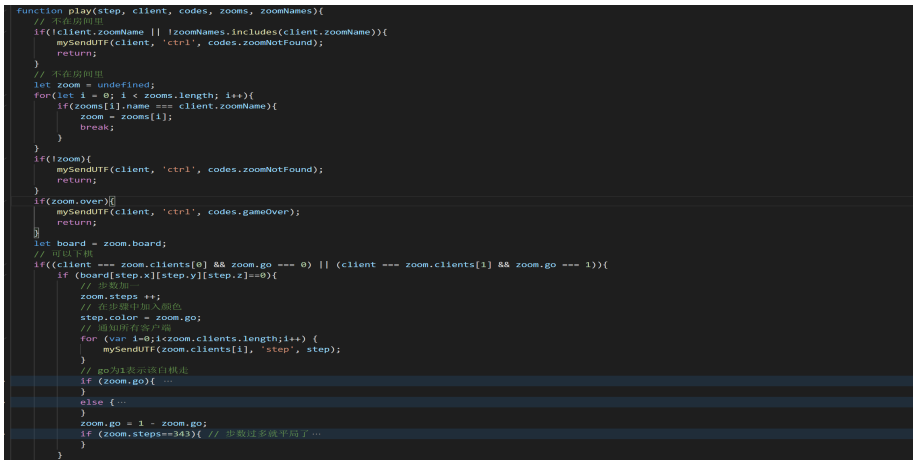
图 6: 加入房间 2.

2.1.5 检查房间

当用户选择进入房间时，服务器需要检查房间是否存在。为了提高检查的效率，服务器维护了一个数组用来管理已有的房间号。利用这个数据，房间是否存在的检查更加高效。

2.1.6 玩家对弈

玩家对弈部分的代码实现包括：房间是否存在检查，落子是否有效检查，落子有效后通知房间内的所有人员棋盘变动信息，根据落子后的棋盘分析游戏是否出现新的状态（比如一方胜利，平局）并通知。其中最重要的游戏状态检查，但这部分已经预先给出的示例代码中完全实现了。而分组的精髓也在这里有所体现：当玩家落子后，只通知在同一个房间的客户端棋盘变动信息。



```
function play(step, client, codes, zooms, zoomNames){
    // 不在房间里
    if(!client.zoomName || !zoomNames.includes(client.zoomName)){
        mySendUTF(client, 'ctrl', codes.zoomNotFound);
        return;
    }
    // 不在房间里
    let zoom = undefined;
    for(let i = 0; i < zooms.length; i++){
        if(zooms[i].name === client.zoomName){
            zoom = zooms[i];
            break;
        }
    }
    if(!zoom){
        mySendUTF(client, 'ctrl', codes.zoomNotFound);
        return;
    }
    if(zoom.over){
        mySendUTF(client, 'ctrl', codes.gameOver);
        return;
    }
    let board = zoom.board;
    // 可以下棋
    if((client === zoom.clients[0] && zoom.go === 0) || (client === zoom.clients[1] && zoom.go === 1)){
        if (board[step.x][step.y][step.z]==0){
            // 下棋加一
            zoom.steps ++;
            // 在步骤中加入颜色
            step.color = zoom.go;
            // 通知所有客户端
            for (var i=0;i<zoom.clients.length;i++) {
                mySendUTF(zoom.clients[i], 'step', step);
            }
            // go为1表示该白棋走
            if (zoom.go){ …
            }
            else {…
            }
            zoom.go = 1 - zoom.go;
            if (zoom.steps==343){ // 步数过多就平局了…
            }
        }
```

图 7: 玩家对弈.

```
client.connection.on('close',function(connection){

    if(!client.zoomName || !zoomNames.includes(client.zoomName)){
        // 房间不存在的话不做处理
        return;
    }
    let zoom = undefined;
    let zoomIndex = undefined;
    for(let i = 0; i < zooms.length; i++){…
    }
    if(!zoom){…
    }
    let index = undefined;
    for(let i = 0; i < zoom.clients.length; i++){…
    }
    if(index === undefined){…
    }
    for(let i = 0; i < zoom.clients.length; i++){
        mySendUTF(zoom.clients[i], 'number', zoom.clients.length);  // 通知总人数
    }

    let ctrl;
    if (index<2) {
        zoom.over = 1;
        if (zoom.clients.length==0){
            console.log((new Date())+" Zoom : " + zoom.name  +" Empty now .");
            zoomNames.splice(zoomIndex,1);
            zooms.splice(zoomIndex,1);
            return;
        }
        if (index==0){
            console.log((new Date())+" Zoom : " + zoom.name  +" Black gets out. There is "+zoom.clients.length+" people totally.");
            ctrl=4;
        } else {
            console.log((new Date())+" Zoom : " + zoom.name  +" White gets out. There is "+zoom.clients.length+" people totally.");
            ctrl=5;
        }
        for (var i=0;i<zoom.clients.length;i++){
            mySendUTF(zoom.clients[i],'ctrl',ctrl);
        }
    }else{
        console.log((new Date())+" Zoom : " + zoom.name  +" A viewer gets out . There is "+zoom.clients.length+" people totally .");
    }
});
```

图 8: 玩家离线.

2.1.7 玩家离线

在已有的代码中，玩家离线的处理相对简单。对于黑棋和白棋选手的离线，才会特别处理。对于其他选手，基本上不处理。而在现在的实现中，首先检查房间是否存在，不存在不进行处理。当房间存在时，执行断开连接操作。当房间中还有人的时候，就会通知他们房间中的总人数。否则，释放房间的内存资源，删除房间信息，这意味着这个房间将不存在了。特别的，对于黑白棋手的离开，系统将会提示所有人员他们的离开。

2.2 前端

2.2.1 首页

```
var ws; var wsState; var zoomName; var checking = false;
function initWS(){
    ws =new WebSocket('ws://49.233.221.184:8081');
    ws.onopen=function(){
        checking = false;
    };
    ws.onerror=function(error){
        checking = false;
        showMessage("Can't connect to the server .", 0, 2000,1);
    };
    ws.onmessage=function(message){
            var json=JSON.parse(message.data);
            checking = false;
            console.log(json);
            if (json.type=='ctrl'){
                switch(json.data){
                    case -5:
                      showMessage('不存在该房间',0,3000);
                      break;
                    case 9:
                      setCookie('zoomOrder',JSON.stringify({order:'enter',data: zoomName}));
                      window.open("./ClientPage.html");
                        break;
                }
            }
        }
}
initWS();
```

图 9: 首页 WebSocket.

首页功能主要是当玩家选择进入房间时检查房间是否存在。所以需要预先建立 WebSocket。建立好 WebScocket 后设定监听函数。监听的信息主要有两种，分别对应房间是否存在两种情况。

当玩家选择进入房间时，客户端发送检查指令。如果反馈信息为房间存在，则会在 cookie 中存入进入房间的指令与房间号信息，跳转到房间页。如果房间不存在，则直接在页面中进行提示，不跳转。当玩家选择创建房间时，会将指令存入 cookie，跳转到房间页。

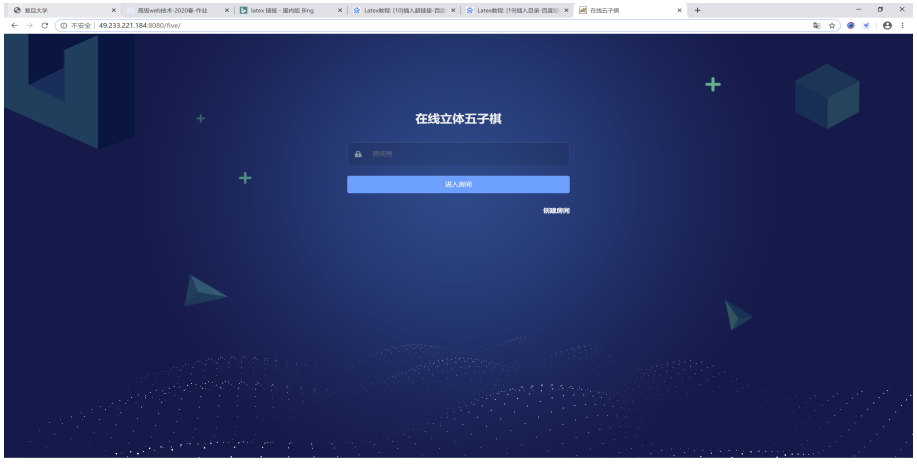


图 10: 粒子波浪效果.

首页粒子波浪效果是利用 Three.js 实现的，其中涉及到每一次 render 所有粒子位置的计算，粒子的渲染等内容。在学习资料中我会给出学习的链接，这里不再赘述。

2.2.2 房间页

```
var connection=new WebSocket('ws://49.233.221.184:8081');
connection.onopen=function(){
    if(getCookie('zoomOrder') == null){
        showMessage('无法初始化房间，请返回大厅', 0, 2000, 1);
        return;
    }
    let str = getCookie("zoomOrder");
    try {
        if (typeof JSON.parse(str) !== "object") {
            showMessage('无法初始化房间，请返回大厅', 0, 2000, 1);
            return;
        }
    } catch(e) {
        showMessage('无法初始化房间，请返回大厅', 0, 2000, 1);
            return;
    }
    let json = JSON.parse(str);
    if(!json.order){
        showMessage('无法初始化房间，请返回大厅', 0, 2000, 1);
        return;
    }
    if(json.order === 'create'){ // 创建房价
        connection.send(JSON.stringify({order:'create'}));
    } else if(json.order === 'enter' && json.data){ // 进入房间
        zoom = json.data;
        connection.send(JSON.stringify({order: 'enter', data: json.data }))
        setZoom(zoom);
    }
};
connection.onerror=function(error){
    showMessage("Can't connect to the server .", 0, 2000,1);
};
```

图 11: 房间页 WS.

在谈到首页时提到，首页会根据用户的操着将指令存入 cookie 中。那么房间页就是提取 cookie 执行相应操作。在打开 Websocket 连接时会判断你是否存在相应的 cookie，如果不存在告知无法初始化房间。否则执行相应操作，如告知服务器创建房间，或告知服务器进入房间。

客户端作为 WebSocket 的连接方，需要监听多种消息。其中大部分消息都在预先给的代码中有了，这里根据功能的需要加入了关于房间人数的通知，关于创建房间、进入房间结果的通知，关于该谁下棋的通知。

2.2.3 源代码改进

关于源代码的实现，我做了如下改进：

- 棋子位置通信：原实现为传递棋子位置信息和棋子颜色信息，考虑到安全性，现在不再需要棋子的颜色信息。服务器可以通过当前客户端识别用户身份，不需要传递自己的身份。
- 落子顺序通信：原实现为当前用户判断自己的身份，根据通知的最近的棋子的颜色来修改自己是否可以下棋的权限。现在修改为服务器通知某个用户可以下棋，不用客户端自己判断。

- 五子棋工作台 UI 部分：当玩家将鼠标移动到某个网格点时，原实现为网格点颜色变为深黄色。考虑到实现效果，这里换为深红色并添加了放射光。

3 部署

3.1 环境配置

环境配置分为两部分：Web 服务器的配置和 node 环境的配置。Web 服务器的环境配置，这里选择的是 Tomcat。关于 Tomcat 的配置和 node 的配置我的放到了学习资料中。

3.2 前后端部署

环境配置好之后，我们就可以运行了。首先，将前端页面放到 tomcat 下的 webapps 下，这里我们先创建了一个目录为 five，然后将前端代码放入其中。如图 12，防止好位置后，就可以通过 ip:port/five 来访问前端页面了。但现在页面上会出现无法连接到服务器的提示，我们需要启动后台服务。

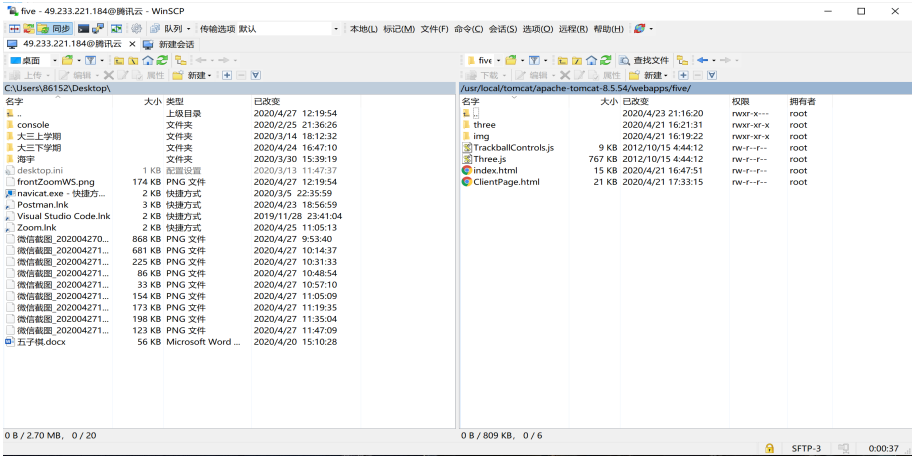


图 12: 服务器上的前端文件位置.

由于 tomcat 占据了 8080 端口，我们先修改 ws 监听端口为 8081。由于我们在服务器中开放了所有的入站安全组，所有无需配置安全组（图 13 为服务器安全组信息）。

同样的我们可以将后台代码放到服务器的某个目录下（后台服务无特定位置限制），然后在 Server.js 同级目录通过指令 node ./Sever.js 启动后台

服务。为了能够实现断开 ssh 连接服务不挂断，启动方式改为 nohup node ./Server.js &。

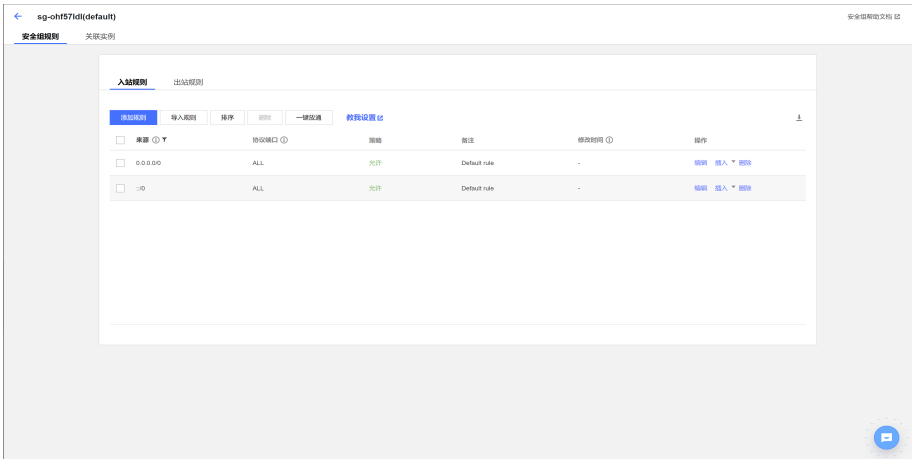


图 13: 云服务器安全组.

4 工具与学习资料

4.1 工具

在项目的部署过程中，用到了几个工具:

- Xshell: 远程登录服务器工具，可以用来执行终端脚本和配置环境。
- WinScp：可视化远程文件服务。比如将前后端代码文件拖入服务器中，将 node 安装包压缩文件拖入服务其中。
- Vscode: 轻量级编程工具。

4.2 学习资料

这里提供三个学习资料：

- Tomcat 安装和配置: https://blog.csdn.net/wqh0830/article/details/86703992
- Node 安装和配置:https://blog.csdn.net/qq_21794603/article/details/68067821
- Three.js 实现粒子波动效果:https://www.jianshu.com/p/38fe115db754