高级 Web Project——虚拟 3D 画展文档

项目部署与运行

安装项目依赖

```
npm install
```

以 docker 方式启动

```
docker-compose up
```

- 启动后默认访问地址为: http://localhost:80/
- 端口可在 docker-compose.yml 文件的 nginx 部分修改）

以 docker 后台方式启动

```
docker-compose up -d
```

终止 docker 容器

```
docker-compose down
```

编译前端静态文件(webpack)

```
npm start
```

开启测试服务器(webpack-dev-server)

```
npm run server
```

服务器地址：http://localhost:8080/gallery/

开启 WebSocket 服务器（热部署）

```
npm run socket-dev
```

开启 WebSocket 服务器

```
npm run socket
```

说明

为方便解释，此文档中的代码均经过简化或转换为伪代码，详细代码实现请参照完整代码

小组分工

阮家炜	PJ 模块化平台搭建，WebSocket 后端编写，灯光配置，人物模型动画编写，服务器部署

毛浩楠	UI 层编写，利用 ajax 与后端交互，“画中”世界编写（包括：巨幅画布，NPC，鼠标监听事件的监听，地面评论标志）
杨范	“大厅”世界编写（包括：鼠标监听事件的监听，三个房间的搭建、路牌的搭建），
王锴	数据库设计，定义相关接口，SpringBoot 后端编写（包括：实体类编写、Service 层编写、Controller 层编写）

Three.js 模块化

基于 Webpack 打包

为方便部署，本项目采用了 webpack 来打包所有的组件，顺便实现了模块化

Component 组件

```
class Component{
    onCreate(){ /*当组件被创建时执行*/ }
    onUnmount(){ /* 当组件被从父组件删除时执行 */}
    onRender(deltaTime){ /* 每帧渲染 */ }
    onAwake(){ /* 切换到组件所在世界时执行 */ }
    onSuspend(){ /* 切换到非组件所在世界时执行 */ }
    use(component){ /* 添加子组件 */ }
    getObject(){ /* 获取封装的原生 Object3D */ }
    setObject(object){ /* 设置封装的原生 Object3D */ }
}
```

Component 包含五个主要生命周期，相关介绍已经写在注释中

World 根组件

```
class World extends Component{
  constructor(){
    this.camera = new THREE.PerspectiveCamera(72, window.innerWidth /
    window.innerHeight , 1.0, 1000.0);
    this.scene = new THREE.Scene();
    this.setObject(this.scene);
  }
}
```

World 可使用 use 方法将组件加入到 scene 中，从而生成一个世界

Framework 框架类

```
class Framework{
  constructor(dom){
    this._dom = dom;
    this._world = null;
    this._renderer = new THREE.WebGLRenderer({antialias:true});
  }
  getWorld(){}
  setWorld(world){}
  run(){}
```

Framework 的 run 方法会通过 requestAnimationFrame()注册渲染事件，每一帧用 renderer 渲染 setWorld 所提供的世界对象（世界对象包含 scene 与 camera，即渲染必须提供的参数），如下方代码所示：

```
run(){
  //注册渲染函数
  requestAnimationFrame((() => {
    let prevStamp;
    let render = (timeStamp) => {
      let deltaTime = prevStamp ? (timeStamp - prevStamp) : 0;
      if(this._world){
        this._renderer.render(this._world.scene, this._world.camera);
        this._world.onRender(deltaTime / 1000.0);
      }
      prevStamp = timeStamp;
      requestAnimationFrame(render);
    };
    return render;
  })());
}
```

依赖注入

```
setWorld(world){
  if(this._world === world)
    return;

  if(!this._historyWorlds.includes(world)){
    // 第一次载入该世界，进行首次依赖注入
    world.$dom = this._dom;
    world.$framework = this;
    world.$client = this._client;
    world.$ui = this._ui;

    world.onCreate();
    this._historyWorlds.push(world);
  }
  this._world = world;
}
```

在第一次载入世界的时候，会进行依赖注入，将 framework 自身与一些需要与组件交互的对象传入 world，这样在 world 及其子组件在用 use 添加子组件的时候，会自动将依赖传递下去，请看 Compnent 类的 use 方法

```
use(component){
  //依赖注入
  if(component instanceof Component){
    component.$dom = this.$dom;
    component.$framework = this.$framework;
    component.$ui = this.$ui;
    component.$world = this.$world;
    component.$client = this.$client;

    component.$parent = this;

    //调用 onCreate 生命周期
    component.onCreate();
    this.$components.push(component);

    // 检测自身与要添加的组件是否都是实体组件
    if(this.getObject() instanceof THREE.Object3D && component.getObject()
    instanceof THREE.Object3D){
        this.getObject().add(component.getObject());
    }
  }
}
```

SocketIO

本项目包含两个后端，一个是 SpringBoot 后端，用于存储用户信息、用户留下的评论、图画信息以及 NPC 数据等。另一个是 SocketIO 后端，用来存储用户行走的位置、发送的弹幕。下面主要介绍 SocketIO 部分

SocketClient

```
class SocketClient{
   constructor(){
      this._io = io(config.server.root);
      this.on = this._io.on.bind(this._io);
      this.emit = this._io.emit.bind(this._io);
   }
}
```

SocketServer

```
class SocketServer{
    start(){ /* 开始监听 */ }
    onConnection(client){ /* 客户端连接时回调 */ }
    onDisconnect(client){ /* 客户端断开时回调 */ }
}
```

SocketServer 不作为前端打包内容组件之一，而是作为一个单独的组件来运行

DataSender

在 FirstPersonController 加入 DataSender 组件，利用 setInterval 向服务器发送位置数据：

```
onSyncMovement(){
       if(this.active){
          this.$client.emit('move', {
              paintingId: this.paintingId,
              position: {
                  x: this.$parent.getObject().position.x,
                  y: this.$parent.getObject().position.y,
                  z: this.$parent.getObject().position.z
              },
              rotation: {
                  x: 0,
                  y: this.$parent.yawObject.rotation.y,
                  z: 0
              }
          });
       }
}
```

人物移动平滑处理

本项目中人物平滑移动参照了 Unity3D 中的 Lerp 操作，在 Three.js 官方文档的 Vector3 部分有这样的描述：

.lerp (v : Vector3, alpha : Float) : this

v - 朝着进行插值的Vector3。
alpha - 插值因数，其范围在[0, 1]闭区间。

在该向量与传入的向量v之间的线性插值，alpha是沿着线的距离长度。 —— alpha = 0 时表示的是当前向量，alpha = 1 时表示的是所传入的向量v。

因此只需要在 Player 的 onRender 方法中（相当于每帧执行）取对应模型自身的位置与服务器传回的同步位置之间的插值向量作为玩家的下个位置即可

首先服务器传回同步位置时，调用 moveTo 方法，将位置保存在 target 中

```
moveTo(position, rotation){
       this.target.position.set(position.x, position.y, position.z);
       this.target.rotation.set(rotation.x, rotation.y, rotation.z);
}
```

再重写 onRender 方法

```
onRender(deltaTime){
       super.onRender(deltaTime);
       this.getObject().position.lerp(this.target.position, 15.0 * deltaTime);
       this.getObject().rotation.set(this.target.rotation.x, this.target.rotation.y,
this.target.rotation.z)
}
```

模型动画加载

先介绍一个术语：蒙皮，三维动画术语，也用于 3D 游戏中。三维动画的一种制作技术。在三维软件中创建的模型基础上，为模型添加骨骼。由于骨骼与模型是相互独立的，为了让骨骼驱动模型产生合理的运动。把模型绑定到骨骼上的技术叫做蒙皮。

也就是说对于同一套骨骼，绑定不同的“蒙皮”，人物就可以显示不同的外表。绑定不同的骨骼动画，人物就会做出不同的动作。下面分享一下 three.js 中加载模型与动画的过程：

```
const loader = new THREE.FBXLoader();

loader.load('蒙皮骨骼.fbx', function(mesh){
  const mixer = new THREE.AnimationMixer();
  mesh.mixer = mixer;

  scene.mixers.push(mixer);
  scene.add(mesh);

  loader.load('骨骼动画.fbx', function(anim){
    const action = mixer.clipAction(anim.animations[0]);
    action.play();
  });
});
```

注意：有时候蒙皮骨骼与动画是在同一个模型文件里的，这时候就可以简化代码：

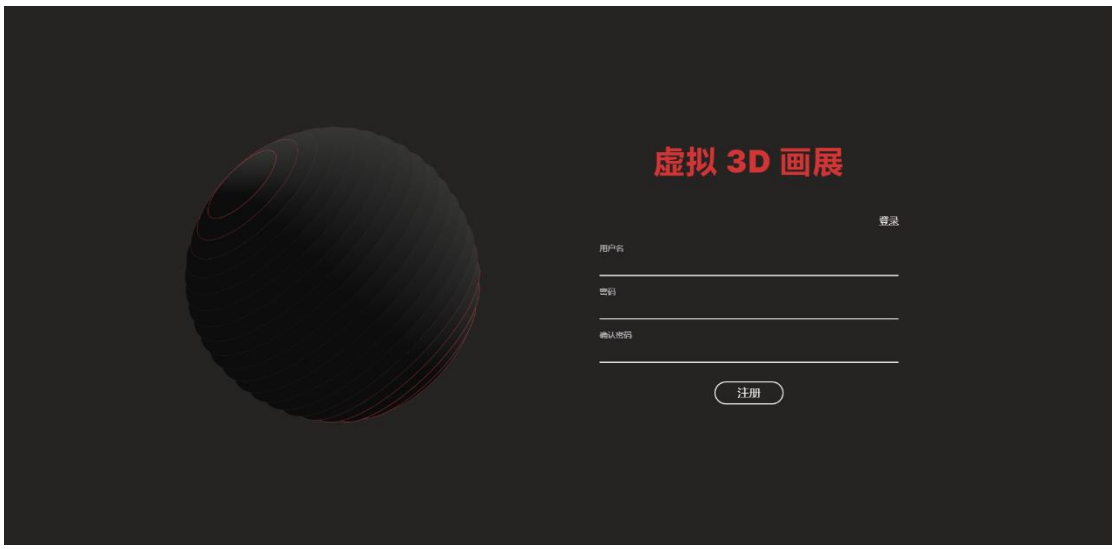
```
const loader = new THREE.FBXLoader();

loader.load('蒙皮骨骼.fbx', function(mesh){
  const mixer = new THREE.AnimationMixer();
  mesh.mixer = mixer;

  scene.mixers.push(mixer);
  scene.add(mesh);

  const action = mixer.clipAction(mesh.animations[0]);
  action.play();
});
```

登录注册页面

登录

登录页面：输入用户名与密码即可登录。
实现的功能：输入框失焦时，判断输入框是否有内容，若没有，则提示输入框为空

注册



注册页面：输入用户名、密码与确认密码即可登录。
实现的功能：输入框失焦时，判断输入框是否有内容，若没有，则提示输入框为空
特殊功能：左边用到的波动动画，使用 anime.js 动画框架实现。

RoomWorld 画中世界

控制器 FirstPersonController

第一人称控制器

NPC

房间内介绍画背景知识的非玩家角色，可以轮播介绍的内容

CommentSender

玩家发送评论功能发起器
监听键盘事件，弹出发送评论 UI 组件

BarrageSender

玩家发送弹幕功能发起器
监听键盘事件，弹出发送弹幕 UI 组件

UserInfoSender

玩家查看个人信息功能发起器
监听键盘事件，弹出个人信息 UI 组件

环境组件

Comment

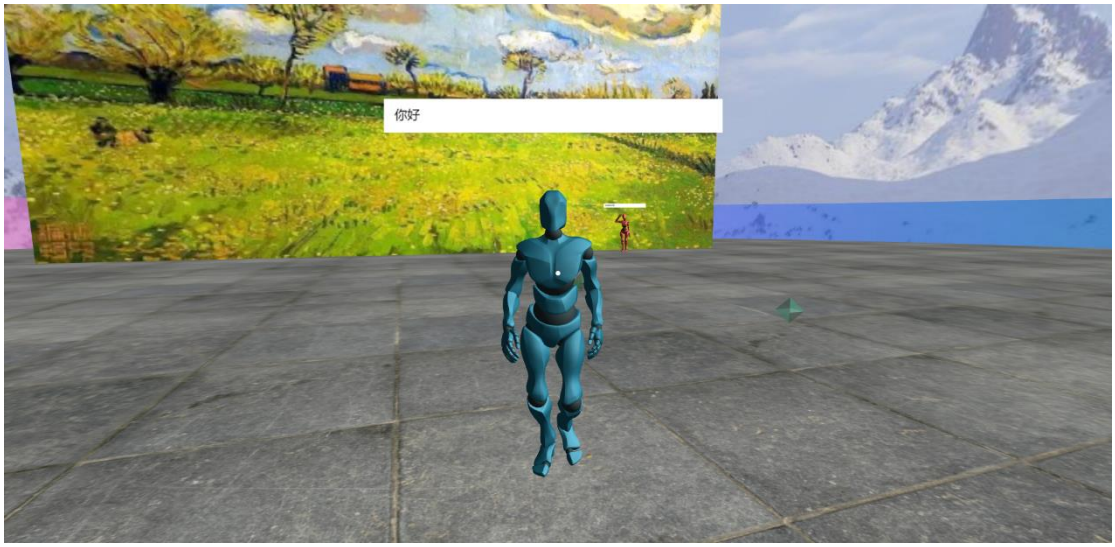
评论标记
记录玩家评论，当玩家发表评论时，会获取玩家实时位置，转角信息，在该玩家的位置生成一个评论标记给所有玩家浏览

Canvas

画布
RoomWorld 内的核心组件，展示名画的高清大图

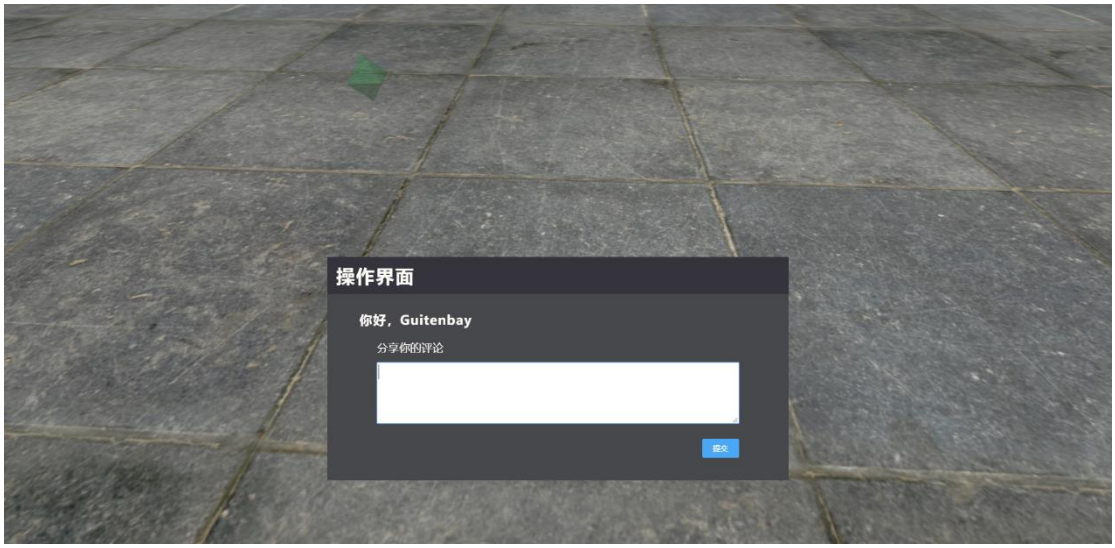
弹幕功能

鼠标聚焦情况下，按键盘 Enter 键可以弹出发送弹幕界面，在其中输入弹幕，点击下方提交按钮即可发送弹幕。发送完成后，界面消失，对面可以看到己方头上弹出弹幕框。
实现的功能：提交评论前，判断输入框中是否有内容，若没有，则提示评论为空。



评论功能

分享评论



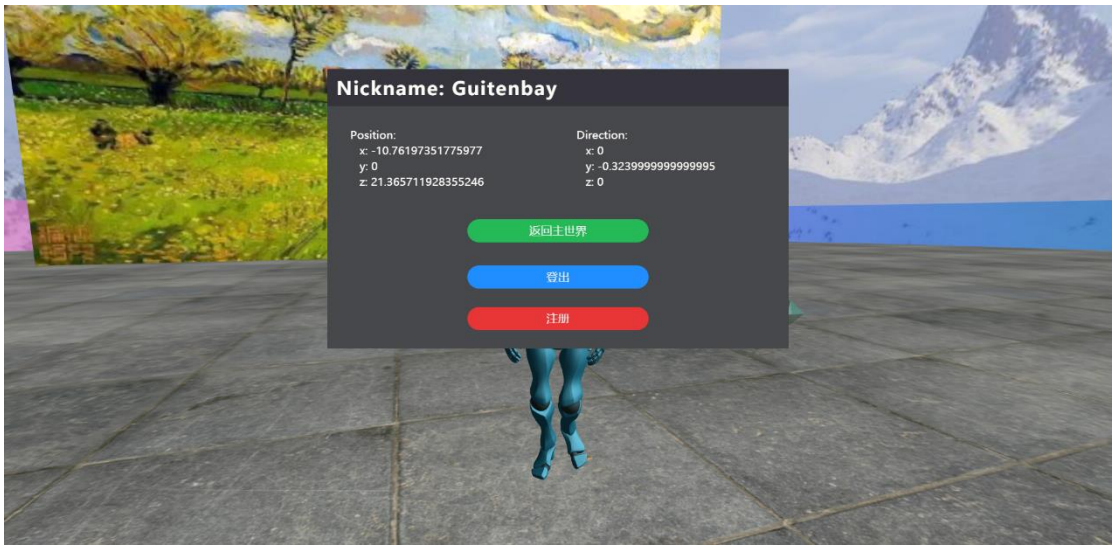
鼠标聚焦情况下，按键盘 E 键可以弹出分享评论界面，在其中输入评论，点击下方提交按钮即可分享评论。分享完成后，界面消失，提示分享成功。
实现的功能：提交评论前，判断文本框中是否有内容，若没有，则提示评论为空。

查看评论

当第一人称控制器从中心射出的 RayCaster 射线触碰到评论标记时，评论标记会变亮。此时点击鼠标左键，可以弹出查看评论界面。

之后按需键盘 Q 键或鼠标点击空白处即可退出查看评论界面。

个人信息功能



鼠标聚焦情况下，按键盘 P 键可以弹出个人信息界面，在此界面可以查看个人昵称，目前位置信息，还有返回主世界按钮、登出按钮和注册按钮。
鼠标点击空白处即可退出个人信息界面。

UI 层

实现 UI 层

1. 在 index.html 上添加 UI 层，利用绝对定位覆盖在渲染 3D 模型的 root 之上。

```
<div id="root" tabindex="0"></div>
<div id="ui"></div>
```

PS: tabindex 全局属性指示其元素是否可以聚焦，以及它是否/在何处参与顺序键盘导航（通常使用 Tab 键，因此得名）。

2. 在 Framework.js 上为 index.html 中的 div#ui 添加 UIRoot 组件

```
class Framework{
    constructor(dom){
        this._ui = <UIRoot />;
        ReactDOM.render(this._ui, document.getElementById("ui"));
        this._ui = UIRoot.instance;
        // this._ui 中需要调用 framework 中的函数，于是将其传入 ui
        this._ui.$framework = this;
    }
}
```

使用 Webpack 打包工具

UI 层组件基于 React 框架。于是本项目使用 Webpack 打包工具整合 React 与 Three.js。webpack.config.js 代码

```
module.exports = {
  module: {
    rules: [
        {
            test: /\.css$/,
            use: [
                { loader: "style-loader" },
                { loader: "css-loader",
                  options: { modules: true } }
            ]
        },
        {
            test: /\.(js|jsx)$/,
            exclude: /node_modules/,
            loader: "babel-loader"
        }
    ]
  }
};
```

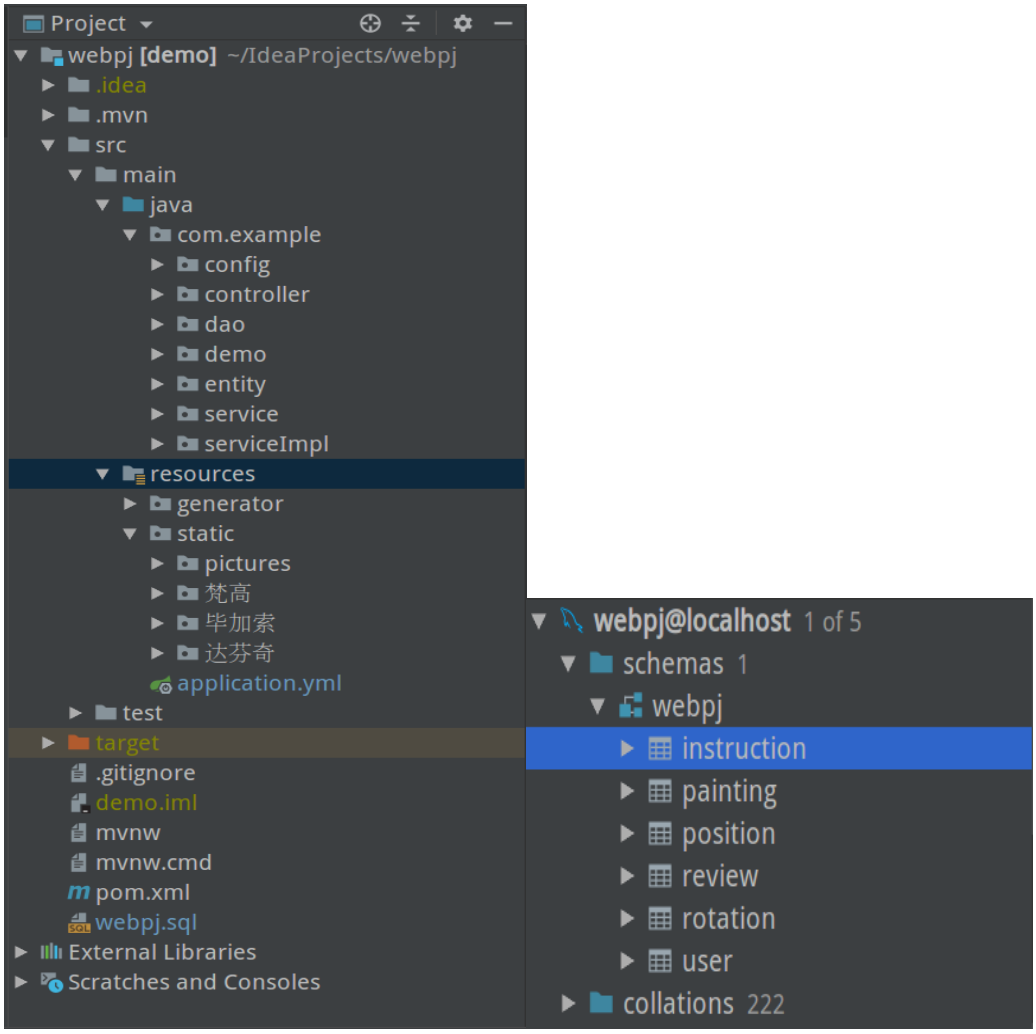
房间搭建（ThreeBSP）

在搭建房间时，需要在墙上面挖门。使用 ThreeBSP 库进行切割。这个方法需要注意的问题是：两个的位置要设置在同一位置，这样，才会将重合的部分挖空

```
let door = new THREE.BoxGeometry(10, 20, 0);
let doors = new THREE.Mesh(door);
doors.rotation.y = Math.PI / 2;
doors.translateZ(-40);
doors.translateY(10);
let meshH4Door = new ThreeBSP(doors);
let meshH4Wall = new ThreeBSP(b);
let resultBSP = meshH4Wall.subtract(meshH4Door);
b = resultBSP.toMesh();
```

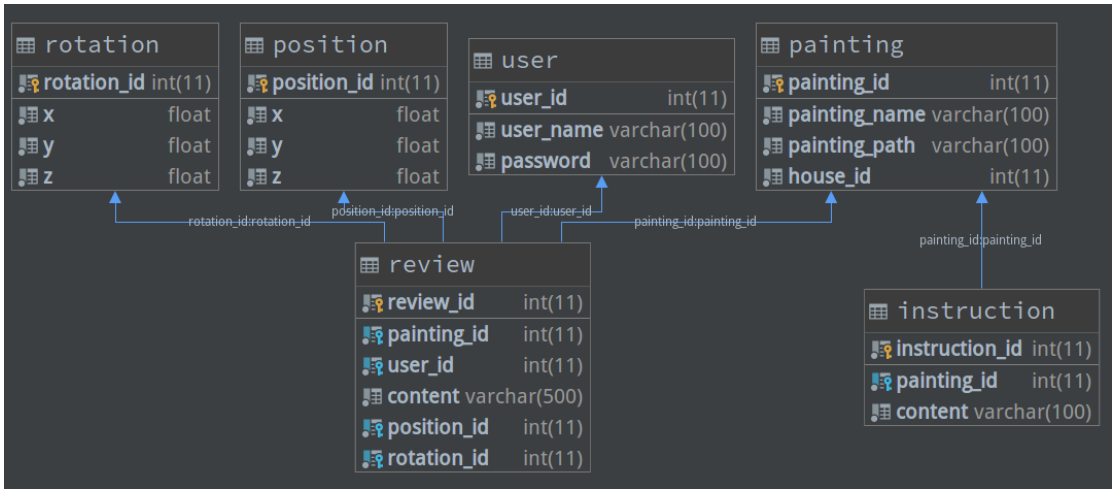
上面是 ThreeBSP 使用时的核心代码，将两个 mesh 对象作为参数传入 ThreeBSP 的构造函数，构造 ThreeBSP 的对象，然后两个对象做 subtract 操作，最后，将操作结果转为 mesh

Spring Boot 后端

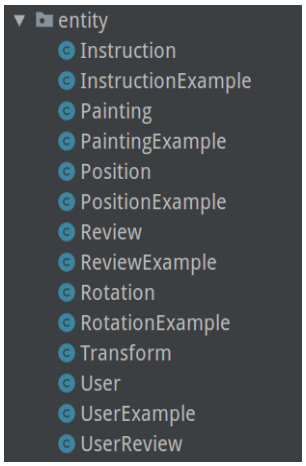


数据库（mybatis）

根据需要建立了 6 张表格，用于存储用户信息，图片信息以及评论信息



entity

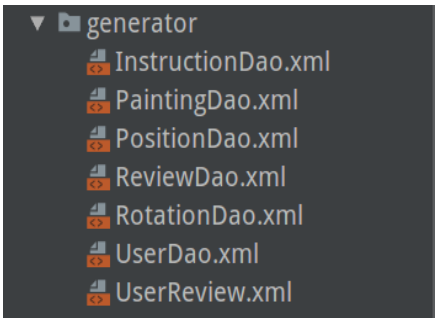


根据数据库中的表格生成了相应的实体类，里面的 Example 文件是专门用来进行单表查询，非常好用。

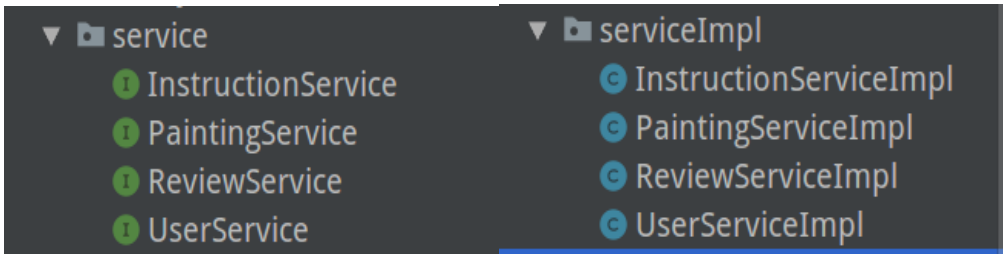
```
UserExample userExample = new UserExample();

userExample.createCriteria().andUserNameEqualTo(userRegister.getUsername());

List<User> userSelects = userDao.selectByExample(userExample);
```

数据库访问

实现了相应的 dao 文件和 xml 文件用于数据库访问。



service 层



controller 层

实现了 url 的匹配和转发。

```
@Autowired
private UserServiceImpl userService;
@RequestMapping(value = "/api/users",method = RequestMethod.POST,produces =
"application/json;charset=UTF-8")
@ResponseBody
public String register(@RequestBody User user){
   return userService.register(user);
}
```

最后实现了如下一些功能

一些细节说明

配置文件的获取

在 application.yml 中添加了如下字段，表示后端运行在哪个地址中，可以简便得进行修改获取图片的地址。

```
polarwk:
  remote-address: http://47.102.212.146:8080/
```

创建 properties.java，来读取配置文件。添加下面注释可以获取配置文件内容。

```
@Component
@ConfigurationProperties(prefix = "polarwk")
public class PolarwkProperties {
    private String remoteAddress;
    public String getRemoteAddress() {
        return remoteAddress;
    }
    public void setRemoteAddress(String remoteAddress) {
        this.remoteAddress = remoteAddress;
    }
}
```

注册登陆问题

注册的时候将用户名进行了单向 hash 加盐操作，保证了用户信息的安全性。
用户登录后使用 session 存储用户相应的信息

更新评论

因为 review 的表里，存储了 rotate 和 position 表的 id 作为外键，所以要先更新评论的位置信息，然后获得相应的 id，再更新 review 表。

```
//更新 position 数据表
Position positionUpdate = new Position();
positionUpdate.setX(position.getFloat("x"));
positionUpdate.setY(position.getFloat("y"));
positionUpdate.setZ(position.getFloat("z"));
positionDao.insert(positionUpdate);
int positionIndex = positionUpdate.getPositionId();
```

返回 json 格式

使用了 alibaba 的 fastjson，可以很方便的把一些对象链表、对象数组转换成 json 嵌套格式。而且不用处理 exception。