WEB在线虚拟博物馆

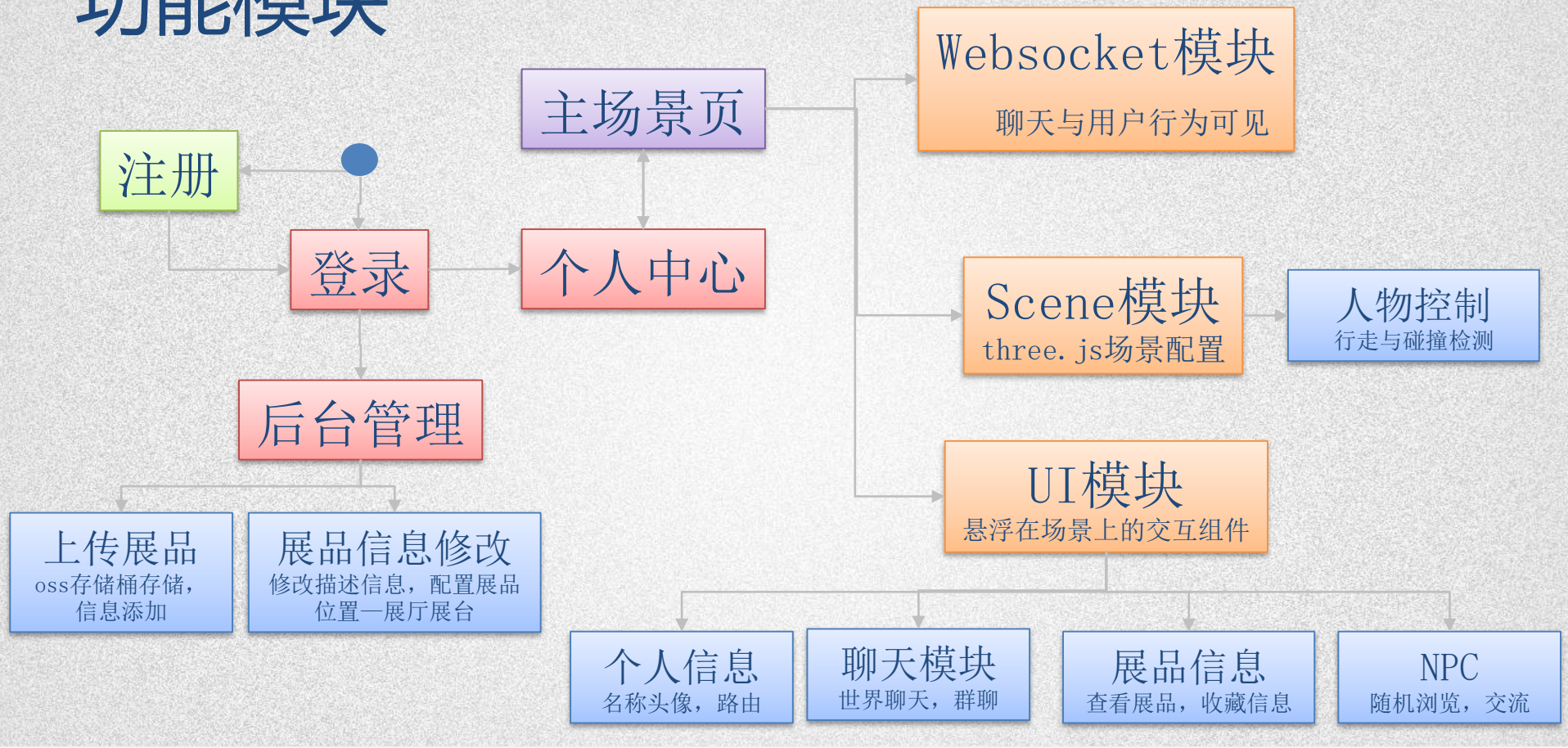
曹佳颖　庄天熠
吴茜雅　杜小林

项目技术架构

模块	技术栈
前端框架	Vue.js + three.js
后端框架	SpringBoot
Websocket	Stomp
数据库	Mysql+redis
部署环境	阿里云ECS（centos7）
资源存储	OSS对象存储

功能模块

展示Vue项目结构



项目开发流程

展示提交记录

第一阶段

- 登录注册页 ---zty
- 个人信息页 ---cjy
- 主场景搭建 ---dxl
- 后端项目框架搭建，接口支持 ---wxy

第二阶段

- Websocket人物互见前后端联调 ---zty
- Websocket聊天功能(世界&群聊) ---cjy
- 人物行走,碰撞检测，更新人物位置 ---dxl
- Websocket框架调研与开发 ---wxy

第三阶段

- NPC 天降奇遇 后台管理登录注册，修改页面 ---zty
- 后台管理上传页面，个人收藏展示 ---cjy
- 展品信息显示，oss存储展品，动态配置展品位置 ---dxl
- 后台管理接口支持，展品表结构变更 ---wxy

登录注册与权限控制

- 登录与注册的格式验证，以邮箱为例

```
let checkEmail = (rule, value, callback) => {
  if (value === "") {
    callback(new Error("请输入邮箱"));
  } else if (!this.checkEmailAddress(value)) {
    callback(new Error("邮箱格式不合法"));
  } else {
    callback();
    this.flag1 = true;
    return;
  }
  this.flag1 = false;
};
```

```
checkEmailAddress(str) {
  let reg = new RegExp(
    "^[a-z0-9]+([._\\-]*[a-z0-9])*@([a-z0-9]+[-a-z0-9]*[a-z0-9]+.){1,63}[a-z0-9]+$"
  );
  if (reg.test(str)) {
    return true;
  } else {
    return false;
  }
},
```

```
email: [{validator: checkEmail, trigger: "change"}],
```

- 只有验证全部通过时，才能向后台发送

登录注册与权限控制

- 登录成功的话：

```
window.sessionStorage.setItem("email", data.data.email);
```

- 路由登录控制：

```
  //如果没有登录，需要进入除登录和注册之外的页面，直接跳到登录页面
if (!token && (route.path !== '/register' && route.path !== '/')) {
  console.log("未登录" + route.path)
  next({
    path: '/'
  })
} else {
  if (route.name) {
    next()
  } else {
    next({
      path: '/nofound'
    })
  }
}
```

个人页面

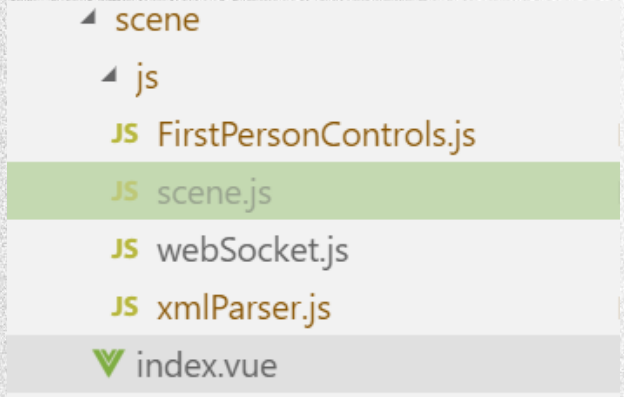
【功能】

修改形象，昵称，邮箱，密码

查看自己收藏的雕像

主场景Scene

- 用前端框架时，data域中尽量放必须的数据，框架的数据绑定会带来性能的影响。如three.js的场景配置，单独用js文件引入更加合理。
- 组件化简化代码结构。
- Loading页面，three.js页面模型未加载完时可能出现闪屏，跳出界面等现象，影响体验(方法：loader回调中计数)




```
<template>
  <div  v-loading="loading"
    element-loading-text="页面加载中"
    element-loading-spinner="el-icon-loading"
    element-loading-background="rgba(0, 0, 0, 0.6)">
    <div id="container"></div>
    <ui
      :websocket="websocket"
      :sculpture="sculptureInfo"
      :isMyFavour="isMyFavour"
      @changeFavour="changeFavour"
    ></ui>
  </div>
</template>
```

Scene——人物部分

- 避免重复加载——第一次加载时设置loading标志
- 模型缓存以提高性能——第一次加载一种形象时进行保存，之后调用copy
- render中广播位置

```
addFigure(id, userName, figure, pos, rotation) {
    if (!this.playerMap.has(id)) {
     this.playerMap.set(id, "LOADING"); //表示在加载中，防止重复加入
     var name = FigureMap[figure];
     if (this.figureCache.get(figure) != null) {//采用copy的方法
       var obj = this.figureCache.get(figure).clone();
       obj.position.set(pos.x, -10, pos.z);
       obj.rotation.set(rotation._x, rotation._y + Math.PI, rotation._z);
       obj.name = name;

       this.playerMap.set(id, obj);
       this.scene.add(obj);
     } else {
       /*objLoader加载模型 ...   */
       this.figureCache.set(figure,obj);
```

```
render() {
    that.fpc.update(that.clock.getDelta());
    if (that.websocket.connected) {
      that.websocket.broadcast(1, {
        position: that.fpc.control.position,
        rotation: that.fpc.control.rotation
      });
    }
}
```

Scene——展品

- 展品加载—— xml加载 or 数据库+json实现
- 获取点击的展品——THREE.Raycaster()
- 展品信息展示，收藏——axios请求后端

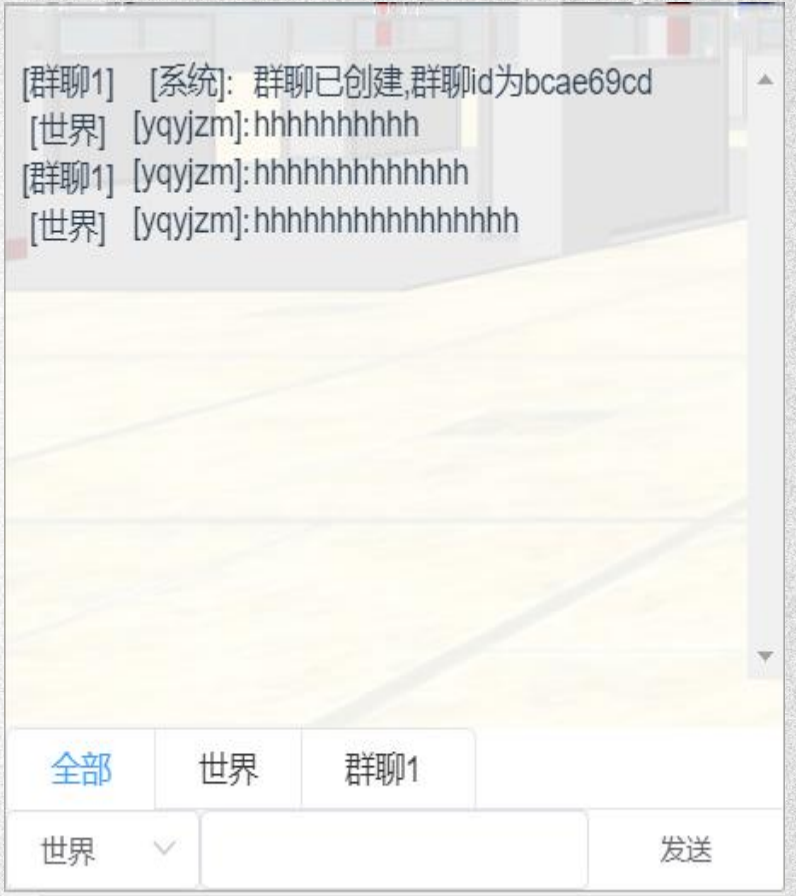
```
async onClick(event) {
  let mouse = new THREE.Vector2;
  mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  mouse.y = - (event.clientY / window.innerHeight) * 2 + 1;
  this.ClickRaycaster.setFromCamera(mouse, this.camera);

  var intersects = this.ClickRaycaster.intersectObjects(this.sculptures, true);
  if (intersects.length != 0) {
    let sid = intersects[0].object.parent.name;

    if (sid != this.sculptureInfo.sid) {
      let res = (await request.getStructureInfo(sid));
      if(res.code==0){
        this.sculptureInfo = res.data.structure;
        this.isMyFavour = res.data.isMyFavour;
      }

    }
```

聊天

【架构】

【前端消息数据结构】

```
msgData: [ {
            channel: channelName
            channelID: '0',
            msg: [{
                        toChannel: 'channelName',
                        sender: 'username',
                        content: 'content'
            }]
}],
```



【websocket】

```
that.stompClient.subscribe(
  "/topic/group/" + that.user.userId,
```

每个用户订阅了以自己userId命名的频道
群聊和私聊时后端只会把属于该用户的信息传过来

【后端】

后端保存了两个map：
一个key为: roomid，value为roomName
另一个key为roomid，value为一个List，里面是所有在此群聊的用户

【世界聊天】

沿用了广播的websocket接口

【群聊】

【数据结构】

```
data:{
  from:{}, 当前用户对象，包括id,name等等
  to: {
    name: string 群聊名字
    roomId: number 群聊id
    }
  type: 0/1/2/3 创建，聊天，加入，退出
  content: 聊天内容
}
```

【群聊流程】

【创建】

content 为所有拉入群聊的用户id
to.name为自定义的群聊名

to.roomId由后台指定一个唯一值，之后转发给所有在其中的用户。

前端收到后端发来的type=0消息后，创建频道

```
if (data.type === 0) {//创建
  that.tempChannel.unshift({
    channel: data.to.name,
    channelId: data.to.roomId,
    initMsg: {
      channel: data.to.roomId,
      sender: {
        userName: '系统',
      },
      content: '群聊已创建,群聊id为'+ data.to.roomId
    }
  });
```

tempChannel是一个队列，每次更改会触发vue的watch，再把它添加到msgData中，这样页面下会自动添加一个频道。

【聊天】

前端发送的信息由后端转发给所有在对应roomid的list中的用户

前端收到后端发来的type=1消息后，加入消息

```
} else if (data.type === 1) {//聊天
  that.tempMsg.unshift({
    channel: data.to.roomId,
    sender: data.from,
    content: data.content
  });
```

【加入】

加入需要手动输入roomid

后端会把当前用户加入list中

前端收到后端发来的type=2消息后，分别对发送者和其他原先的用户加入消息

```
if (data.from.userId === that.user.userId) {
  that.tempChannel.unshift({
    channel: data.to.name,
    channelId: data.to.roomId,
    initMsg: {
      channel: data.to.roomId,
      sender: {
        userName: '系统'
      },
      content: '您已加入群聊，群聊id为'+ data.to.roomId
    }
  });
} else {
  that.tempMsg.unshift({
    channel: data.to.roomId,
    sender: {
      userName: '系统'
    },
    content: data.from.userName + '加入群聊'
  })
}
```

【退出】

只要关闭前端的tab页，自动退出群聊

后端会把当前用户从list中删除，把消息转发给其他list中用户，当List中没人时，这个roomid会被删除

其他用户会收到谁退出的消息

```
if (data.from.userId !== that.user.userId) {
  that.tempMsg.unshift({
    channel: data.to.roomId,
    sender: {
      userName: '系统'
    },
    content: data.from.userName + '退出群聊'
  })
}
```

天降奇遇

```
animate() {
  aniId = requestAnimationFrame(that.animate);
  if (i <= 500) {
    that.control.position.set(that.control.position.x, i / 2, that.control.position.z);
    that.control.rotation.set(0, Math.PI / 2, 0);
    i++;
  }
  else if ((i > 500) && (i <= 1500)) {
    that.control.position.set(that.control.position.x + (i - 500) * (that.resultX - that.control.position.x) / 1000,
      250-(i-500)/4.65, that.control.position.z + (i - 500) * (that.resultZ - that.control.position.z) / 1000);
    that.control.rotation.set(0, 10, 0);
    i++;
  }
  else if (i > 1500) {
    i = 1;
    that.npc.rotation.set(0, -Math.PI / 2, 0);
    that.npc.position.set(that.resultX, 100, that.resultZ);
    cancelAnimationFrame(aniId);
  }
}
```

后台修改页面

- 编辑状态的切换，只修改一条数据

```
<el-table
  :data="tempList"
  style="width: 100%">
  <el-table-column>
    <template slot-scope="props">
      <el-form label-position="left" inline class="demo-table-expand">
        <el-form-item label="展品编号">
          <div v-if="!props.row.editing">
            <span>{{ props.row.sid }}</span>
          </div>
          <div v-else>
            <el-input v-model="props.row.sid" placeholder="请填写展品编号"></el-input>
          </div>
        </el-form-item>
```

```
// 编辑
handleEdit($index, row) {
  this.$set(this.tempList[$index], 'editing', true)
},
```

上传页

```
<el-upload
  :http-request="upload"
  :on-success="handleSuccess"
  accept=".obj"
  class="upload"
  drag
  action=""
>
```

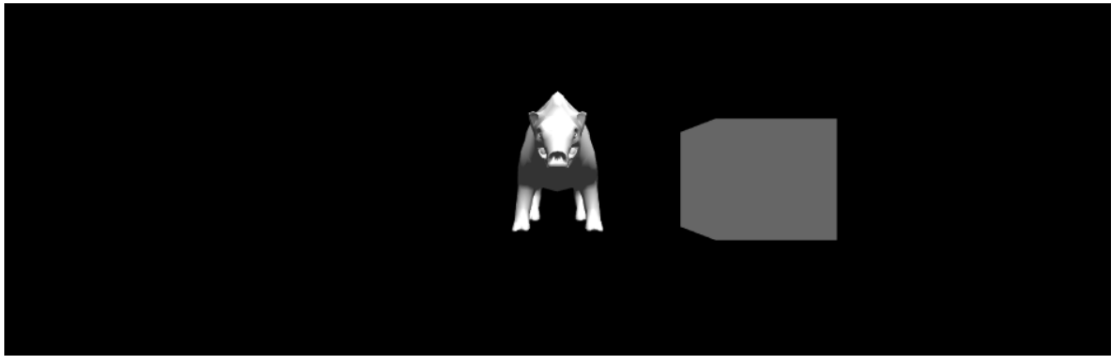
在initPreview中触发子组件的方法

http-request覆盖原来的上传行为

```
upload(file) {
  if (file.file.size > 1000000) {
    this.$message.error("文件超出大小限制");
    return
  }
  this.obj = file;
  this.initPreview(file);
},
initPreview(file) {
  var url = this.getObjectURL(file.file);
  this.localurl = url;
  this.$refs.preview.addModel(url);
},
getObjectURL(file) {
  let url = null;
  if (window.createObjectURL !== undefined) { // basic
    url = window.createObjectURL(file);
  } else if (window.webkitURL !== undefined) { // webkit or chrome
    url = window.webkitURL.createObjectURL(file);
  } else if (window.URL !== undefined) { // mozilla(firefox)
    url = window.URL.createObjectURL(file);
  }
  return url;
}
```

```
addModel(file) {
  var manager = new THREE.LoadingManager();
  new OBJLoader(manager).load(
    file,
    obj => {
      obj.position.set(-15, -10, 0);
      this.model = obj;
      this.scene.add(obj);
    }
  );
}
```

预览



提示　请调整参数使雕像大小与参照物大致相同，并使雕像角度是之正对屏幕

* 调整大小　0.3

* 调整角度　0

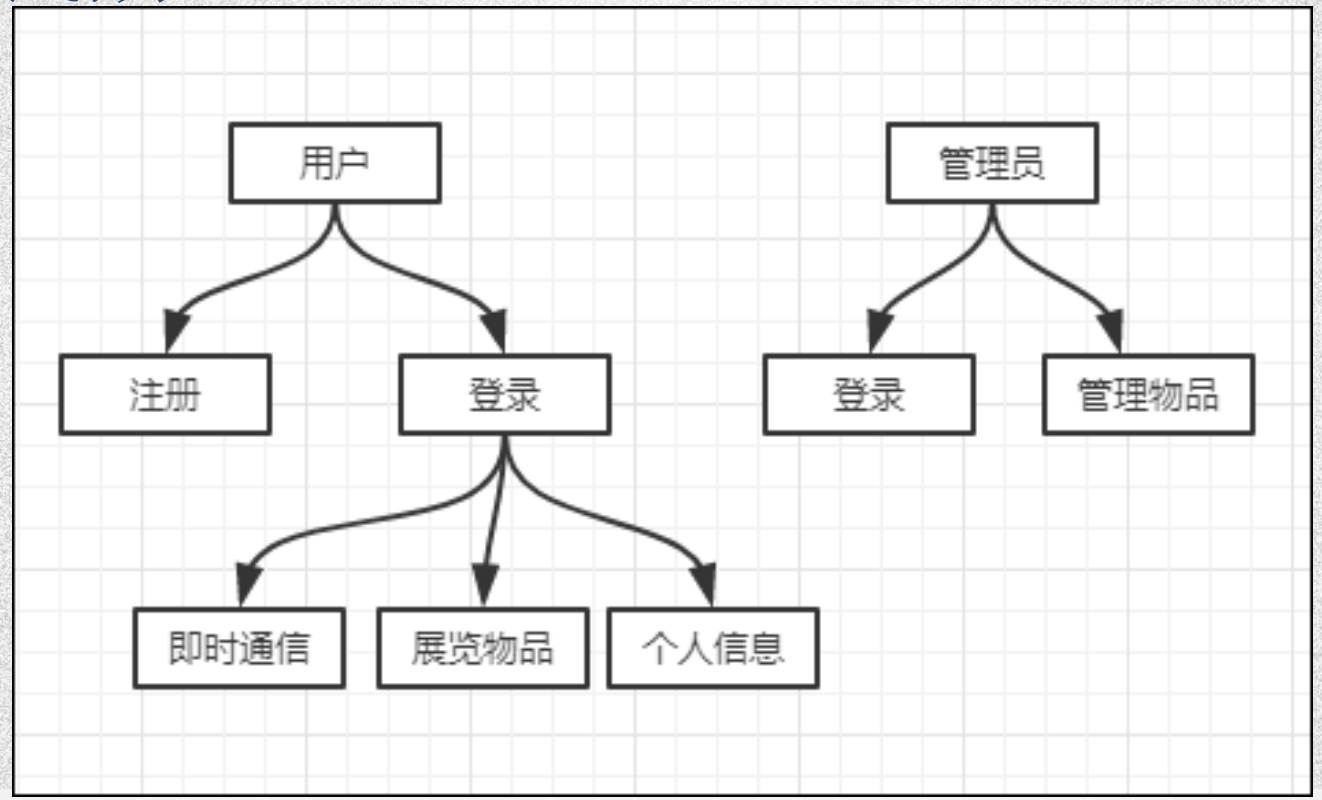
技术储备

Server

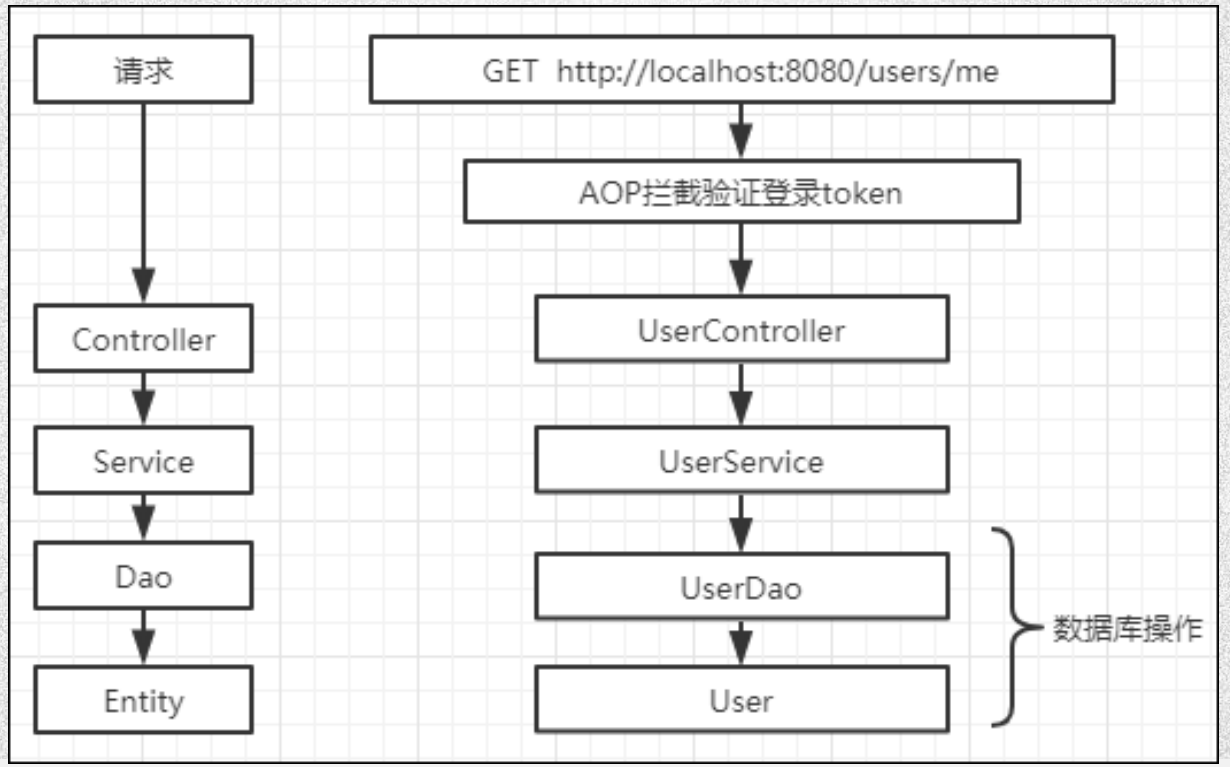
- **SpringBoot, Mysql, Redis,Mybatis,WebSocket**
- **SpringBoot技术路线:Maven，RESTful，Spring注解，Spring基础，进阶**
- **Redis：string数据结构，key-value**
- **Websocket/Sockjs：**STOMP在WebSocket之上提供了一个基于帧的线路格式（frame-based wire format）层，用来定义消息的语义。

模块划分

Server

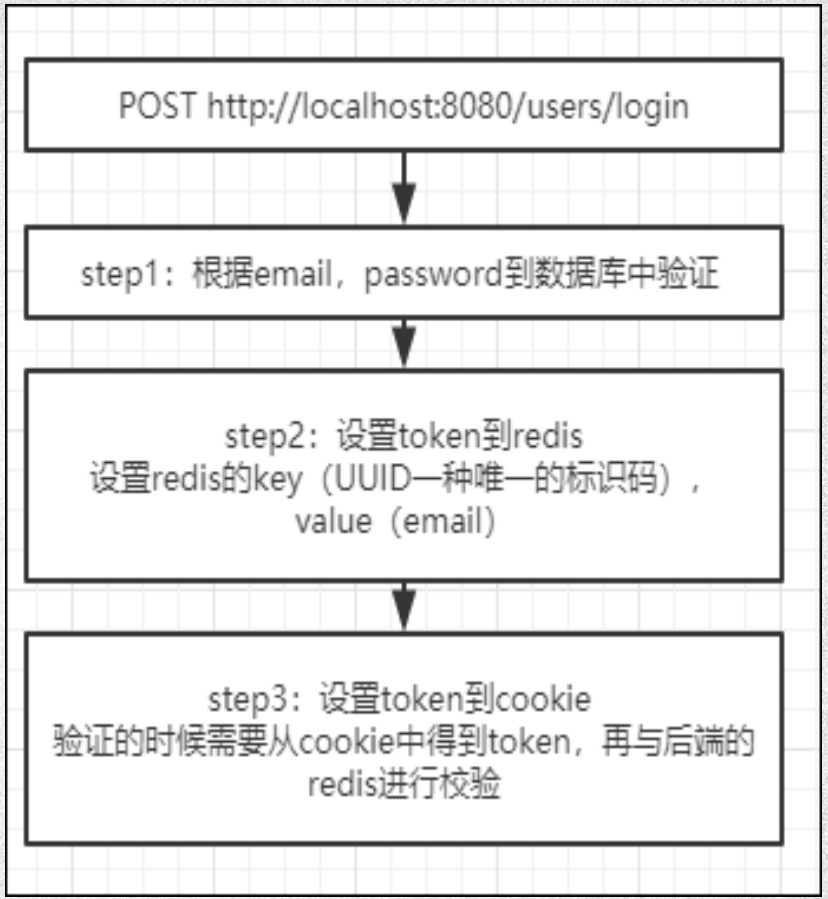


层次划分


请求
Controller
Service
Dao
Entity
GET http://localhost:8080/users/me
AOP拦截验证登录token
UserController
UserService
UserDao
User
数据库操作

难点分析

- **保存登录状态?**
- **Redis缓存Session**



Server

难点分析

- **如何进行权限控制？——AOP**

```
@Aspect
@Component
public class UserAuthorizeAspect {
    /**
     * 注意校验的时候需要排除登录登出的部分
     */
    @Pointcut("execution(public * edu.fudan.adweb.lab3.controller.User*.*(..))" +
            "&& !execution(public * edu.fudan.adweb.lab3.controller.UserController.*(..))")
    public void verify() {
    }
```

难点分析

Server

- **返回统一的数据格式**

```
{
        code: int,
        msg: String,
        data:{
                userId:int,
                userName:String,
                password:null,
                figure:int,
                email:String
        }
}
```

难点分析

- **全局统一异常处理——@RestControllerAdvice**

```
@RestControllerAdvice
public class UserExceptionHandler {

    //拦截用户个人信息异常
    @ExceptionHandler(value = UserException.class)
    public ResultVO handler(UserException e) {
        return ResultVOUtil.error(e.getCode(), e.getMessage());
    }
```

部署上线

Server

- **Mavan命令：mvn clean package –Dmaven.skip.test=true**
- **服务器命令：**
- **&:在命令后面加上& 实现后台运行**
- **nohup:一但把当前控制台关掉(退出帐户时)，作业就会停止运行。nohup命令可以在你退出帐户之后继续运行相应的进程。**
- **nohup java –jar web.jar &**

THANKS