**基于canvas+VUE实现大剧院选座功能（PC端）**

2023年09月05日

**目录**

[一、 研发背景 3](#_Toc80630604)

[二、 技术架构 3](#_Toc80630605)

[三、 具体实现 3](#_Toc80630606)

[1 服务端（该部分均用于自定义拍照&水印SDK的支撑） 3](#_Toc80630607)

[1.1 建立支撑库（sys\_dict和sys\_dict\_detail） 4](#_Toc80630608)

[1.2 核心配置参数说明 5](#_Toc80630609)

[1.3 SDK集成介绍 5](#_Toc80630610)

[1.3.1 初始化 5](#_Toc80630611)

[1.3.2 LiveParams对象 6](#_Toc80630612)

[1.3.3 视频文件存储路径 6](#_Toc80630613)

[1.3.4 回调函数 7](#_Toc80630614)

[2 APP端 7](#_Toc80630615)

[2.1 资源部署 7](#_Toc80630616)

[2.2 插件的开发 7](#_Toc80630617)

[2.2.1 编写插件类 7](#_Toc80630618)

[2.2.2 注册插件（dcloud\_properties.xml） 7](#_Toc80630620)

[2.3 开放API调用 7](#_Toc80630621)

[2.3.1 打开相机预览界面 7](#_Toc80630622)

[2.3.2 获取视频文件路径 8](#_Toc80630623)

[2.4 页面调用插件 8](#_Toc80630624)

[2.4.1 文件上传 8](#_Toc80630625)

[2.4.2 录制完成后获取视频文件路径 9](#_Toc80630626)

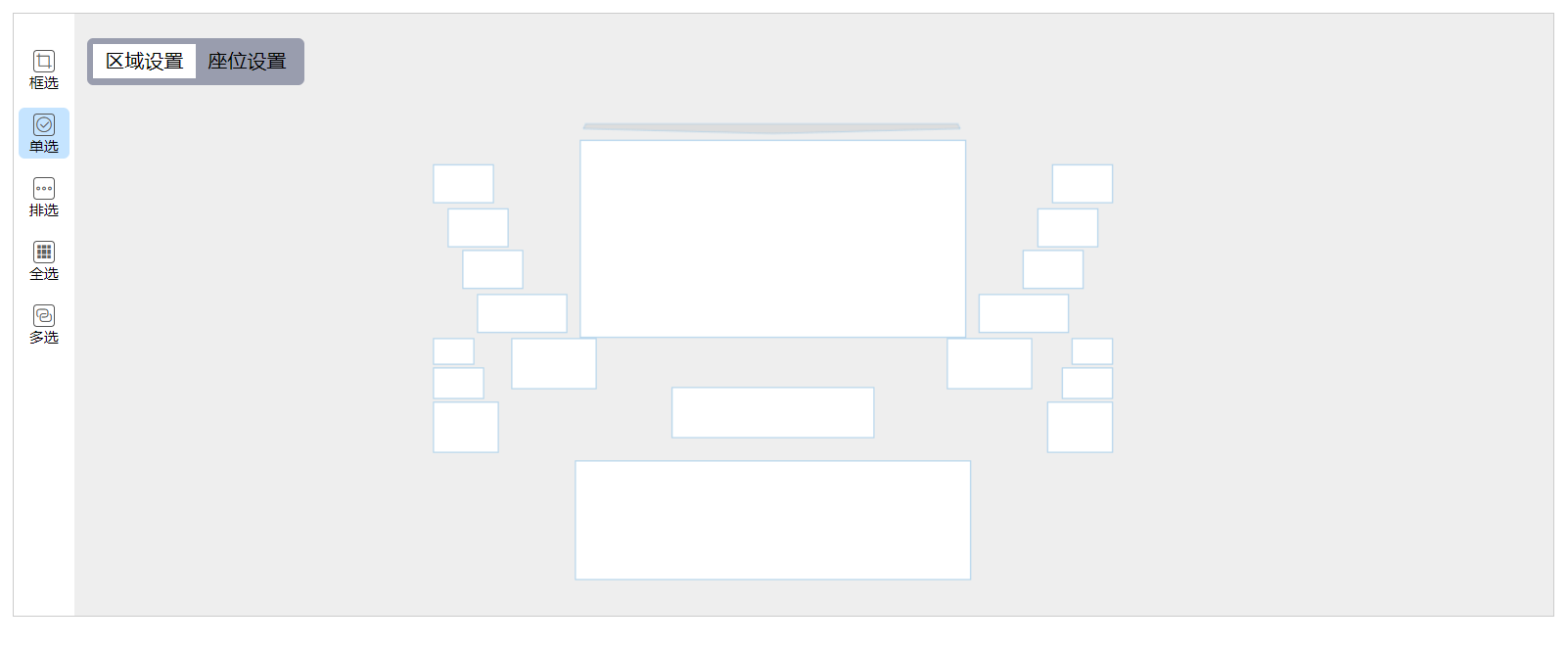
# 研发背景

某项目为大剧院管理系统，座位数在700+以上，使用div/li等dom元素渲染模拟座位，性能低，交互差，响应慢，甚至导致页面崩溃。基于这个情况，实现以canvas实现的剧场管理与座位管理功能。

# 技术架构

**Vue+canvas**

# 具体实现



图一

## 基础图形渲染

|  |  |  |
| --- | --- | --- |
| **变量** | **初始值** | **说明** |
| warpDom | Null | 画布容器 |
| EL | Null | Canvas对象 |
| width | 800px | 画布初始宽度 |
| Height | 600 | 画布初始高度 |
| Ctx | Null | Canvas 2D对象 |
| Scale | 0.2 | 缩放比 |
| MaxScale | 3 | 最大缩放比 |
| Minscale | 0.1 | 最小缩放比 |
| Step | 0.1 | 缩编步进值 |
| OffsetX | 0 | X坐标偏移量 |
| OffsetY | 0 | Y坐标偏移量 |
| activeShape | Null | 鼠标点击选中对象 |
| MouseDownOriginX | 0 | 鼠标按下X坐标轴初始记录对象 |
| MouseDownOriginY | 0 | 鼠标按下Y坐标轴初始记录对象 |

## Canvas构建区域

在vue中定义canvas区域，设定基础画布

1， <div id="chart\_wrap" class="chart\_wrap\_mid"></div>

2， this.wrapDom = document.getElementById("chart\_wrap");

**var** wrapDomStyle = getComputedStyle(this.wrapDom);

    this.width = parseInt(wrapDomStyle.width, 10);

    this.height = parseInt(wrapDomStyle.height, 10);

首先定义图形区域，也就是id为chart\_warp的dom元素，此元素是基于页面设定了自适应页面大小的宽高值，其中chart\_warp\_mid的属性如下：

.chart\_wrap\_mid {

  flex: 1; // 利用flex布局，左右2侧弹性盒子，flex：1为自适应宽度

  background: #eee;

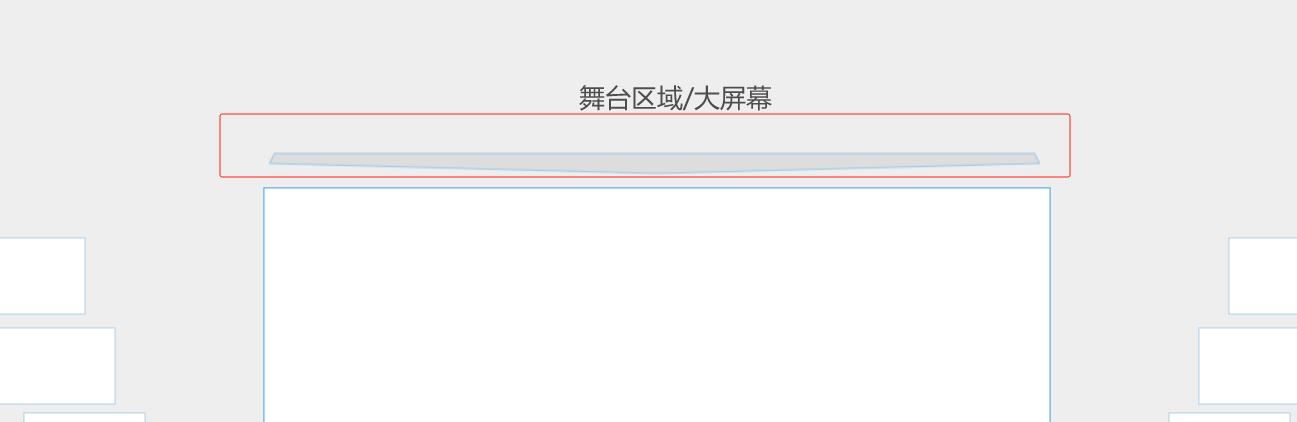
  height: calc(100vh - 215px);

}

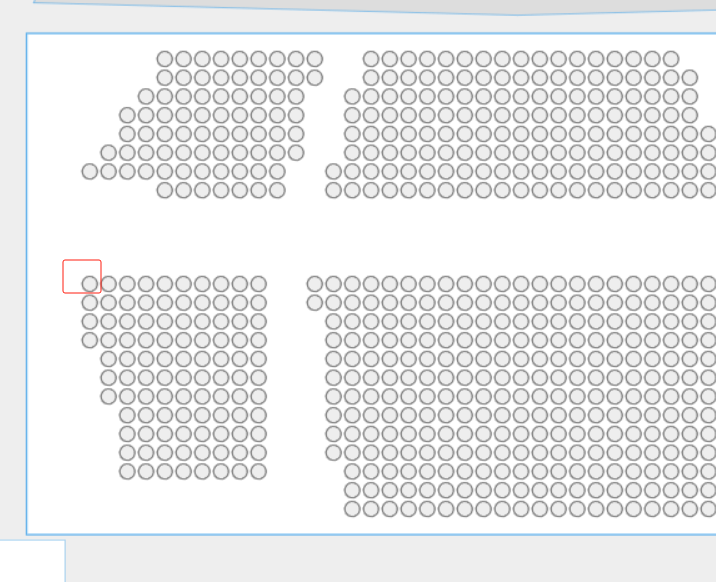
## 基础图形渲染

在大剧院中常用基础图形包括，房间，以长方体图形表示，房间包括剧场大厅，2楼大厅，2楼vip包房，以大小不同的长方形分别代表(如图2-1)。大屏幕以特定多边形代表(如图2-2)，座位以圆形小圈代表(如图2-3)。



2-1 

2-2



2-3

⑴ 房间区域/长方形渲染实现

*//  绘制矩形方法*

  drawRect(data, mark) {

    this.ctx.beginPath();

    this.ctx.fillStyle = data.fillStyle;

    this.ctx.strokeStyle = "#0000ff";

    this.ctx.strokeRect(...data.data);

    this.ctx.fillRect(...data.data);

  }

⑵ 大屏幕/线条渲染实现

*// 绘制线条方法*

  drawLine(data) {

**var** arr = data.data.concat();

**var** ctx = ctx || this.ctx;

    ctx.beginPath();

    ctx.moveTo(arr.shift(), arr.shift());

    ctx.lineWidth = data.lineWidth || 1;

    do {

      ctx.lineTo(arr.shift(), arr.shift());

    } while (arr.length);

    ctx.fillStyle = data.fillStyle;

    ctx.fill();

    ctx.stroke();

  }

⑶ 座位/圆圈渲染

*// 绘制圆形*

  drawCircle(data) {

    this.ctx.beginPath();

    this.ctx.fillStyle = data.fillStyle;

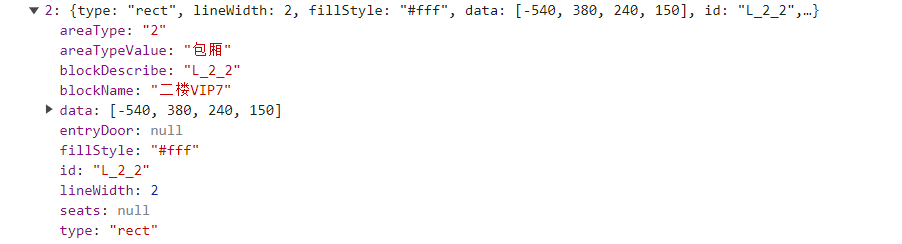
    this.ctx.arc(data.x, data.y, data.r, 0, 2 \* Math.PI);

    this.ctx.fill();

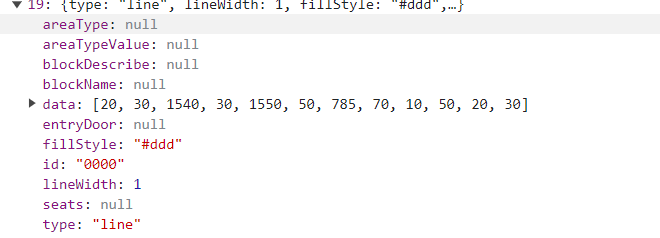
  }

## 数据判断以及绑定到画布对象，根据不同类型进行判断

下图为房间的基础图形数据



下图为屏幕的基础图形数据



下图为座位基础数据



根据不同类型渲染图形方法

draw(item, mark) {

    console.log(item, "draw方法");

    this.ctx.setTransform(

      this.scale,

      0,

      0,

      this.scale,

      this.offsetX,

      this.offsetY

    );

    switch (item.type) {

      case "line":

        this.drawLine(item);

        break;

      case "rect":

        this.drawRect(item, mark);

        break;

      case "circle":

        this.drawCircle(item);

        break;

    }

  }

## 移动画布

鼠标按下可以移动画布，下面是事件绑定和实现

 *// 移动画布*

  moveCanvasFunc = (e) **=>** {

**var** maxMoveX = this.El.width / 2;

**var** maxMoveY = this.El.height / 2;

**var** offsetX = this.mousedownOriginX + (e.offsetX - this.targetX);

**var** offsetY = this.mousedownOriginY + (e.offsetY - this.targetY);

    this.offsetX = Math.abs(offsetX) > maxMoveX ? this.offsetX : offsetX;

    this.offsetY = Math.abs(offsetY) > maxMoveY ? this.offsetY : offsetY;

    this.render();

  };

  removeMouseMove = () **=>** {

    this.wrapDom.style.cursor = "";

    this.El.removeEventListener("mousemove", this.moveCanvasFunc, false);

    this.El.removeEventListener("mousemove", this.moveShapeFunc, false);

  };

## 定义鼠标点击注册点击事件

鼠标点击画布不同的元素用于获取当前用户操作的对象：

*// 添加鼠标移动 功能，获取保存当前点击坐标*

  addMouseMove = (e) **=>** {

    this.targetX = e.offsetX;

    this.targetY = e.offsetY;

    this.mousedownOriginX = this.offsetX;

    this.mousedownOriginY = this.offsetY;

**var** x = (this.targetX - this.offsetX) / this.scale;

**var** y = (this.targetY - this.offsetY) / this.scale;

    this.activeShape = null;

    this.data.forEach((item) **=>** {

      switch (item.type) {

        case "rect":

*// this.isInnerRect(...item.data, x, y) && (this.activeShape = item);*

          if (item.seats) {

            item.seats.forEach((element) **=>** {

              this.isInnerCircle(

                element.positionLeft,

                element.positionTop,

                14,

                x,

                y

              ) && (this.activeShape = element);

            });

          }

          break;

        case "circle":

          this.isInnerCircle(item.x, item.y, item.r, x, y) &&

            (this.activeShape = item);

          break;

        case "line":

**var** lineNumber = item.data.length / 2 - 1;

**var** flag = false;

          for (**let** i = 0; i < lineNumber; i++) {

**let** index = i \* 2;

            flag = this.isInnerPath(

              item.data[index],

              item.data[index + 1],

              item.data[index + 2],

              item.data[index + 3],

              x,

              y,

              item.lineWidth || 1

            );

            if (flag) {

              this.activeShape = item;

              break;

            }

          }

      }

    });

    if (!this.activeShape) {

      this.wrapDom.style.cursor = "grabbing";

      this.El.addEventListener("mousemove", this.moveCanvasFunc, false);

    } else {

      this.activeShape.fillStyle = "#CCCCFF";

      this.data.forEach((element) **=>** {

        if (element.id !== this.activeShape.id && element.id !== "0000") {

          element.fillStyle = "#fff";

        }

      });

      console.log(this.activeShape, "cccccc");

**let** clickMark = "AS";

      this.render(clickMark);

      this.wrapDom.style.cursor = "all-scroll";

      this.shapedOldX = null;

      this.shapedOldY = null;

      this.El.addEventListener("mousemove", this.moveCanvasFunc, false);

    }

  };

## 缩放功能实现与事件绑定

scrollFunc = (e) **=>** {

*// 阻止默认事件 （缩放时外部容器禁止滚动）*

    e.preventDefault();

    if (e.wheelDelta) {

**var** x = e.offsetX - this.offsetX;

**var** y = e.offsetY - this.offsetY;

**var** offsetX = (x / this.scale) \* this.step;

**var** offsetY = (y / this.scale) \* this.step;

      if (e.wheelDelta > 0) {

        this.offsetX -= this.scale >= this.maxScale ? 0 : offsetX;

        this.offsetY -= this.scale >= this.maxScale ? 0 : offsetY;

        this.scale += this.step;

      } else {

        this.offsetX += this.scale <= this.minScale ? 0 : offsetX;

        this.offsetY += this.scale <= this.minScale ? 0 : offsetY;

        this.scale -= this.step;

      }

      this.scale = Math.min(this.maxScale, Math.max(this.scale, this.minScale));

      this.render();

    }

  };

*// 添加 mousewhell 事件*

  addMouseWhell = () **=>** {

    document.addEventListener("mousewheel", this.scrollFunc, {

      passive: false,

    });

  };

*// 移除mousewhell 事件*

  removeMouseWhell = () **=>** {

    document.removeEventListener("mousewheel", this.scrollFunc, {

      passive: false,

    });

  };

## Cancas重绘，针对于用户操作，实现数据重绘canvas

*// 渲染整个 图形画布*

    render() {

      this.El.width = this.width;

      this.areaData.forEach((item) **=>** {

        this.draw(item);

      });

    },

## 绑定所有事件

mounted() {

    this.fullscreenLoading = true;

    this.wrapDom = document.getElementById("chart\_wrap");

**var** wrapDomStyle = getComputedStyle(this.wrapDom);

    this.width = parseInt(wrapDomStyle.width, 10);

    this.height = parseInt(wrapDomStyle.height, 10);

    console.log(this.width, this.height, "宽度---高度");

*// 创建canvas画布*

    this.El = document.createElement("canvas");

    this.El.height = 800;

    this.El.width = this.width;

    this.ctx = this.El.getContext("2d");

    this.wrapDom.appendChild(this.El);

    this.offsetX = this.width / 2 - 360; *// 画布X轴偏移值*

    this.offsetY = this.height / 4; *// 画布Y轴偏移值*

**let** venueId = this.$route.query.venueId;

*// 接口数据，*

    allAreaList({ venueId }).then((res) **=>** {

      if (res.success) {

        this.areaData = res.result;

*// 添加滚轮判断事件*

        this.addScaleFunc();

*// 添加拖拽事件*

        this.addDragFunc();

        this.render();

      }

    });

  },

# 四，总结

组件构建了基础的逻辑实现底层，根据此组件分别拓展了，渲染不同剧场。座位单选，多选，框选等批量选择。绑定座位票品与价格，已售未售等属性绑定。满足了项目的实际需求和性能需求。