

- 十五届模拟二
  - 1. 相不相等
  - 2. 三行情书
  - 3. 电影院订票
  - 4. 老虎机
  - 5. 星际通讯
  - 6. 蓝桥排位赛
  - 7. 拼出一个未来
  - 8. 实时展示权限日志
  - 9. 超级英雄联盟
  - 10. 万能合成台 - 题解
  - 11. 账户验证

## 十五届模拟二

---

### 1. 相不相等

---

- 解题思路：

这段代码实现了一个简单的断言函数，用于进行值的比较。它包含两个方法：`toBe`和`notToBe`。我们可以根据题目要求或者特定的测试条件，使用这些方法来判断给定的值是否符合预期。

- 解题代码：

```
// 定义一个断言函数
var expectFn = function(val) {
  // 返回一个包含两个方法的对象
  return {
    // 判断值是否相等
    toBe: function(expected) {
      if (val === expected) {
        return true; // 值相等，返回true
      } else {
        return "Not Equal"; // 值不相等，返回"Not Equal"
      }
    },
    // 判断值是否不相等
    notToBe: function(expected) {
      if (val !== expected) {
        return true; // 值不相等，返回true
      }
    }
  };
}
```

```
    } else {  
        return "Equal"; // 值相等, 返回"Equal"  
    }  
}  
};  
};  
  
// 使用示例  
var result1 = expectFn(5).toBe(5);  
console.log(result1); // true  
  
var result2 = expectFn(10).notToBe(5);  
console.log(result2); // true  
  
var result3 = expectFn(10).toBe(5);  
console.log(result3); // "Not Equal"  
  
var result4 = expectFn(5).notToBe(5);  
console.log(result4); // "Equal"
```

这段代码可以用于测试特定的条件，例如判断两个值是否相等或不相等。根据题目要求或者具体的测试需求，我们可以调用`toBe`和`notToBe`方法来进行值的比较，并根据返回结果判断测试是否通过或失败。

## 2. 三行情书

- 解题思路

这道题主要考察 CSS 中的单行、多行文本溢出的使用，当然本题也有坑点，请看接下来的解析。

1. 对于 `span` 要求单行溢出显示，首先要有想法是，有个限定宽度，这个在父元素已经限制好了，可以省略，接着想要让他限制在一行，不换行，于是要用到 `white-space` 属性，
2. 然后不换行超过就要隐藏，于是有 `overflow`，
3. 超出后显示省略号，是 `text-overflow: ellipsis`。

要这样一步一步循序渐进地思考。

但是这有个坑点是 `span`，默认情况下是一个内联元素，它主要用于在行内添加样式或标记特定的文本部分。当一个文本内容超 `span` 元素的宽度时，溢出的部分可能会影响周围的元素，但是 `span` 本身不会自动换行或调整其尺寸来适应溢出的内容。要让它生效，就要将其改变成块级元素 或者行内块元素。

接下来是对 `p` 进行三行显示。这使用了一些特定于 WebKit 浏览器引擎的属性，用于控制文本的溢出和显示行数。它们的作用如下：

**display: -webkit-box:** 这个属性与值的组合将元素的显示类型设置为 `-webkit-box`，它在 WebKit 浏览器中允许多行文本溢出的处理方式。

**-webkit-box-orient: vertical:** 这个属性与值的组合用于设置元素内部内容的排列方向为垂直方向。也就是说，文本内容将按垂直方向排列。

**-webkit-line-clamp: 3:** 这个属性与值的组合用于限制元素内部文本的行数。在这个例子中，3 表示只显示 3 行文本，超过 3 行的部分将被隐藏。

**overflow: hidden:** 这个属性用于控制元素内部内容的溢出处理方式。在这个例子中，超过指定行数的文本将被隐藏。

通过这些属性的组合，可以实现 WebKit 浏览器中限制文本行数并进行溢出处理的效果。当然如果把这里的 3 换成 1,也可以实现 `span` 的单行显示。

- 解题代码

```
span {
  font-size: 20px;
  color: #837362;
  display: block;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}

p {
  color: #837362;
  display: -webkit-box;
  -webkit-box-orient: vertical;
  -webkit-line-clamp: 3;
  overflow: hidden;
}
```

## 3. 电影院订票

- 解题思路

获取 DOM 元素：首先通过 `document.getElementById` 和 `document.querySelector` 方法获取需要操作的DOM元素，包括座位区域、电影

名、电影价格、容器以及座位计数和总价格的节点。

发起数据请求并生成电影布局：使用Axios库发起数据请求，获取电影信息。通过 `axios.get` 方法获取数据后，使用解构赋值将电影名称、价格和座位信息提取出来。然后根据座位信息，使用 `document.createElement` 方法创建相应的DOM元素，并添加相应的类名和属性。根据座位的占用情况，添加 `occupied` 类名以标识已占用的座位。最后，将生成的座位元素添加到座位区域的 DOM 元素中。

更新选中座位计数和总价格：定义一个函数 `updateSelectedCount`，该函数通过 `document.querySelectorAll`方法选择所有选中的座位元素，并获取其数量。然后更新座位计数和总价格的显示。

添加点击事件监听器：使用 `addEventListener` 方法给容器元素添加点击事件监听器。当点击的元素是座位且未被占用时，切换座位的选中状态，即添加或移除 `selected` 类名。然后调用 `updateSelectedCount` 函数更新选中座位计数和总价格的显示。

调用函数生成电影布局：在最后调用`generateMovieLayout` 函数，触发数据请求和电影布局的生成。

```
// TODO: 1. 完成数据请求，生成电影名，价格以及座位情况
// 获取座位区域、电影名和价格的节点
const seatAreaNode = document.getElementById("seat-area");
const movieNameNode = document.getElementById("movie-name");
const moviePriceNode = document.getElementById("movie-price");

// TODO: 2. 绑定点击事件，实现订票功能
// 获取容器、座位计数和总价格的节点
const container = document.querySelector(".container");
const count = document.getElementById("count");
const total = document.getElementById("total");

let ticketPrice = null; // 用于存储电影票价格的变量

// 生成电影布局的函数
function generateMovieLayout() {
  // 发起数据请求，获取电影信息
  axios.get("./data.json").then(({ data }) => {
    const { name, price, seats } = data; // 解构电影信息对象中的名称、价格和座位信息
    ticketPrice = price; // 将电影票价格存储到变量中

    // 更新电影名和价格的显示
    movieNameNode.innerText = name;
    moviePriceNode.innerText = price;

    // 生成座位布局
    seats.forEach((row) => {
      const rowNode = document.createElement("div"); // 创建表示座位行的div元素
```

```

        rowNode.classList.add("row"); // 添加row类到div元素中，用于样式控制
        row.forEach((seat) => {
            const seatNode = document.createElement("div"); // 创建表示座位的div元素
            seatNode.classList.add("seat"); // 添加seat类到div元素中，用于样式控制
            if (seat) {
                seatNode.classList.add("occupied"); // 如果座位已被占用，添加occupied类到div元素中，用于样式控制
            }
            rowNode.appendChild(seatNode); // 将座位div元素添加到座位行div元素中
        });
        seatAreaNode.appendChild(rowNode); // 将座位行div元素添加到座位区域中
    });
}

// 更新选中座位计数和总价格的函数
function updateSelectedCount() {
    const selectedSeats = document.querySelectorAll(".row .seat.selected"); // 获取所有选中的座位元素
    const selectedSeatsCount = selectedSeats.length; // 获取选中座位的数量
    count.innerText = selectedSeatsCount; // 更新选中座位计数的显示
    total.innerText = selectedSeatsCount * ticketPrice; // 更新总价格的显示
}

// 给容器添加点击事件监听器
container.addEventListener("click", (e) => {
    if (
        e.target.classList.contains("seat") && // 如果点击的元素是座位
        !e.target.classList.contains("occupied") // 并且座位未被占用
    ) {
        e.target.classList.toggle("selected"); // 切换座位的选中状态

        updateSelectedCount(); // 更新选中座位计数和总价格的显示
    }
});

generateMovieLayout(); // 调用函数生成电影布局

```

## 4. 老虎机

- 解题思路：

1. 首先，根据给定的参数 r1、r2、r3，使用 `document.querySelector` 方法选择对应的元素。这些元素通过 CSS 选择器指定，分别是 `#sevenFirst>li:nth-child(${r1})`、`#sevenSecond>li:nth-child(${r2})` 和 `#sevenThird>li:nth-child(${r3})`。这些选择器表示选择第 r1、r2、r3 个 `li` 元素，前面的 `#sevenFirst`、`#sevenSecond` 和 `#sevenThird` 是父元素的 id。

2. 通过 `getAttribute` 方法获取选中元素的 `data-point` 属性值，并分别赋给变量 `n1`、`n2`、`n3`。
3. 使用条件语句判断 `n1`、`n2`、`n3` 是否相等，即判断三个变量的值是否相等。如果三个变量都相等，即 `n1 == n2 && n1 == n3 && n2 == n3` 成立，则表示中奖了。
4. 根据中奖与否的判断结果，使用 `innerText` 属性将对应的文本信息赋给 `textPanel` 元素。如果中奖，则赋值为 "恭喜你，中奖了"；否则，赋值为 "很遗憾，未中奖"。

总结：该函数的作用是根据给定的 `r1`、`r2`、`r3` 参数，选择对应的元素，获取其 `data-point` 属性值，并判断是否中奖。最后，将结果显示在 `textPanel` 元素上。

- 解题代码

```
GetResult(r1, r2, r3) {
  let firstSelected = document.querySelector(
    `#sevenFirst>li:nth-child(${r1})`
  );
  let n1 = firstSelected.getAttribute("data-point");
  let secondSelected = document.querySelector(
    `#sevenSecond>li:nth-child(${r2})`
  );
  let n2 = secondSelected.getAttribute("data-point");
  let thirdSelected = document.querySelector(
    `#sevenThird>li:nth-child(${r3})`
  );
  let n3 = thirdSelected.getAttribute("data-point");
  if (n1 == n2 && n1 == n3 && n2 == n3) {
    textPanel.innerText = "恭喜你，中奖了";
  } else {
    textPanel.innerText = "很遗憾，未中奖";
  }
}
```

## 5. 星际通讯

- 解题思路

给定一个外星人的密语，每个密语由三个字符组成，我们需要将其翻译成人类可读的语言。给定的密语和对应的翻译规则存储在一个名为 `codonTable` 的对象中。我们需要遍历密语的每个密文，将其翻译成相应的文字，并将翻译后的文字存储在一个数组中。如果遇到无效的密文或遇到 "stop" 密语，翻译过程将停止。最后，将翻译后的词语数组连接成一个字符串作为结果返回。

- 解题代码

```
const translate = (alienMessage) => {
  if (!alienMessage) {
    return '';
  }

  const codonTable = {
    'IIX': '人类',
    'VII': '哈喽',
    'III': '你好',
    'IXI': '赞',
    'XVI': '嗨',
    'CUV': '打击',
    'XII': '夜晚',
    'IVI': '我',
    'XIC': '想',
    'XIV': '交个朋友',
    'VIX': '月亮',
    'XCI': '代码',
    'XIX': '祈福',
    'XVI': '和',
    'XXI': 'stop',
  };

  const proteins = []; // 创建一个空数组用于密码序列

  for (let i = 0; i < alienMessage.length; i += 3) {
    const codon = alienMessage.substr(i, 3); // 从 alienMessage 中提取长度为3的密码子

    const protein = codonTable[codon]; // 根据密码子在密码子表 (codonTable) 中查找对应的密语

    if (!protein) {
      return '无效密语'; // 如果密码子无法在密码子表中找到对应的密语，返回错误提示信息
    }

    if (protein === 'stop') {
      break; // 如果找到的密语是停止信号 ('stop')，则终止循环
    }

    proteins.push(protein); // 将找到的密语添加到密码序列数组中
  }

  return proteins.join(''); // 返回密语序列数组中的元素，使用空字符串连接成一个字符串
}
```

## 6. 蓝桥排位赛

- 解题思路

1. 在 `setup` 函数中使用 `ref` 创建响应式变量 `chartsData`，用于存放请求到的数据。
2. 在 `onMounted` 钩子函数中发送 Axios 请求，获取 `./mock/map.json` 的数据，并将数据赋值给 `chartsData.value`。
3. 在 `showChartBar` 方法中，使用 ECharts 初始化图表实例，并根据数据计算出前 10 名学校的战力值 (`power`) 和名称 (`name`)。
4. 配置地图的 `series`，将数据 `chartsData.value` 应用到地图中。

- 解题代码

```
const {createApp,ref,onMounted} = Vue;
const app = createApp({
  setup() {
    const chartsData = ref([]);
    onMounted(() => {
      axios.get('./mock/map.json').then((response) => {
        chartsData.value = response.data;
        showChartBar();
        showChinaMap();
      });
    });
  }
});

// 展示柱状图
const showChartBar = () => {
  const myChart = echarts.init(document.getElementById('chart'));

  let data = chartsData.value.map((item, index) => {
    return item.school_power;
  });
  let result = data.flat(1).sort((a, z) => {
    return z.power - a.power;
  });
  let arr = result.slice(0, 10);
  let school = arr.map((item) => {
    return item.name;
  });
  let power = arr.map((item) => {
    return item.power;
  });

  // 指定配置和数据
  const option = {
    xAxis: {
      type: 'category',
      axisLabel: { interval: 0, rotate: 40 },
      data: school,
    },
    grid: {
      left: '3%',
      right: '4%',
      bottom: '3%',
      containLabel: true,
    },
  };
};
```



```

    },
    yAxis: {
      type: 'value',
      boundaryGap: [0, 0.01],
    },
    series: [
      {
        data: power,
        type: 'bar',
        showBackground: true,
        backgroundStyle: {
          color: 'rgba(180, 180, 180, 0.2)',
        },
        itemStyle: {
          color: '#8c7ae6',
        },
      },
    ],
  },
};

// 把配置给实例对象
myChart.setOption(option);
// 根据浏览器大小切换图表尺寸
window.addEventListener('resize', function () {
  myChart.resize();
});

// 展示地图
const showChinaMap = () => {
  const chinaMap = echarts.init(document.getElementById('chinaMap'));

  // 进行相关配置
  const mapOption = {
    tooltip: [
      {
        backgroundColor: '#fff',
        subtext: 'aaa',
        borderColor: '#ccc',
        padding: 15,
        formatter: (params) => {
          return (
            params.name +
            '热度值:' +
            params.value +
            '<br>' +
            params.data.school_count +
            '所学校已加入备赛'
          );
        },
      },
    ],
    textStyle: {
      fontSize: 18,
      fontWeight: 'bold',
      color: '#464646',
    },
    subtextStyle: {
      fontSize: 12,

```

```

        color: '#6E7079',
    },
},
],
geo: {
    // 这个是重点配置区
    map: 'china', // 表示中国地图
    label: {
        normal: {
            show: false, // 是否显示对应地名
        },
    },
    itemStyle: {
        normal: {
            borderColor: 'rgb(38,63,168)',
            borderWidth: '0.4',
            areaColor: '#fff',
        },
        emphasis: {
            // 鼠标移入的效果
            areaColor: 'rgb(255,158,0)',
            shadowOffsetX: 0,
            shadowOffsetY: 0,
            shadowBlur: 20,
            borderWidth: 0,
            shadowColor: 'rgba(0, 0, 0, 0.5)',
        },
    },
},
visualMap: {
    show: true,
    left: 'center',
    top: 'bottom',
    type: 'piecewise',
    align: 'bottom',
    orient: 'horizontal',
    pieces: [
        {
            gte: 80000,
            color: 'rgb(140,122,230)',
        },
        {
            min: 50000,
            max: 79999,
            color: 'rgba(140,122,230,.8)',
        },
        {
            min: 30000,
            max: 49999,
            color: 'rgba(140,122,230,.6)',
        },
        {
            min: 10000,
            max: 29999,
            color: 'rgba(140,122,230,.4)',
        },
        {

```

```

        min: 1,
        max: 9999,
        color: 'rgba(140,122,230,.2)',
    },
    ],
    textStyle: {
        color: '#000',
        fontSize: '11px',
    },
    },
    series: [
        {
            type: 'map',
            geoIndex: 0,
            data: chartsData.value,
        },
    ],
    ],
};

// 把配置给实例对象
chinaMap.setOption(mapOption);
};

return {
    chartsData,
    showChartBar,
    showChinaMap,
};
},
});

app.mount('#app');
```

## 7. 拼出一个未来

- 解题思路

本题要求实现一个拼图游戏，用户可以通过拖动拼图块来改变其位置。当所有拼图块按照正确的顺序排列时，显示成功消息；否则隐藏成功消息。

为了实现这个功能，我们需要完成以下几个步骤：

1. 获取拖动的拼图块和目标拼图块的图片元素。
2. 交换拖动的拼图块和目标拼图块的图片源、ID和替代文本。
3. 获取所有图片元素的 dataset.id 属性值并转换为数字。
4. 定义成功拼图后的数组顺序。
5. 根据当前拼图块顺序判断是否成功，显示或隐藏成功消息。

- 解题代码

```
// 获取拖动的拼图块和目标拼图块的图片元素
const [draggedPieceImg, thisImg] = [draggedPiece.querySelector('img'),
this.querySelector('img')];

// 交换拖动的拼图块和目标拼图块的图片源、ID和替代文本
[thisImg.src, thisImg.dataset.id, draggedPieceImg.src,
draggedPieceImg.dataset.id] =
    [draggedPieceImg.getAttribute('src'), draggedPieceImg.dataset.id,
thisImg.getAttribute('src'), thisImg.dataset.id];

// 获取所有图片元素的 dataset.id 属性值并转换为数字
const imageSrcArray = [...document.querySelectorAll(".puzzle-piece
img")].map(item => Number(item.dataset.id));

// 定义成功拼图后的数组顺序
const successOrder = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// 根据当前拼图块顺序判断是否成功，显示或隐藏成功消息
successMessage.className = JSON.stringify(imageSrcArray) ===
JSON.stringify(successOrder) ? 'show' : 'hide';
```

## 8. 实时展示权限日志

- 解题思路 本道题考察了四个知识点：
  - nodejs 内置模块（**fs**、**http**）
  - vue3 生命周期钩子函数 **onBeforeMount**，**setup** 组合式 api
  - js 内置对象 **JSON**
  - **axios**

本题的大致解题思路如下：

### 1. 实现服务器 **GET** 请求接口。所需知识点：

- **nodejs http** 模块。该模块中的 **http.ClientRequest** 对象：本需求已经对需要进行响应的请求做出了判断，考生需处理获取学生数据并响应给客户端。因为已经实现了响应数据的方法 **send**，因此，仅需要将参数传递给该方法即可，参数详见注释。
- **fs**：本题使用到了读取数据的功能，下面仅介绍读取数据的功能。读取文件示例：

```

// 1. 创建读取流
const rs = fs.createReadStream(url); // 传入读取的文件路径，创建一个可读流对象
let result = ''; // 声明一个变量来存储文件内容
rs.on('data', data => {
  // 文件读取时触发
  result += data; // 将读取的文件数据添加到所定义的变量里，以供后续使用
}); // 读取文件内容
rs.on('end', () => {
  // 文件数据读取完毕后触发
  // ...do something
}); // 文件读取完毕

// 2. 使用fs的同步读取文件方法
let fileContent = ''; // 保存文件内容
fileContent = fs.readFileSync(dataUrl, 'utf-8'); // 传入文件路径，和编码格式

// 3. 使用fs的异步读取文件方法，需保证外层函数为async标记的函数
let fileContent = ''; // 保存文件内容
fileContent = await fs.promises.readFile(dataUrl, 'utf-8'); // 传入文件路径，和编码格式

```

## 2. 实现服务器 PUT 请求接口。所需知识点：

- **nodejs http** 模块。该模块中的 **http.ClientRequest** 对象：本需求使用到了该实例对象的 **req.on** 方法，通过流的方式读取请求体。用法见下方示例。本需求使用到了文件读取操作，具体见上方示例。已经对需要进行响应的请求做出了判断，考生需处理获取请求体数据并响应给客户端。因为已经实现了响应数据的方法 **send**，因此，仅需要将参数传递给该方法即可，参数详见注释。

示例如下：

```

let result = ''; // 保存请求体
req.on('data', data => {
  // 通过流的方式读取
  result += data; // 将当前读到的请求体保存到result中
})
req.on('end', () => {
  // 请求体读取完毕时调用
  if (result) {
    send(res, 0, '', result); // 调用已经实现的send方法，将result响应给客户端
  }
})

```

## 3. 实现服务器 POST 请求接口。所需知识点：

- `nodejs http` 模块。该模块中的 `http.ClientRequest` 对象：本需求使用到了该实例对象的 `req.on` 方法，通过流的方式读取请求体。用法见上方示例。本需求使用到了文件读取操作，具体见上方示例，`Date` 构造函数在本需求中的相关用法请参见下方示例。已经对需要进行响应的请求做出了判断，考生需处理获取请求体数据并响应给客户端。因为已经实现了响应数据的方法 `send`，因此，仅需要将参数传递给该方法即可，参数详见注释。

```
new Date().toLocaleString();//获取该日期对象的字符串
```

4. 在完成 GET 请求的基础上，实现获取响应数据并渲染用户权限列表功能。所需知识点：`axios`、`Vue.onBeforeMount`，详见下方示例和描述。
5. 在完成 PUT 接口并将 GET 请求数据正确渲染的基础上，实现获取响应数据并渲染用户权限列表功能。所需知识点：`axios`、`JSON` 内置对象，详见下方示例和描述。
6. 在完成 POST 接口并将 GET 请求数据正确渲染的基础上，实现获取响应数据并将最新的日志信息渲染到日志查看区域的首位功能。

- 所需知识点：同第五点

#### 1. nodejs 内置模块（`fs`、`http`）

- `fs`：见上文。
- `http`：见上文中关于 `http.ClientRequest` 对象的示例。

#### 2. vue3 生命周期钩子函数 `onBeforeMount`，`setup` 组合式 api

- `onBeforeMount`：

```
//注册一个钩子，在组件被挂载之前被调用。  
function onBeforeMount(callback: () => void): void  
//当这个钩子被调用时，组件已经完成了其响应式状态的设置，但还没有创建 DOM 节点。它即将首次执行 DOM 渲染过程。
```

- `setup`：

- `setup()` 钩子是在组件中使用组合式 API 的入口，通常只在以下情况下使用：

1. 需要在非单文件组件中使用组合式 API 时。

2. 需要在基于选项式 API 的组件中集成基于组合式 API 的代码时。

- 我们可以使用响应式 API 来声明响应式的状态，在 `setup()` 函数中返回的对象会暴露给模板和组件实例。其他的选项也可以通过组件实例来获取 `setup()` 暴露的属性：

```
setup() {  
  const count = ref(0)  
  
  // 返回值会暴露给模板和其他的选项式 API 钩子  
  return {  
    count  
  }  
},
```

### 3. js 内置对象 JSON

- **JSON.stringify()**：该方法将一个 JavaScript 对象或值转换为 JSON 字符串。

语法：

```
JSON.stringify(value); // 类型：object || string  
// 返回：一个表示给定值的 JSON 字符串。
```

- **JSON.parse()**：该方法用来解析 JSON 字符串，构造由字符串描述的 JavaScript 值或对象。

语法：

```
JSON.parse(text); // 类型：string  
// 返回：对应给定 JSON 文本的对象/值。
```

### 4. axios

axios：

介绍：Axios 是一个基于 **promise** 网络请求库，作用于 node.js 和浏览器中。它是 isomorphic 的（即同一套代码可以运行在浏览器和 node.js 中）。在服务端它使用原生 node.js http 模块，而在客户端（浏览端）则使用 XMLHttpRequests。

## 基本用法：

```
// 发送请求
axios({
  method: 'post',//请求方法
  url: '/user/12345',//请求地址
  headers:{
    ...
  },//自定义请求头
  data: {
    ...
  },//作为请求体被发送的数据，仅适用 'PUT', 'POST', 'DELETE 和 'PATCH' 请求方法
  params:{
    ...
  },//与请求一起发送的 URL 参数，必须是一个简单对象或 URLSearchParams 对象
});
```

- 解题代码

```
// node.js
if (req.method === 'GET' && req.url === '/users') {
  // TODO 处理获取文件内容的操作
  let fileContent = '';
  fileContent = fs.readFileSync(dataUrl, 'utf-8');
  send(res, 0, '', eval(fileContent));
}
else if (req.method === 'PUT' && req.url === '/editUser') {
  let body = '';
  req.on('readable', () => {
    let chunk = '';
    if (null !== (chunk = req.read())) {
      body += chunk;
    }
  })
  req.on('end', () => {
    if (body) {
      // TODO 处理更改文件数据并将最新的文件数据响应给客户端
      const resp = JSON.parse(body);
      const userContent = fs.readFileSync(dataUrl, 'utf-8');
      let users = eval(userContent);
      users = users.map(i => {
        if (i.id === resp.params.id) {
          return {
            ...i,
            ...resp.data
          }
        }
        return i
      })
      fs.writeFile(dataUrl, JSON.stringify(users), 'utf8', () => {
        send(res, 0, '', users)
      })
    }
  })
}
```



```

    }
  })
}
else if (req.method === 'POST' && req.url === '/logger') {
  let body = '';
  req.on('readable', () => {
    let chunk = '';
    if (null !== (chunk = req.read())) {
      body += chunk;
    }
  })
  req.on('end', () => {
    if (body) {
      // TODO 处理新增日志
      const info = {
        id: getLoggerId(),
        msg: JSON.parse(body).data,
        time: new Date().toLocaleString(),
      }
      fs.writeFile(loggerUrl, JSON.stringify(info) + '\n', {
        flag: 'a',
        encoding: 'utf-8'
      }, () => {
        send(res, 0, '', info);
      });
    }
  })
}
}

```

```

// index.js
const handleChange = async (e) => {
  if (e.target.tagName !== 'INPUT') {
    return
  }
  // TODO 处理发送请求修改当前用户的权限并更新一条日志记录
  const item = data.userList[data.userList.findIndex((val) => val.id === e.target.dataset.id)]
  const res = await axios.put('/editUser', JSON.stringify({
    data: {
      power: !item.power
    },
    params: {
      id: item.id
    }
  })).then(res => parseRes(res))
  data.userList = res.data;
  const logger = await axios.post('/logger', JSON.stringify({
    data: `超级管理员将用户${item.name}设置为${getPowerText(!item.power)}权限`
  })).then(res => parseRes(res))
  data.loggerList.unshift(logger.data)
}
// TODO 在页面挂载之前请求用户数据并修改对应的响应数据
Vue.onBeforeMount(async () => {
  const res = await axios.get('/users').then(res => parseRes(res))

```

```
data.userList = res.data;
})
```

## 9. 超级英雄联盟

- 解题思路：

1. **HeroList** 组件：

- 组件模板中使用 **v-for** 遍历 **store.HeroList** 数组，生成每个英雄的列表项。
- 每个列表项显示英雄的名称、能力和力量值，并使用绑定表达式 **:disabled="isAddedToTeam(item)"** 判断按钮是否禁用。
- 点击按钮时调用 **addhero(item)** 方法添加英雄到队伍，并根据 **isAddedToTeam(item)** 的返回值显示添加按钮的文本。

2. **TeamList** 组件：

- 组件模板中使用 **v-for** 遍历 **store.team** 数组，生成每个英雄的列表项。
- 每个列表项显示英雄的名称、力量值，并添加一个移除按钮。
- 点击移除按钮时调用 **remove(item)** 方法，从队伍中移除对应的英雄。
- 按钮 "按实力排序" 的点击事件调用 **sorthero()** 方法，对队伍中的英雄按照力量值进行降序排序。
- 使用计算属性 **count** 计算当前队伍的战斗力，即对队伍中每个英雄的力量值求和。

注意：以上是代码的基本思路和功能描述，具体实现还需要确保 Vue 及相关库的正确引入，以及 **useHeroStore** 方法的正确实现。另外，还需要确保 **heroes.json** 文件中包含正确的英雄数据。

- 解题代码

```
// heroList.js
const HeroList = {
  template: `
    <div class="hero-list">
      <h2>可选英雄</h2>
      <ul>
        <li v-for='item in store.HeroList' class="hero-item">
          <span>{{item.name}}</span>
          <span>{{item.ability}}</span>
          <span>{{item.strength}}</span>
```

```

        <button :disabled="isAddedToTeam(item)" @click="addhero(item)">
          {{ isAddedToTeam(item) ? '已添加' : '添加至队伍' }}
        </button>
      </li>
    </ul>
  </div>
  `,
  setup() {
    const {onMounted, } = Vue
    const store = useHeroStore();

    onMounted(async()=>{
      let res = await axios.get('./js/heroes.json');
      store.HeroList = res.data
    })
    const isAddedToTeam = (hero) => {
      let isfind = store.team.find(item=>item.id==hero.id);
      return !!isfind;
    };
    const addhero =(item)=>{
      store.team=[...store.team,item]
    }
    return {
      store,
      addhero,
      isAddedToTeam
    }
  },
};
// TODOEnd

```

```

// TeamList.JS
// TODO: 补全代码，实现目标效果
const TeamList = {
  template: `
    <div class="team-list">
      <h2>我的队伍</h2>
      <ul>
        <li v-for='item in store.team' class="team-item">
          <span>{{item.name}}</span>
          <span>{{item.strength}}</span>
          <button @click="remove(item)">移除</button>
        </li>
      </ul>
      <button @click="sorthero" class="sort-button">按实力排序</button>
      <p class="total-strength">当前队伍战斗力: {{count}} </p>
    </div>
  `,
  setup() {
    const {computed} = Vue

    const store = useHeroStore();
    const count = computed(()=>{
      return store.team.reduce((prev,next)=>prev+next.strength,0)
    })
  }
}

```

```

    })
    const remove = hero => {
      store.team = store.team.filter(item => item.id !== hero.id)
    }
    const sorthero = () => {
      store.team.sort((a, b) => {
        return b.strength - a.strength
      })
    }

    return {store, remove, sorthero, count}
  },
};
// TODOEnd

```

## 10. 万能合成台 - 题解

- 解题思路

为了完成预期功能，我们需要实现的有：合成栏的状态管理，根据当前状态更新可能的目标合成物品名称，合成配方的匹配检查

其中第一和第二条由 `onPickItem` 函数实现，相对来说较为简单，能看懂源码结构就很容易补上：

```

// 监听物品更改事件
function onPickItem(name, pos) {
  // name 为此次修改为的物品，可能为空字符串（通过右下角清空），也可能为物品
  // pos 为一个两个元素的数字数组，分别指示物品的所在横行与所在纵列
  console.log('onPickItem', name, pos);

  // 更新 state
  state[pos[0] - 1][pos[1] - 1] = name;

  // 检查是否有配方与 state 相匹配
  targetItemName = checkRecipe(state) || '';
  console.log('targetItemName', targetItemName);
}

```

下面我们重点看第三条，主要涉及 `checkRecipe` 函数的具体实现。

首先尝试实现一下解析 `3x3` 完整合成表的代码。合成表中如果是 `3x3` 的配方，直接一对一和合成表 `state` 的数据进行匹配就行了：

```

// 检查当前的合成配方是否能够合成某个物品
function checkRecipe(map) {
    // map 为一个 3*3 的二维数组，第一维为物品的每一横行，第二维为每一横行物品下的每个纵列
    // 数组元素均为表示物品类型的字符串

    // 遍历 recipes，检查是否有配方与 map 相匹配
    for (const targetName of Object.keys(window['recipes'])) {
        const recipes = window['recipes'][targetName];

        // 遍历 recipes 中的每一个配方
        for (const recipe of recipes) {
            // 遍历 recipe 中的每一个物品
            let match = true;

            // 匹配每一个物品是否与 map 相同
            for (let row = 0; row < sizeX; row++) {
                for (let col = 0; col < sizeY; col++) {
                    if (recipe[row][col] !== map[row][col]) {
                        match = false;
                        break;
                    }
                }
            }
            if (!match) {
                break;
            }
        }

        if (match) {
            return targetName;
        }
    }
}

```

仅仅实现 3x3 合成表的完整匹配其实是不够的，因为实际合成的物品可能不会占据完整的三行三列的空间。

为了简化问题，我们可以考虑先实现确定合成表“占据空间”的逻辑，也就是简化版的凸包算法。



如图所示，我们应该想办法确定目前的合成表的有效部分占据了多大空间，然后还需要确定偏移：

```

// 首先确定 map 目前布置物品的闭包，即非空物品的位置的四个角落坐标
// 借这个信息，可以确定 map 的偏移量，以便后续的配方匹配
let topLeft = [3, 3];
let bottomRight = [-1, -1];

for (let row = 0; row < 3; row++) {
  for (let col = 0; col < 3; col++) {
    if (map[row][col] !== '') {
      // 更新左上角坐标
      topLeft[0] = Math.min(topLeft[0], row);
      topLeft[1] = Math.min(topLeft[1], col);
      // 更新右下角坐标
      bottomRight[0] = Math.max(bottomRight[0], row);
      bottomRight[1] = Math.max(bottomRight[1], col);
    }
  }
}

// 确定 map 的有效内容大小
const sizeX = bottomRight[0] - topLeft[0] + 1;
const sizeY = bottomRight[1] - topLeft[1] + 1;

```

然后就可以利用 `sizeX` 和 `sizeY`，首先快速判断这个合成表能不能用——如果大小不匹配，那后面的算法当然也就不用跑了，直接确定这个合成表无效：

```

// 在 recipes 的 for 遍历循环中补充这一部分，在正式匹配前运行

// 首先匹配合成表的大小是否与 map 相同
if (recipe.length !== sizeX) {
  continue;
}
if (recipe[0].length !== sizeY) {
  continue;
}

```

然后，在匹配时也应当考虑偏移的影响，确保两边的坐标对得上号：

```

if (recipe[row][col] !== map[row + topLeft[0]][col + topLeft[1]]) {
  match = false;
  break;
}

```

然后问题就解决了。

- 解题代码

```

// 辅助记录当前合成栏内容的全局变量
let state = [
  ['', '', ''],
  ['', '', ''],
  ['', '', '']
];

// 辅助记录当前合成栏的可能目标合成物品
// 每次 onPickItem 执行后, 前端界面会根据此全局变量的值更改图标
// 代码测试时会检查该全局变量是否有更改, 请勿篡改变量名
let targetItemName = '';

// 检查当前的合成配方是否能够合成某个物品
function checkRecipe(map) {
  // map 为一个 3*3 的二维数组, 第一维为物品的每一横行, 第二维为每一横行物品下的每个纵列
  // 数组元素均为表示物品类型的字符串
  console.log('checkRecipe', map);

  // 提示, 可以使用 recipes.js 下定义的一批合成配方进行匹配, 以名为 recipes 的全局变量
  // 导入浏览器全局
  // 如果该配方合法, 需要返回一个合成的目标物品名, 否则返回一个空字符串

  // 首先确定 map 目前布置物品的闭包, 即非空物品的位置的四个角落坐标
  // 借这个信息, 可以确定 map 的偏移量, 以便后续的配方匹配
  let topLeft = [3, 3];
  let bottomRight = [-1, -1];

  for (let row = 0; row < 3; row++) {
    for (let col = 0; col < 3; col++) {
      if (map[row][col] !== '') {
        // 更新左上角坐标
        topLeft[0] = Math.min(topLeft[0], row);
        topLeft[1] = Math.min(topLeft[1], col);

        // 更新右下角坐标
        bottomRight[0] = Math.max(bottomRight[0], row);
        bottomRight[1] = Math.max(bottomRight[1], col);
      }
    }
  }

  // 确定 map 的有效内容大小
  const sizeX = bottomRight[0] - topLeft[0] + 1;
  const sizeY = bottomRight[1] - topLeft[1] + 1;

  // 遍历 recipes, 检查是否有配方与 map 相匹配
  for (const targetName of Object.keys(window['recipes'])) {
    const recipes = window['recipes'][targetName];

    // 遍历 recipes 中的每一个配方
    for (const recipe of recipes) {
      // 遍历 recipe 中的每一个物品
      let match = true;

      // 首先匹配合成表的大小是否与 map 相同
      if (recipe.length !== sizeX) {

```

```

        continue;
    }
    if (recipe[0].length !== sizeY) {
        continue;
    }
    console.log('recipe', recipe)

    // 然后匹配每一个物品是否与 map 相同
    for (let row = 0; row < sizeX; row++) {
        for (let col = 0; col < sizeY; col++) {
            if (recipe[row][col] !== map[row + topLeft[0]][col + topLeft[1]])
        {
                match = false;
                break;
            }
        }
        if (!match) {
            break;
        }
    }

    if (match) {
        return targetName;
    }
}
}

// 监听物品更改事件
function onPickItem(name, pos) {
    // name 为此次修改为的物品，可能为空字符串（通过右下角清空），也可能为物品
    // pos 为一个两个元素的数字数组，分别指示物品的所在横行与所在纵列
    console.log('onPickItem', name, pos);

    // 提示，需要更新全局变量 state、targetItemName

    // 更新 state
    state[pos[0] - 1][pos[1] - 1] = name;

    // 检查是否有配方与 state 相匹配
    targetItemName = checkRecipe(state) || '';
    console.log('targetItemName', targetItemName);
}

```

## 11. 账户验证

- 解题思路

1. 使用createPinia创建了名为"my"的Pinia store，该store包含了Captcha和phoneNumber两个响应式引用（ref）。



2. 在Vue应用中，定义了一个名为`phone`的组件，该组件的模板使用了id为"phone"的`<template>`元素。该组件使用了`emits`选项声明了一个自定义事件`show`，并在`setup`函数中创建了响应式数据和方法。
3. `checkId`方法用于验证用户输入的手机号码和是否勾选了协议。如果验证通过，则生成一个随机的验证码并存储到Pinia store中，然后触发`show`事件，将组件切换为"check"。
4. `getRandom`方法用于生成一个6位的随机验证码。
5. 组件`check`的模板使用了id为"check"的`<template>`元素。在`setup`函数中，创建了响应式数据和方法。
6. `resend`方法用于重新发送验证码，它会生成一个新的随机验证码，并存储到 Pinia store中。
7. 在`onMounted`钩子函数中，对验证码输入框进行了事件监听，实现了按键输入验证码的功能。当用户输入完毕后，会进行验证，如果验证码正确，则触发`show`事件，将组件切换为"success"。组件`success`的模板使用了id为"success"的`<template>`元素，没有定义额外的逻辑。

- 解题代码

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>新增地址</title>
  <link rel="stylesheet" type="text/css" href="./css/index.css" />
  <link rel="stylesheet" href="./css/element-plus@2.3.7/index.css">
  <script src="./js/vue3.global.js"></script>
  <script src="./css/element-plus@2.3.7/index.full.js"></script>
  <script type="importmap">
    {
      "imports":{
        "vue-demi":"./js/index.mjs",
        "vue":"./js/vue.esm-browser.prod.js",
        "pinia":"./js/pinia.esm-browser.js"
      }
    }
  </script>
  <script src="./js/pinia.esm-browser.js" type="module"></script>
</head>

<body>
  <div id="app">
```

```

<div class="header">
  
  <span id="main_title">使用手机号登录</span>
  <span class="blank"></span>
</div>
<component @show='changeShowName' :is="showName"></component>
</div>

<template id="phone">
  <div>
    <ul class="phone">
      <span>输入手机号码</span>
      <li>
        <input type="text" autofocus id="numberInput" v-
model="inputValue"/>
      </li>
      <li>
        <input type="checkbox" name="" id="checkbox" v-model="checked"
/>
        <span>已阅读并同意 <a href="javascript:;">服务协议</a>和 <a
href="javascript:;">隐私保护指引</a></span>
      </li>
      <button id="btn" @click="checkId">下一步</button>
    </ul>
  </div>
</template>

<template id="check">
  <ul class="number">
    <span>输入短信验证码</span>
    <li class="hassend">已向
      <i>{{phoneNumber}}</i>
      发送验证码
    </li>
    <li class="code-container">
      <input type="number" class="code" min="0" max="9" required>
      <input type="number" class="code" min="0" max="9" required>
      <input type="number" class="code" min="0" max="9" required>
      <input type="number" class="code" min="0" max="9" required>
      <input type="number" class="code" min="0" max="9" required>
      <input type="number" class="code" min="0" max="9" required>
    </li>
    <a href="javascript:;" id="resend" @click="resend">重新发送</a>
  </ul>
</template>

<template id="success">
  <div class="success">
    <ul>
      <div>验证成功! </div>
      <div>5s后将自动跳转</div>
    </ul>
  </div>
</template>
</body>

```

```

<script type="module">
  import { createPinia, defineStore } from 'pinia'

  const { createApp, reactive, ref, onMounted, watch, defineProps, toRefs }
= Vue
  const { ElNotification } = ElementPlus

  const useMyStore = defineStore('my', () => {
    let Captcha = ref(0)
    let phoneNumber = ref(0)

    function refreshCaptcha(val) {
      Captcha.value = val
    }
    function refreshPhoneNumber(val){
      phoneNumber.value = val
    }

    return { refreshCaptcha, refreshPhoneNumber, Captcha,phoneNumber }
  })

  const app = createApp({
    // -----这部分内容全部删掉让考生自己写-----
    setup() {
      let data = reactive({
        showName: "phone",
        inputValue: "123",
      })

      const changeShowName = (val) => {
        data.showName = val
      }

      return {
        ...toRefs(data),
        changeShowName
      }
    },
    // -----
  })
  app.use(ElementPlus)
  app.use(createPinia())

  app.component("phone", {
    template: "#phone",
    // -----这部分内容全部删掉让考生自己写-----
    emits: ['show'],
    setup(props, context) {
      const store = useMyStore()
      let data = reactive({
        inputValue: "",
        checked: false
      })
      function checkId() {
        let a = getRamdom()
        if (data.checked){

```

```

    if (/^18[0-9]{9}$/.test(data.inputValue)) {
      store.refreshPhoneNumber(data.inputValue)
      ElNotification({
        title: '发送成功',
        message: `您的验证码为${a}`,
        type: 'success',
        duration: 0
      })
      context.emit('show', 'check')
      store.refreshCaptcha(a)
    } else {
      ElNotification({
        title: '发送失败',
        message: `无效的手机号码`,
        type: 'error',
        duration: 0
      })
    }
  } else {
    ElNotification({
      title: '发送失败',
      message: `请先阅读并同意下方协议`,
      type: 'error',
      duration: 0
    })
  }
}

const getRandom = () => {
  let str = '';
  for (let i = 0; i < 6; i++) {
    str += parseInt(Math.random() * 10);
  }
  return str
}

return {
  ...toRefs(data),
  checkId
}
}
// -----

})
app.component("check", {
  template: "#check",
  // -----这部分内容全部删掉让考生自己写-----
  setup(props, context) {
    let data = reactive({
      phoneNumber: ""
    })
    const store = useMyStore()
    data.phoneNumber = store.phoneNumber.substr(0, 3) + "*****" +
store.phoneNumber.substr(9, 12)
    const resend = () => {
      let a = getRandom()
      ElNotification({

```

```

        title: '发送成功',
        message: `您的验证码为${a}`,
        type: 'success',
        duration: 0
    })
    store.refreshCaptcha(a)
}

```

```

onMounted(() => { // 组件完成初始渲染并创建 DOM 节点后运行代码
    const codes = document.querySelectorAll('.code')
    codes[0].focus()
    let typed = [false, false, false, false, false, false]
    codes.forEach((code, idx) => {
        code.addEventListener('keydown', (e) => {
            if (e.key >= 0 && e.key <= 9) {
                typed[idx] = true
                setTimeout(() => {
                    codes[idx + 1] && codes[idx + 1].focus()
                    if (typed.indexOf(false) == -1) {
                        let number = ""
                        for (let i = 0; i < codes.length; i++) {
                            number += codes[i].value
                        }
                        if (number == store.Captcha) {
                            ElNotification({
                                title: '验证成功',
                                message: `欢迎回来`,
                                type: 'success',
                                duration: 0
                            })
                            context.emit('show', 'success')
                        } else {
                            ElNotification({
                                title: '验证失败',
                                message: `您输入的验证码有误`,
                                type: 'error',
                                duration: 0
                            })
                            for (let i = 0; i < codes.length; i++) {
                                codes[i].value = ''
                                codes[0].focus()
                                for (let i in typed) {
                                    typed[i] = false
                                }
                            }
                        }
                    }
                }, 10)
            } else if (e.key === 'Backspace') {
                typed[idx] = false
                setTimeout(() => {
                    codes[idx - 1] && codes[idx - 1].focus()
                }, 10)
            }
        })
    })
})

```

```
    })

    const getRandom = () => {
      let number = "1234567890"
      let result = ""
      for (let i = 0; i < 6; i++) {
        result += number[Math.floor(Math.random() * 10)]
      }
      return result
    }
    return {
      ...toRefs(data),
      resend
    }
  }
  // -----
})
app.component("success", {
  template: "#success"
})
app.mount('#app')
</script>
</html>
```