十五届模拟三

1. 创意广告牌

- 考察：背景图片设置以及 css3 中的圆角设置、倾斜设置

```
.someclass {
  /* TODO: 待补充代码  设置圆角 10px，背景图片为woodiness.jpg  */
  background-image: url("../images/woodiness.jpg");
  border-radius: 10px;
  /* TODO: 待补充代码   上面两个角是圆角，下面两个角是直角  元素 x 轴倾斜 20度*/
  border-radius: 15px 15px 0 0;
  transform: skew(-20deg);
}
```

- 上面圆角的写法顺序为 左上角、右上角、右下角和左下角，另外的写个设置多个圆角的多个写法：

```css
.some-class {
  /* TODO: 待补充代码 上面两个角是圆角，下面两个角是直角 */
  border-top-left-radius: 10px; /* 设置左上角为圆角 */
  border-top-right-radius: 10px; /* 设置右上角为圆角 */
  border-bottom-left-radius: 0; /* 设置左下角为直角 */
  border-bottom-right-radius: 0; /* 设置右下角为直角 */
  /* 其他样式... */
}
```

2. 原子化 css

考察：`css` 属性选择器、`flex` 布局

- 解题思路

从题目需求可以想到本题需要使用 CSS 的属性选择器，参考：属性选择器 - CSS：层叠样式表 | MDN。由于本题需要在一个属性下有多个值，所以需要使用 `[attr~=value]`，这个表示带有以 *attr* 命名的属性的元素，并且该属性是一个以空格作为分隔的值列表，其中至少有一个值为 *value*。

- 解题代码

```css
[flex~="~"] {
  display: flex;
}
[flex~="col"] {
  flex-direction: column;
}
```

3. 神秘咒语

- 考察 `axios` 的使用

```
axios.defaults.headers.common[
  "Authorization"
] = `2b58f9a8-7d73-4a9c-b8a2-9f05d6e8e3c7`;
```

- 也可以使用下面的方式，使用拦截器统一设置

```
axios.interceptors.request.use(
  (config) => {
    config.headers["Authorization"] = "2b58f9a8-7d73-4a9c-b8a2-
9f05d6e8e3c7";
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);
```

4. 朋友圈

- 考察：`localstorage` 存储、函数封装、dom 操作

```
// 1. 用户没有输入任何内容，disable按钮
// 2. 用户输入内容后点击发表，清空文本框内容和缓存
// 3. 用户改变文本框内容，缓存用户当前修改

// 页面初始化时，提取localStorage中的缓存内容，如果没有缓存要disable按钮
document.addEventListener("DOMContentLoaded", () => {
  // TODO: 学生需要补充下面这部分代码
  const savedText = localStorage.getItem("savedText");
  if (savedText) {
    document.getElementById("text").value = savedText;
  } else {
    document.getElementById("post").setAttribute("disabled", "disabled");
  }
});

// 当文本框输入内容改变时，动态地设置localStorage缓存，并根据有没有文本改变按钮状态
// 此处使用了防抖函数，避免太过频繁地更新缓存
document.getElementById("text").addEventListener(
  "input",
  debounce(() => {
    // 提示正在保存中
    document.getElementById("prompt").textContent = "正在保存中...";

    // TODO: 学生需要补充下面这部分代码
    const text = document.getElementById("text").value;
    localStorage.setItem("savedText", text);
    if (text) {
      document.getElementById("post").removeAttribute("disabled");
    } else {
      document.getElementById("post").setAttribute("disabled", "disabled");
    }

    // 过一段时间后提示保存完成，模拟上传数据至后台的效果
    setTimeout(() => {
      document.getElementById("prompt").textContent = "内容已保存";
```

```
    }, 750);
  }, 200)
);

// 当用户点击发表时，在列表中添加一条新发布内容，清空缓存并disable按钮
document.getElementById("post").addEventListener("click", () => {
  const content = document.getElementById("text").value;
  const element = createContent(content);
  document.querySelector(".contents").appendChild(element);
  document.getElementById("prompt").textContent = "";

  // TODO: 学生需要补充下面这部分代码
  localStorage.removeItem("savedText");
  document.getElementById("text").value = "";
  document.getElementById("post").setAttribute("disabled", "disabled");
});

// 防抖工具函数
function debounce(fn, delay) {
  let timer;
  return function (...args) {
    if (timer) clearTimeout(timer);
    timer = setTimeout(() => fn.apply(this, args), delay);
  };
}

// 创建发布内容的函数
function createContent(content) {
  const div = document.createElement("div");
  const d = new Date();
  const deleteBtn = document.createElement("button");
  deleteBtn.textContent = "删除";
  deleteBtn.addEventListener("click", () => {
    div.remove();
  });
  div.innerHTML = `<div><span class="content">${content}</span><span
class="date">${d.toLocaleString()}</span></div>`;
  div.appendChild(deleteBtn);
  return div;
}
```

5. 美食蛋白质解密

考察： echarts、vue3、axios 请求

```
<div
  class="protein-item"
  v-for="(item, index) in data.slice(1)"
  :key="item.name"
>
  {{item.name}} {{item.value}}
```

```
</div>
<script>
  async function fetchData() {
    const response = await fetch(MockURL);
    const result = await response.json();
    console.log(result);
    data.value = [{ name: "表头", icon: "none" }, ...result];
    echartsInit(data.value);
  }
</script>
```

6. 营业状态切换

```
function useToggle(state) {
  const _state = ref(state); // 使用ref函数创建响应式数据
  const toggle = () => {
    _state.value = !_state.value; // 切换状态
  };
  return [_state, toggle]; // 返回状态和切换函数
}
```

7. 小说阅读器

- 解题思路

本题考察的是 `fs` 文件读写、`axios` 发送 `ajax` 请求、`vue3` 基础。

1. 实现文件读写功能：首先需要知道文件读写的方法是`readFile` 和 `writeFile` ，在这里我们使用他的同步方法 `readFileSync` 和 `writeFileSync` 。然后对读取出来的数据进行逐步解析，并转换为 json 格式返回。最后根据返回的数据写入文件里。
    - 文件读取详细步骤：

      1.根据观察文件：第一行为书名，每一章使用----------隔开，n 章会有一个卷名（第 n 卷 xx），章节名（第 n 章 xx）后为内容，并且所有的数据都会使用换行隔开。

      2.先使用 `fs.readFileSync(file, 'UTF-8')` 读取文件的数据。

3.使用 `split` 方法将读取出来的数据，进行字符串根据换行分割为字符串数组。

4.那么字符串数组的第一行为书名，并且使用 `trim()` 方法去除首尾空格。

5.将剩下的字符串数组进行循环，判断每行内容是否为空，根据正则匹配卷、章，按照格式添加到数据里，并去除首尾空格和多余符号。

6.匹配到章节名之后，后面匹配到的都是这一章的内容，直到------结束符号为止，再把这一整章的内容添加到数据里。

7.所以数据进行匹配添加后，即可返回数据。

- 文件写入：使用 `fs.writeFileSync(file,data,'UTF-8')` 将数据写入即可。

```
const fs = require("fs");
//读取txt小说文件
let readFilePath = "./run/book.txt";
let writeFilePath = "./run/book.json";
let options = "UTF-8";

//请按照要求修改一下两个函数内容。

//读取txt小说文件,并按对应数据格式返回。
const readFile = (file) => {
  //TODO: 待补充代码
  try {
    let result = { name: "", data: [] }; //返回的数据
    let pageName = ""; //章节名
    let content = []; //每章内容
    const res = fs.readFileSync(file, options);
    let data = res.split("\r\n"); //根据换行符分隔每一行
    result.name = data[0]; //第一行为书名。
    data.shift(); //弹出第一行
    data.forEach((v) => {
      if (v.trim() != "") {
        //判断是否为空行
        if (v.match(/第.+卷/)) {
          //匹配卷
          result.data.push({
            isRoll: true,
            title: v.replaceAll("-", "").trim(), //去掉卷名里面的多余字符---并去掉首尾空格
          });
        } else if (v.match(/第\d+章/)) {
          //匹配章
          pageName = v.trim();
        } else if (v.includes("----")) {
          //判断是否为每章结束符，需要大于3个-
          if (pageName != "") {
```

```
            //判断是否为空
            result.data.push({ title: pageName, content }); //加入数据
            pageName = ""; //清空一下
            content = [];
          }
        } else {
          content.push(v.trim());
        }
      }
    });
    return JSON.stringify(result); //转为json字符串返回。
  } catch (err) {
    return null;
  }
};
//写入json文件中
const writeFile = (file, data) => {
  //TODO: 待补充代码
  fs.writeFileSync(file, data, options);
};

//以下内容请勿修改，否则将影响结果
let data = readFile(readFilePath);
if (data != null) writeFile(writeFilePath, data);

module.exports = {
  writeFile,
  readFile,
};
```

2. 添加文件读写操作完成后务必在终端运行 `node run/index.js` 的操作。

3. 使用 `axios.get(path)` 发起请求，并设置 `vue` 的 `data` 值。

```
axios.get("./run/book.json").then((res) => {
  let data = JSON.parse(JSON.stringify(res.data));
  this.bookName = data.name;
  this.chapters = data.data;
});
```

4. 根据计算属性 `selectedChapter` 的代码来看，修改 `activeChapter` 的值即可达到切换章节的效果，`next` 方法的参数 `value`的值为 1 和 -1 ，那么只要根据现有的 `activeChapter` 加上 `value` 即可修改，再判断切换之后的章节内容是否为卷，再跳过即可。最后判断一下要修改的值是不是在有没有超出或越界即可。

```
next(value) {
      let index = this.activeChapter + value;//修改后切换的章节索引
      if (index < 1 || index >= this.chapters.length) return;//判断是否超出章节。
```

```
        if (this.chapters[index].isRoll) {//判断切换的索引，是否为卷名
          index += value;//跳过卷名。
        }
        this.activeChapter = index;
      }
```

- 解题代码

```
/** - run/index.js */
const fs = require("fs");
//读取txt小说文件
let readFilePath = "./run/book.txt";
let writeFilePath = "./run/book.json";
let options = "UTF-8";

//请按照要求修改一下两个函数内容。

//读取txt小说文件,并按对应数据格式返回。
const readFile = (file) => {
  //TODO: 待补充代码
  try {
    let result = { name: "", data: [] };
    let pageName = "";
    let content = [];
    const res = fs.readFileSync(file, options);
    let data = res.split("\r\n");
    result.name = data[0];
    data.shift();
    data.forEach((v) => {
      if (v.trim() != "") {
        if (v.match(/第.+卷/)) {
          result.data.push({
            isRoll: true,
            title: v.replaceAll("-", "").trim(),
          });
        } else if (v.match(/第\d+章/)) {
          pageName = v.trim();
        } else if (v.includes("----")) {
          if (pageName != "") {
            result.data.push({ title: pageName, content });
            pageName = "";
            content = [];
          }
        } else {
          content.push(v.trim());
        }
      }
    });
    return JSON.stringify(result);
  } catch (err) {
    return null;
  }
};
//写入json文件中
```

```
const writeFile = (file, data) => {
  //TODO: 待补充代码
  fs.writeFileSync(file, data, options);
};

//以下内容请勿修改，否则将影响结果
let data = readFile(readFilePath);
if (data != null) writeFile(writeFilePath, data);

module.exports = {
  writeFile,
  readFile,
};
```

```
<!-- mycomponent.vue -->
<template>
  <div class="content">
    <el-card v-show="drawer" class="pages">
      <h3>章节列表</h3>
      <el-menu
        :default-active="activeChapter"
        active-text-color="#ee4545"
        @select="handleChapterSelect"
        background-color="#faf9f4"
      >
        <block v-for="(chapter, index) in chapters" :key="index">
          <h4 v-if="chapter.isRoll" class="juan">{{ chapter.title }}</h4>
          <el-menu-item v-else :index="index" class="mitem">
            {{ chapter.title }}
          </el-menu-item>
        </block>
      </el-menu>
    </el-card>
    <div class="main">
      <div v-if="selectedChapter">
        <div
          style="
            display: flex;
            align-items: center;
            justify-content: space-between;
          "
          v-if="activeChapter == 1"
        >
          <h2 class="book">{{ bookName }}</h2>
          <span>引用小说为程序自动生成</span>
        </div>
        <p style="font-size: 22px; font-weight: 500" class="title">
          {{ selectedChapter.title }}
        </p>
        <p
          id="ptext"
          v-for="(item2, index2) in selectedChapter.content"
          :key="index2"
          style="
            font-size: 18px;
```

```
            font-weight: 400;
            margin: 20px;
            color: #555;
            line-height: 35px;
          "
        >
              {{ item2 }}
        </p>

        <div class="tools">
          <div class="tool pre" @click="next(-1)">上一章</div>
          <div class="tool" @click="handleDrawer()">目录</div>
          <div class="tool after" @click="next(+1)">下一章</div>
        </div>
      </div>
      <div v-else>
        <p>请从章节列表中选择章节</p>
      </div>
    </div>
    <div class="list-icon" @click="handleDrawer()">
      <div class="icons">
        <span class="square"> </span>
        <span class="square"> </span>
        <span class="square"> </span>
        <span class="square"> </span>
      </div>
      <div>目录</div>
    </div>
  </div>
</template>

<script>
  module.exports = {
    name: "myComponent",
    data() {
      return {
        bookName: "", //书名
        chapters: [], //章节列表与内容
        activeChapter: 1, //当前选中章节
        drawer: false, //是否显示章节列表。
      };
    },
    computed: {
      selectedChapter() {
        return this.chapters[this.activeChapter];
      },
    },
    methods: {
      handleChapterSelect(index) {
        this.activeChapter = index;
        this.handleDrawer();
      },
      handleDrawer() {
        this.drawer = !this.drawer;
      },
      //切换章节，需跳过卷名。
      //value为 -1 或 1
```

```
      next(value) {
        // TODO: 待补充代码
        let index = this.activeChapter + value;
        if (index < 1 || index >= this.chapters.length) return;
        if (this.chapters[index].isRoll) {
          index += value;
        }
        this.activeChapter = index;
      },
    },
    //通过axios发起请求json数据，并渲染界面。
    created() {
      // TODO: 待补充代码
      axios.get("./run/book.json").then((res) => {
        let data = JSON.parse(JSON.stringify(res.data));
        this.bookName = data.name;
        this.chapters = data.data;
      });
    },
  };
</script>
```

8. 嘟嘟购物

考察： element-plus、pinia、vue3

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="./css/element-plus@2.3.7/index.css">
  <link rel="stylesheet" href="./css/index.css">
  <script src="./lib/vue.min.js"></script>
  <script src="./css/element-plus@2.3.7/index.full.js"></script>
  <script src="./lib/axios.js"></script>
  <script src="./lib/vueDemi.js"></script>
  <script src="./lib/pinia.min.js"></script>
  <script src="./js/store.js"></script>
  <title>嘟嘟购物</title>
</head>

<body>
  <div id="app">
    <div class="c-container">
      <div class="w">
        <div class="cart-filter-bar">
          <em>全部商品</em>
        </div>
        <!-- 购物车主要核心区域 -->
```

```
        <div class="cart-warp">
          <div class="cart-thead">
            <div class="t-checkbox">
            </div>
            <div class="t-goods">商品</div>
            <div class="t-price">单价</div>
            <div class="t-num">数量</div>
            <div class="t-sum">小计</div>
            <div class="t-action">操作</div>
          </div>
          <!-- 商品详细模块 -->
          <div class="cart-item-list">
            <div class="cart-item check-cart-item" v-for="item in showList">
              <div class="p-checkbox">
                <input type="checkbox" name="" id="" class="j-checkbox"
:checked="item.checked"
                  @change="change(item.id)">
              </div>
              <div class="p-goods">
                <div class="p-img">
                  <img :src="item.image" alt="">
                </div>
                <div class="p-msg">{{item.name}}</div>
              </div>
              <div class="p-price">{{item.price}}</div>
              <div class="p-num">
                <div class="quantity-form">
                  <a href="javascript:;" class="decrement"
@click="decrement(item.id)">-</a>
                  <input type="text" :value="item.num" class="itxt">
                  <a href="javascript:;" class="increment"
@click="increment(item.id)">+</a>
                </div>
              </div>
              <div class="p-sum">￥{{item.total}}</div>
              <div class="p-action"><a href="javascript:;">删除</a></div>
            </div>
          </div>
          <!-- 结算模块 -->
          <div class="cart-floatbar">
            <div class="select-all">
            </div>
            <div class="toolbar-right">
              <div class="amount-sum">已经选<em>{{totalCount}}</em>件商品
</div>
              <div class="price-sum">总价： <em>￥{{totalPrice}}</em></div>
              <div class="btn-area" @click="settle">去结算</div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
<script type="module">

  const { createApp, onMounted, reactive, toRefs } = Vue
```

```
  const { createPinia } = Pinia
  const { ElMessage, ElNotification } = ElementPlus
  const app = Vue.createApp({
    setup() {
      // TODO:补充代码，实现目标效果
      let data = reactive({
        showList: [],
        totalCount: 0,
        totalPrice: 0
      })

      const goodsStore = useFlowerStore()
      onMounted(() => {
        axios({
          url: "./js/goods.json"
        }).then(res => {
          res.data.forEach(item => {
            if (goodsStore.store.find(good => good.id == item.id).stock !=
0) {
              item.num = 1
              item.total = item.num * item.price
              item.checked = false
              data.showList.push(item)
            }
          })
        })
      })

      const decrement = (id) => {
        data.showList.forEach(item => {
          if (item.id == id && item.num > 1) {
            item.num--
            item.total = (item.num * item.price).toFixed(2)
          }
        })
        refresh()
      }

      const increment = (id) => {
        data.showList.forEach(item => {
          if (item.id == id) {
            item.num++
            item.total = (item.num * item.price).toFixed(2)
          }
        })
        refresh()
      }

      const change = (id) => {
        data.showList.forEach(item => {
          if (item.id == id) {
            item.checked = !item.checked
          }
        })
        refresh()
      }
```

```
      const refresh = () => {
        data.totalCount = 0
        data.totalPrice = 0
        data.showList.forEach(item => {
          if (item.checked) {
            data.totalCount += item.num
            data.totalPrice =
(parseFloat(parseFloat(data.totalPrice).toFixed(2)) +
parseFloat(item.total)).toFixed(2)

            if (data.totalPrice.toString().split(".")[1].length != 2) {
              data.totalPrice += "0"
            }
          }
        })
      }


      const settle = ()=>{
        let all = []
        data.showList.forEach(item=>{
          if(item.checked == true){
            all.push(goodsStore.fetchInventory(item.name))
          }
        })
        if(all.length == 0){
          ElMessage({
            message: '请至少选择一件商品',
            type: 'warning',
          })
          return
        }

        Promise.all(all).then(res=>{
          let result = true
          for(let i =0;i<res.length;i++){
            if (data.showList[i].num > res[i]) {
              result = false
              ElMessage({
                message: `结算失败，${data.showList[i].name}库存不足`,
                type: 'error',
              })
              return
            }
          }

          if(result){
            ElMessage({
              message: '结算成功',
              type: 'success',
            })
          }

        }).catch(error=>{
          ElMessage({
            message: `结算失败，${error}`,
            type: 'error',
```

```
          })
        })

        // data.showList.forEach()
      }

      return {
        ...toRefs(data),
        decrement,
        increment,
        change,
        settle
      }

      // TODOEnd
    },
  });
  app.use(ElementPlus)
  app.use(createPinia())
  app.mount('#app');

</script>

</html>
```

9. 冰岛人

- 知识点 本题主要考察考生的逻辑思维能力以及使用 `js` 处理数据的能力。 以下是本题中主要使用到的 `js` 处理数据的方法。
- **String.prototype.substr()**
- **Array.prototype.map()**
- **String.prototype.split()**
- **Array.prototype.find()**
- **String.prototype.slice()**

1. String.prototype.substr()

`substr()` 方法返回一个字符串中从指定位置开始到指定字符数的字符。 示例：

```
var str = "abcdefghij";

console.log("(1,2): "    + str.substr(1,2));   // (1,2): bc
console.log("(-3,2): "   + str.substr(-3,2));  // (-3,2): hi
console.log("(-3): "     + str.substr(-3));    // (-3): hij
```

```
console.log("(1): "      + str.substr(1));     // (1): bcdefghij
console.log("(-1): "      + str.substr(1));     // (-1): j
console.log("(-20, 2): " + str.substr(-20,2)); // (-20, 2): ab
console.log("(20, 2): "  + str.substr(20,2));  // (20, 2):
```

关于 `String.prototype.substr()` 的更多知识，可以查看 MDN 官方文档 。

2. Array.prototype.map()

`map()` 方法创建一个新数组，这个新数组由原数组中的每个元素都调用一次提供的函数后的返回值组成。

示例：

```
const kvArray = [
  { key: 1, value: 10 },
  { key: 2, value: 20 },
  { key: 3, value: 30 },
];

const reformattedArray = kvArray.map(item=>item.key);

console.log(reformattedArray); // [1,2,3]
```

关于 `Array.prototype.map()` 的更多知识，可以查看 MDN 官方文档 。

3. String.prototype.split()

`split()` 方法接受一个模式，通过搜索模式将字符串分割成一个有序的子串列表，将这些子串放入一个数组，并返回该数组。

示例：

```
const str = "abc def ghi";
console.log(str.split(" "));  // ["abc","def","ghi"]
```

关于 `String.prototype.split()` 的更多知识，可以查看 MDN 官方文档 。

4. Array.prototype.find()

`find()` 方法返回数组中满足提供的测试函数的第一个元素的值。否则返回 undefined。

示例：

```
const inventory = [
  { name: "apples", quantity: 2 },
  { name: "bananas", quantity: 0 },
  { name: "cherries", quantity: 5 },
];

function isCherries(fruit) {
  return fruit.name === "cherries";
}

console.log(inventory.find(isCherries));
// { name: 'cherries', quantity: 5 }
```

关于 `Array.prototype.find()` 的更多知识，可以查看 MDN 官方文档 。

5. String.prototype.slice()

`slice()` 方法提取某个字符串的一部分，并返回一个新的字符串，且不会改动原字符串。

示例：

```
var str1 = 'The morning is upon us.', // str1 的长度 length 是 23。
    str2 = str1.slice(1, 8),
    str3 = str1.slice(4, -2),
    str4 = str1.slice(12),
    str5 = str1.slice(30);
console.log(str2); // 输出：he morn
console.log(str3); // 输出：morning is upon u
console.log(str4); // 输出：is upon us.
console.log(str5); // 输出：""
```

关于 `String.prototype.slice()` 的更多知识，可以查看 MDN 官方文档 。

在掌握上述本题主要使用的处理数据的方法后，这道题的答案就信手拈来啦！

- 解题思路

题中要求函数根据几种不同的情况返回不同的字符串，因此我们要把代码分三部分来处理。

1. 判断名字是否在名单内

要判断名字是否在名单内，我们首先需要从数据里面拿到所有人的名字，然后判断传入的两个参数是否在里面即可。

该部分代码为：

```
const names = data.map(item=>item.givenName)
if(names.indexOf(name1) == -1 || names.indexOf(name2) == -1){
        return "NA"
}
```

在上述代码中，我们使用数组的 `map` 方法返回所有的人的姓名，保存在 `names` 这个变量中，然后使用数组的 `indexOf()` 方法判断 `name1` 或 `name2` 是否在 `names` 中，不在的话直接返回字符串 `NA` 。

2. 判断两人是否同性

要判断两人是否是同性，我们要从 `data` 当中拿到这两人的 `familyName` ，根据其后缀判断即可。为了方便后续操作，我们对 `data` 做处理，使其子元素都加上 `sex` 这个 key 值用来表示当前这个人的性别，然后判断的时候直接拿其 `sex` 值做对比即可。 该部分代码为：

```
let givenName1 = name1.split(" ")[0]
let givenName2 = name2.split(" ")[0]

data = data.map((item) => {
    let sex = 0; // 1为男，-1为女
    if (
      item.familyName.substr(-1) == "m" ||
      item.familyName.substr(-4) == "sson"
    ) {
      sex = 1;
    } else if (
      item.familyName.substr(-1) == "f" ||
      item.familyName.substr(-7) == "sdottir"
    ) {
      sex = -1;
    }
    return {
      givenName: item.givenName,
      familyName: item.familyName,
      sex,
    };
  });

if(data.find((item) => item.givenName == givenName1).sex ==
```

```
    data.find((item) => item.givenName == givenName2).sex){
        return "Whatever"
    }
```

在上述代码中，我们先使用 `split()` 方法拿到两个人的名字，然后给 `data` 每个元素都加了 `sex` 值用来表示性别，在判断的时候使用 `find()` 方法拿到对象的 `sex` 值做对比即可，如果是同性，则返回字符串 `Whatever` 。

3. 判断其五代以内是否有公共祖先

要判断五代以内是否有公共祖先，我们要分别拿到两个人的祖先以及都是他们的第几代祖先，然后找他们公共祖先，由于题中要求两人的公共祖先必须比任何一方的曾祖父辈分高，因此我们要拿到公共祖先的辈数小的那个做判断，如果其辈数大于等于 4 ，就满足了比任何一方的曾祖父辈分高这一条件，返回相应的字符串即可。该部分代码如下：

```
let zupu1 = find(givenName1, 0);
let zupu2 = find(givenName2, 0);
console.log(zupu1);
console.log(zupu2);
let x = [];

for (let i of Object.keys(zupu1)) {
  for (let j of Object.keys(zupu2)) {
    if (i == j) {
      x.push(zupu1[i]);
      x.push(zupu2[j]);
    }
  }
}
if (Math.min(...x) >= 4) {
  return "Yes";
} else {
  return "No";
}

function find(name, generation) {
    let result = {};
    function find1(name, generation) {
      result[name] = generation;
      let familyName = data.find((item) => item.givenName == name);
      if (!familyName) {
        return result;
      }

      familyName = familyName.familyName;
      if (
        familyName.indexOf("sson") != -1 ||
        familyName.indexOf("sdottir") != -1
      ) {
        if (familyName.indexOf("sson") != -1) {
          find1(familyName.slice(0, familyName.length - 4), ++generation);
```

```
        } else {
          find1(familyName.slice(0, familyName.length - 7), ++generation);
        }
      } else {
        return result;
      }
    }
    find1(name, generation);
    return result;
  }
```

在上述代码中，我们分别定义 `zupu1` 与 `zupu2` 来分别表示 `name1` 与 `name2` 的祖先情况。我们定义了一个函数 `find()`，在这个函数中，有个函数 `find1()`，`find1()` 作用是对 `data` 中的数据进行递归，不停的去找所传入名字的祖先元素，并以 key 为祖先名，value 为向上的第几辈的形式保存在变量 result 中，在进行递归后，result 中就是要查找的那个人的祖先情况，并由 `find()` 返回。

在经过 `find()` 的处理后，`zupu1` 与 `zupu2` 就分别是 `name1` 与 `name2` 的祖先情况了。然后我们从中找到共同祖先，并将他们的辈数都放在数组 `x` 中，然后使用 `Math.min()` 方法拿到数组中最小的那个值，这个值就是公共祖先的小的那一方的辈数，拿该值与 4 做对比，大于等于 4 证明满足情况，返回字符串 `Yes`，否则返回字符串 `No` 即可。

综上，本题的完整代码为：

```
function marry(data, name1, name2) {
  data = data.map((item) => {  // 对data做处理，给每一项加上sex这一值用来标识性别
    let sex = 0; // 1为男，-1为女
    if (
      item.familyName.substr(-1) == "m" ||  // 使用substr方法拿到名字的后缀
      item.familyName.substr(-4) == "sson"
    ) {
      sex = 1;
    } else if (
      item.familyName.substr(-1) == "f" ||
      item.familyName.substr(-7) == "sdottir"
    ) {
      sex = -1;
    }
    return {   // 加上sex这一值并返回
      givenName: item.givenName,
      familyName: item.familyName,
      sex,
    };
  });

  let allName = data.map((item) => item.givenName);  // 拿到data每一项的
givenName 并返回
  let givenName1 = name1.split(" ")[0];  // 拿到传入的name1的givenName
```

```
  let givenName2 = name2.split(" ")[0];  // 拿到传入的name2的givenName

  if (allName.indexOf(givenName1) == -1 || allName.indexOf(givenName2) ==
-1) {  // 判断传入的名字在data中是否存在
    return "NA";
  } else if (  // 判断是否是同性别
    data.find((item) => item.givenName == givenName1).sex ==
    data.find((item) => item.givenName == givenName2).sex
  ) {
    return "Whatever";
  } else {
    let zupu1 = find(givenName1, 0);  // 拿到name1的祖先情况
    let zupu2 = find(givenName2, 0);  // 拿到name2的祖先情况
    let x = [];  // 用来保存祖先辈数

    for (let i of Object.keys(zupu1)) {  // 遍历去找公共祖先
      for (let j of Object.keys(zupu2)) {
        if (i == j) {
          x.push(zupu1[i]);   // 将其辈数放入x中
          x.push(zupu2[j]);
        }
      }
    }

    if (Math.min(...x) >= 4) {  // 拿到小的那一方的倍数与4做对比
      return "Yes";
    } else {
      return "No";
    }
  }

  function find(name, generation) {
    let result = {};   // 定义变量用于存储倍数信息
    function find1(name, generation) {
      result[name] = generation;  // key为名字，value为辈数
      let familyName = data.find((item) => item.givenName == name);  // 从
data中找givenName为传入name的那一项
      if (!familyName) {  // 如果为undefined证明找完了
        return result;
      }

      familyName = familyName.familyName;  // 拿到givenName为name的人的
familyName
      if (  // 如果有sson或sdottir这一后缀证明有父亲
        familyName.indexOf("sson") != -1 ||
        familyName.indexOf("sdottir") != -1
      ) {
        if (familyName.indexOf("sson") != -1) {
          // 通过slice方法从姓氏中拿到其父亲的名字，并让辈数加一，然后接着去找
          find1(familyName.slice(0, familyName.length - 4), ++generation);
        } else {
          find1(familyName.slice(0, familyName.length - 7), ++generation);
        }
      } else {
        return result;  // 如果没有这个后缀，证明找到祖宗了，不能往下找了，直接返回结
果即可
      }
```

```
    }
    find1(name, generation);  // 调用find1方法开始递归寻找祖先
    return result;  // 把结果返回
  }
}
```

10. 这是一个浏览器

- 知识点

本题考察的知识点是：

- 对 js 中类的使用
- js 中函数的 bind 方法
- js 中元素的 insertAdjacentHTML 方法
- arguments 对象
- element.classList
- 模板字符串

1. js 中的类

如果对于 js 中类的知识还不了解，可以查阅 MDN 官方文档 ，这里不再阐述。

2. js 中的函数 bind 方法

js中函数的 `bind()` 方法返回一个新的函数，在 `bind()` 被调用时，这个新函数的 `this` 被指定为 `bind()` 的第一个参数，而其余参数将作为新函数的参数，供调用时使用。比如：

```
function add(x, y, z) {
  return this.x + this.y + this.z;
}

const obj = {
  x: 1,
  y: 2,
  z: 3,
};

const add1 = add.bind(obj, 1, 2, 3); // bind 会返回一个新的函数
console.log(add1()); // 执行新的函数，输出 6
```

关于 `bind()` 的更多知识，可以查看 MDN 官方文档 。

3. js 中元素的 insertAdjacentHTML 方法

js中元素的 `insertAdjacentHTML` 方法将指定的文本解析为 Element 元素，并将结果节点插入到 DOM 树中的指定位置。它不会重新解析它正在使用的元素，因此它不会破坏元素内的现有元素。这避免了额外的序列化步骤，使其比直接使用 innerHTML 操作更快。其使用格式为：

```
element.insertAdjacentHTML(position, text);
```

其中，position 表示插入内容相对于元素的位置，并且必须是以下字符串之一：

- 'beforebegin'：元素自身的前面。
- 'afterbegin'：插入元素内部的第一个子节点之前。
- 'beforeend'：插入元素内部的最后一个子节点之后。
- 'afterend'：元素自身的后面。

text是要被解析为 HTML 或 XML 元素，并插入到 DOM 树中的 DOMString。

例如：

```
// 原为 <div id="one">one</div>
var d1 = document.getElementById('one');
d1.insertAdjacentHTML('afterend', '<div id="two">two</div>');

// 此时，新结构变成：
// <div id="one">one</div><div id="two">two</div>
```

关于 `insertAdjacentHTML` 的更多知识，可以查看 MDN 官方文档 。

4. arguments 对象

`arguments` 是一个对应于传递给函数的参数的类数组对象。 `arguments` 对象是所有（非箭头）函数中都可用的局部变量。你可以使用 `arguments` 对象在函数中引用函数的参数。此对象包含传递给函数的每个参数，第一个参数在索引 0 处。例如，如果一个函数传递了三个参数，你可以以如下方式引用他们：

```
arguments[0]
arguments[1]
arguments[2]
```

参数也可以被设置：

```
arguments[1] = 'new value';
```

关于`arguments` 的更多知识，可以查看 MDN 官方文档 。

5. element.classList

`element.classList` 是一个只读属性，返回一个元素 class 属性的动态 DOMTokenList 集合。这可以用于操作 class 集合。

相比将 element.className 作为以空格分隔的字符串来使用，classList 是一种更方便的访问元素的类列表的方法。

这里列举一些使用 classList 的常用操作：

- 给元素添加类名

```
element.classList.add('active')  // 给元素添加类名 active
```

- 移除元素类名

```
element.classList.remove('active')  // 移除元素的 active 这一类名
```

- 判断元素是否含有某个类名

```
// 判断元素是否包含 active 这一类名，包含返回true，不包含返回false
element.classList.contains('active')
```

关于 `element.classList` 的更多知识，可以查看 MDN 官方文档 。

6. 模板字符串

模板字面量是用反引号（`）分隔的字面量，允许多行字符串、带嵌入表达式的字符串插值和一种叫带标签的模板的特殊结构。

模板字面量有时被非正式地叫作模板字符串，因为它们最常被用作字符串插值（通过替换占位符来创建字符串）。然而，带标签的模板字面量可能不会产生字符串——它可以与自定义标签函数一起使用，来对模板字面量的不同部分执行任何操作。

常用的字符串插值语法：

```
let name = '小蓝';
let str1 = '我的名字叫小蓝';
let str2 = `我的名字叫${name}`;

str1 === str2;   // true
```

关于模板字符串的更多知识，可以查看 MDN 官方文档 。

- 解题思路

首先我们来分析题目中已经给定的 js 代码。

可以看到在类 `Tab` 的 `constructor` 构造函数中，执行了一些获取元素的操作，并分别把这些元素赋值给类的几个属性。

由于创建类的实例后，类中的属性会变为固定的值，**不会随着元素的变化而再次变化**，因此针对需要不断更新的属性或事件，作者提供了 `init` 方法，在其中获取必要的元素赋值给类的几个属性，并给这些元素绑定了事件。当我们需要更新属性或事件时，直接调用 `init` 方法即可。

在 `clearClass` 方法中，执行了清空所有标签页和内容页类名的操作。

剩余的几个方法就是由我们补充添加的了。

1. 目标 1

目标 1 要求我们点击标签页时标签页及其内容页变为选中状态，实则就是使用 js 给标签页和内容页分别动态添加和删除 `liactive` 和 `conactive` 这两个类名。

现在你可能会想，那这个简单，作者已经提供了获取好的标签页元素 `this.lis` 和内容页元素 `this.sections`，直接在函数中使用 `this.lis` 修改点击元素的类名不就行了？那你就大错特错了！要知道，在 `toggleTab` 函数这个局部作用域中，`this` 的指向是该函数的调用者，而这个函数的调用者是 `this.lis` 中的每一个元素，因此不能通过 `this.lis` 来改变每个标签页的类名。

那我们该怎么做呢？仔细看 `toggleTab` 这个方法的调用位置，即 `this.lis[i].onclick = this.toggleTab.bind(this.lis[i], this)` 这句代码，我们发现它是通过 `bind` 方法调用，返回了一个新的函数作为 `this.lis` 中每个元素点击事件的回调函数，而 `bind` 方法传了 2 个参数，第一个参数是 `this.lis` 中的元素，改变了新函数的 `this` 指向；第二个参数 `this` 是 `Tab` 这个类的实例，即 `tab`，作为了返回的新函数的参数。此时你应该恍然大悟了吧？我们要使用的标签页是类 `Tab` 的 `lis` 属性，因此就可以通过 `toggleTab` 这个函数的参数来使用这个属性。这里我们有两种可选思路来获取函数参数：

- 自定义参数：假设 `toggleTab` 的参数名为 `that` ，则 `toggle` 中使用类`Tab` 的 `lis` 属性的语句应为：`that.lis`。
- 使用 `arguements`：自定义一个常量 `that`，则通过 `that = arguements[0]` 即可拿到函数的参数，在 `toggle` 中使用类`Tab` 的 `lis` 属性的语句应为：`that.lis`。

那么问题来了，我们可以通过 `toggleTab` 函数中的 `this` 拿到用户所点击的标签页对象，进而改变其类名，但我们还要使点击的标签页所对应的内容页也高亮，怎么拿到点击的标签页所对应的内容页对象呢？由于标签页与其内容页是一一对应的关系，因此点击的标签页在其父元素中的索引也是其内容页在其内容页父元素中的索引，很明显 `this.index` 即是点击的标签页在其父元素中的索引，因此所点击的标签页所对应的内容页对象就可以通过 `that.sections[this.index]` 来拿到。

切换标签页高亮的基本逻辑是：首先使所有标签页全部取消高亮，然后再高亮点击的标签页，这样就实现了标签页切换高亮状态的效果。但是注意一点，还是由于创建类的实例后，类中的属性会变为固定的值，**不会随着元素的变化而再次变化**，因此我们需要手动调用 `init` 方法来更新元素。

如果使用自定义参数的方法拿到参数，则 `toggleTab` 就应该是：

```
toggleTab(that) {  // 参数that是Tab类的实例
        // TODO: 添加代码，点击标签页，切换到对应标签页的功能
    that.clearClass();   // 首先把所有类名清空
        that.init()  // 手动更新元素
        this.className = 'liactive';  // 高亮点击的标签页
        that.sections[this.index].className = 'conactive';  // 高亮对应的内容也
        // TODO结束
}
```

如果使用 `arguements` 来获取参数，则 `toggleTab` 就应该是：

```
toggleTab() {
        // TODO: 添加代码，点击标签页，切换到对应标签页的功能
        const that = arguements[0] // that是Tab类的实例
    that.clearClass();    // 首先把所有类名清空
        that.init()  // 手动更新元素
        this.className = 'liactive';  // 高亮点击的标签页
        that.sections[this.index].className = 'conactive';  // 高亮对应的内容也
        // TODO结束
}
```

在后续目标的实现中，我们均使用 `arguements` 来得到函数的参数。

2. 目标 2

目标 2 要求在 `editTab` 方法中添加代码实现输入框失焦后改变原文本内容的效果。

我们先来看 `editTab` 方法中已经给定的代码，其已经实现了当双击标签页或内容页文本时，其 innerHTML 会变为一个 `input` 输入框，输入框中默认文本是双击的标签页或内容页的文本，并且处于选中状态。要实现目标 2 的要求，需要用到 `input` 的 `onblur` 事件，即失去焦点事件，在其回调函数中，应当使原标签页或内容页的内容替换为输入框中输入的值。

于是 `editTab` 就应该是：

```
editTab() {
        var str = this.innerHTML;
        window.getSelection ?
window.getSelection().removeAllRanges():document.Selection.empty()
        this.innerHTML = '<input type="text" />';
        var input = this.children[0];
        input.value = str;
        input.select();  //让文本框里的文字处于选定状态
        // TODO: 实现双击修改内容，当文本框失焦时，把修改的值赋给被双击的对象

        input.onblur = function () {
            this.parentNode.innerHTML = this.value;
        }

        // TODO结束
}
```

在输入框的 `onblur` 的回调函数中，`this` 指向 `input` ，则 `this.parentNode` 即是标签页或内容页的文本标签，使其 `innerHTML` 等于 `this.value` ，即输入框中输入的

值，即可完成文本的替换效果，同时输入框也因为被替换而消失。

3. 目标 3

目标 3 要求实现当点击 `.tabadd` 时，页面添加新的标签页和内容页的效果。 `addTab` 方法的调用与目标 1 中 `toggleTab` 方法的调用方式相同，这里不再叙述，我们假设 `addTab` 方法的参数为 `that` 。

要添加一个新的元素，js 中的方法有很多，为了方便，这里我们使用 `insertAdjacentHTML` 方法。作者已经给出了标签页和内容页的 DOM 结构，我们将其复制下来，使用 `insertAdjacentHTML` 插入到父元素的最后方即可。由于插入新标签页后所有标签页与内容页的内容需要按照一定规则进行替换，因此我们还要替换字符串模板中标签页及其内容页的文本值，这里使用模板字符串的插值表达式即可完成替换。

`addTab` 方法应该是：

```
addTab() {
    // TODO: 添加代码，当点击加号，添加新的标签页（对应的内容页也应一并添加）
    const that = arguments[0];
    that.clearClass();
    var index = that.ul.children.length; // 拿到容器元素个数
    index++;    // 在原个数基础上加1
    // (1)创建li元素和section元素
    var li = `<li class="liactive"><span class="content">标签页${index}
</span><span class="iconfont icon-guanbi"><span class="glyphicon  glyphicon-
remove"></span></span></li>`;
    var section = `<section class="conactive">标签页${index}的内容</section>`;
    // (2)把这两个元素追加到对应的父元素里面
    that.ul.insertAdjacentHTML("beforeend", li);
    that.fsection.insertAdjacentHTML("beforeend", section);
    that.init();

    // TODO结束
}
```

4. 目标 4

目标 4 要求实现当点击某个标签页的 `.icon-guanbi` 时，该标签页及其内容页从页面中删除，所有标签页及其内容页的内容仍以目标 3 中的规则开始重新排，标签页的选中状态按照一定规则改变。

`removeTab` 方法的调用与目标1中 `toggleTab` 方法的调用方式相同，这里不再叙述，我们假设 `removeTab` 方法的参数为 `that` 。

js 中可以使用元素的 remove 方法删除这个这个元素，现在关键是找到被点击按钮的标签页及其内容页对象。removeTab 方法的调用者是标签页的 .icon-guanbi ，因此在 removeTab 中 this.parentNode.index 即是被点击删除按钮的标签页及其内容页在其父元素中的索引。因为删除元素后原结构发生改变，因此要重新调用 init 方法来更新标签页。则删除标签页及其内容页的代码为：

```
var index = this.parentNode.index;
that.lis[index].remove();
that.sections[index].remove();
that.init();
```

要使所有标签页及其内容页的内容仍以目标 3 中的规则开始重新排，则重新遍历每个标签页及其内容页文本标签，然后使其文本标签文本值按照规则替换即可。代码如下：

```
for (let i = 0; i < that.sections.length; i++) {
        that.sections[i].innerText = "标签页" + (i + 1) + "的内容";
        that.spans[i].innerText = "标签页" + (i + 1);
}
```

由于对于删除标签页后高亮状态的转移也有规定，因此还要判断一下被删除的标签页是否是选中状态。可以通过遍历删除后剩下的标签页，如果剩下的标签页中存在类名为 liactive 的清空，证明被删除的标签页不是选中状态，则直接退出 removeTab 方法，代码如下：

```
for (let i = 0; i < that.lis.length; i++){
  if (that.lis[i].className == 'liactive') {
      return   // 退出removeTab方法
  }
}
```

如果被删除的标签页是选中状态，则需要判断是否是最后一个标签页。如果是最后一个标签页，则高亮状态应该转移到其上一个标签页；如果不是最后一个标签页，则其高亮状态转移到其下一个标签页。高亮状态转移到某个标签页上相当于用鼠标点击了一下这个标签页，则可以通过元素的 click 方法模拟点击来实现高亮状态转移的效果。判断是否是最后一个标签，可以拿被删除的标签页在其父元素中的索引与删除后父容器的长度做对比，如果相等证明删除的是最后一个，否则不是。该部分代码如下：

```
if (index == that.lis.length) { // 是最后一个，则让他前方那个处于选中状态
    that.lis[that.lis.length-1] && that.lis[that.lis.length-1].click()
```

```
} else { //不是最后一个，则让他后方那个处于选中状态
    that.lis[index] && that.lis[index].click()
}
```

因此 `removeTab` 的代码应为：

```
removeTab() {
    // TODO: 当点击标签页右上角.icon-guanbi时，对应标签页及其内容页被删除。
    const that = arguments[0];
    const e = arguments[1]
    e.stopPropagation(); //防止冒泡
 // this是被点击的那个删除符号
 var index = this.parentNode.index;
 // 根据索引号删除对应的li和section
    that.lis[index].remove();
    that.sections[index].remove();
    that.init();
    for (let i = 0; i < that.sections.length; i++){
        that.sections[i].innerText = '标签页' + (i + 1) + "的内容"
        that.spans[i].innerText = "标签页"+(i+1)
    }

    // 当我们删除的不是选中状态的li的时候，原来的选中状态li保持不变
    for (let i = 0; i < that.lis.length; i++){
        if (that.lis[i].className == 'liactive') {
            return
        }
    }
    // 如果删除的是选中状态
    // 如果删除的是最后一个li，让他前方的那个处于选中状态
    if (index == that.lis.length) {
        that.lis[that.lis.length-1] && that.lis[that.lis.length-1].click()
    } else { //不是最后一个，则让他后方那个处于选中状态
        that.lis[index] && that.lis[index].click()
    }

    // TODO结束
}
```

注意，因为 `removeTab` 方法的调用者是每个标签页中的 `.icon-guanbi` ,它是标签页的子元素，因此会出现事件冒泡的情况，如果我们不防止冒泡，则标签页对象也会执行 click 方法，会调用 `toggleTab` 方法，当执行 `toggleTab` 中 `that.sections[this.index].className = 'conactive'` 这句代码时，由于 `this.index` 指的是被点击的标签页对象在父元素中的索引，而此时该标签页已经被删除，所以会出现报错的情况，因此还要使用 `stopPropagation` 来阻止事件冒泡。

- 解题代码

```
"use strict";
class Tab {
  // 构造方法
  constructor(id) {
    // 获取元素
    this.main = document.querySelector(id);
    this.add = this.main.querySelector(".tabadd");
    this.ul = this.main.querySelector(".fisrstnav ul");
    this.fsection = this.main.querySelector(".tabscon");
    this.init();
  }
  // 初始化
  init() {
    // 更新所有的li和section
    this.lis = this.main.querySelectorAll("li");
    this.remove = this.main.querySelectorAll(".icon-guanbi");
    this.sections = this.main.querySelectorAll("section");
    this.spans = this.main.querySelectorAll(".content");
    // init初始化操作让相关元素绑定事件
    this.add.onclick = this.addTab.bind(this.add, this);
    for (var i = 0; i < this.lis.length; i++) {
      this.lis[i].index = i;
      this.lis[i].onclick = this.toggleTab.bind(this.lis[i], this);
      this.remove[i].onclick = this.removeTab.bind(this.remove[i], this);
      this.spans[i].ondblclick = this.editTab;
      this.sections[i].ondblclick = this.editTab;
    }
  }
  // 1.切换功能
  toggleTab() {
    // TODO: 添加代码，点击标签页，切换到对应标签页的功能
 const that = arguments[0];
    that.clearClass();
    that.init();
    this.className = "liactive";
    that.sections[this.index].className = "conactive";

    // TODO结束
  }
  // 2.清空所有内容页类名
  clearClass() {
    for (var i = 0; i < this.lis.length; i++) {
      this.lis[i].className = "";
      this.sections[i].className = "";
    }
  }
  // 3.添加标签页
  addTab() {
    // TODO: 添加代码，当点击加号，添加新的标签页（对应的内容页也应一并添加）
 const that = arguments[0];
    that.clearClass();
    var index = that.ul.children.length;
    index++;
    // (1)创建li元素和section元素
    var li =
      '<li class="liactive"><span class="content">标签页' +
```

```
      index +
      '</span><span class="iconfont icon-guanbi"><span class="glyphicon
glyphicon-remove"></span></span></li>';
    var section =
      '<section class="conactive">标签页' + index + "的内容</section>";
    // (2)把这两个元素追加到对应的父元素里面
    that.ul.insertAdjacentHTML("beforeend", li);
    that.fsection.insertAdjacentHTML("beforeend", section);
    that.init();

    // TODO结束
  }
  // 4.删除功能
  removeTab() {
    // TODO: 当点击标签页右上角.icon-guanbi时，对应标签页及其内容页被删除。
 const that = arguments[0];
 const e = arguments[1];
    e.stopPropagation(); //防止冒泡
    // this是被点击的那个元素
    var index = this.parentNode.index;
    // 根据索引号删除对应的li和section
    that.lis[index].remove();
    that.sections[index].remove();
    that.init();
    for (let i = 0; i < that.sections.length; i++) {
      that.sections[i].innerText = "标签页" + (i + 1) + "的内容";
      that.spans[i].innerText = "标签页" + (i + 1);
    }
    // 当我们删除的不是选中状态的li的时候，原来的选中状态li保持不变
    for (let i = 0; i < that.lis.length; i++) {
      console.log(that.lis[i].index);
      console.log(that.lis[i].className);
      if (that.lis[i].className == "liactive") {
        return;
      }
    }

    // 如果删除的是选中状态
    // 如果删除的是最后一个li，让他前方的那个处于选中状态
    if (index == that.lis.length) {
      that.lis[that.lis.length - 1] && that.lis[that.lis.length -
1].click();
    } else {
      //不是最后一个，则让他后方那个处于选中状态
      that.lis[index] && that.lis[index].click();
    }
    // TODO结束
  }
  // 5.修改功能
  editTab() {
    var str = this.innerHTML;
    window.getSelection
      ? window.getSelection().removeAllRanges()
      : document.Selection.empty();
    this.innerHTML = '<input type="text" />';
    var input = this.children[0];
    input.value = str;
```

```
    input.select(); //让文本框里的文字处于选定状态
    // TODO：实现双击修改内容，当文本框失焦时，把修改的值赋给被双击的对象

    // 当我们离开文本框就把文本框里面的值传给span
    input.onblur = function () {
      this.parentNode.innerHTML = this.value;
    };
    // TODO结束
  }
}
var tab = new Tab("#tab");
```

11. 趣味加密解密

```
/*
    常量 defaultCodes 是默认密码表，当输入的密码表长度小于 2 时采用
    请勿进行修改 ，否则可能造成考生提交的函数与检测用例所采用的默认密码表不一致，导致检测无法通过
*/
const defaultCodes="这是十六位默认密码表请勿进行改动";


/*
    函数 string2Unit8Array 接受一个参数 str ，能够返回 str 以 UTF8 格式解码后的 Unit8Array 数组。
    参数 str 类型为 String 字符串。
*/
function string2Unit8Array(str){
    return new TextEncoder().encode(str);
}


/*
    函数 uint8Array2String 接受一个参数 arr ，能够返回 arr 以 UTF8 格式编码之后的 String 字符串。
    参数 arr 类型为 Array 数组。
*/
function uint8Array2String(arr){
    return new TextDecoder().decode(new Uint8Array(arr));
}


/*
    加密函数 encryption 接受两个参数 plainText 和 codes ，能够实现对明文的加密，并返回加密后的密文字符串。
    函数参数 plainText 是用户输入的明文，类型为 String 字符串。
    函数参数 codes 是用户输入的密码表，类型也为 String 字符串，并且可能存在重复字符，需要进行去重
    若去重后 codes 长度小于 2 则应采用默认密码表 defaultCodes 。
*/
function encryption(plainText,codes){
```

```
    //TODO
    //对密码表去重
    //利用split将密码表字符串转为数组，使用 set 去重后以 es6 语法重新转为数组并使用
join 转回字符串，达到字符串去重的目的
    codes = [...new Set((codes||defaultCodes).split(""))].join("");
    //判断密码表是否有效，如果密码长度小于 2 则使用默认密码表，因为零进制和一进制不存在
    if(codes.length<2){
        codes=defaultCodes;
    }
    //以去重后的密码表长度 base 作为进制数
    let base=codes.length;
    //编码单位长度，即每多少个编码代表一个字符
    let unitLength=0;
    while(base**unitLength<256){
        unitLength++;
    };
    //将明文解码为 Unit8Array 数组
    let uint8Array=string2Unit8Array(plainText);
    //开始拼装密文
    let cipherText="";
    let temporaryNum=0;
    let temporaryArray=[];
    let temporaryString="";
    //遍历明文编码数组
    for(let i=0;i<uint8Array.length;i++){
        temporaryNum=uint8Array[i];
        temporaryArray=[];
        temporaryString="";
        //将当前遍历到的编码转为 base 进制，并存入 temporaryArray 中
        //注意，此步骤后 temporaryArray 中的编码为倒序
        while(temporaryNum>0){
            temporaryArray.push(Math.floor(temporaryNum%base));
            temporaryNum=Math.floor(temporaryNum/base);
        }
        //补齐到编码单位长度，否则解密时无法确定多少个密码为一组编码
        while(temporaryArray.length<unitLength){
            temporaryArray.push(0);
        }
        //根据进制转换后的编码与密码表字符串的映射关系，将编码翻译为密码
        for(let j=0;j<temporaryArray.length;j++){
            temporaryString=temporaryString+codes[temporaryArray[j]];
        }
        //将密码拼装入密文
        cipherText+=temporaryString;
    }
    //返回密文
    return cipherText;
}


/*
    加密函数 decryption 接受两个参数 cipherText 和 codes ，能够实现对密文的解密，并
返回解密之后的明文字符串。
    函数参数 cipherText 是用户输入的密文，类型为 String 字符串。
    函数参数 codes 是用户输入的密码表，类型也为 String 字符串，并且可能存在重复字符，需
要进行去重
    若去重后 codes 长度小于 2 则应采用默认密码表 defaultCodes 。
```

```
*/
function decryption(cipherText,codes){
    //TODO
    //对密码表去重
    //利用 split 将密码表字符串转为数组，使用 set 去重后以 es6 语法重新转为数组并使用
join 转回字符串，达到字符串去重的目的
    codes = [...new Set((codes||defaultCodes).split(""))].join("");
    //判断密码表是否有效，如果密码长度小于 2 则使用默认密码表，因为零进制和一进制不存在
    if(codes.length<2){
        codes=defaultCodes;
    }
    //以去重后的密码表长度 base 作为进制数
    let base=codes.length;
    //编码单位长度，即每多少个编码代表一个字符
    let unitLength=0;
    while(base**unitLength<256){
        unitLength++;
    }
    //判断密文长度是否是编码单位长度的整数倍，如若不是则密码表不符合或密文损坏，返回
ERROR 字符串
    if(cipherText.length%unitLength){
        return "ERROR";
    }
    //开始拼装明文
    let temporaryString="";
    let temporaryNum=0;
    let temporaryArray=[];
    //遍历密文字符串，步长为编码单位长度，即以密码组为单位取出并进行翻译
    for(let i=0;i<cipherText.length;i+=unitLength){
        //获取密码组
        temporaryString=cipherText.slice(i,i+unitLength);
        temporaryNum=0;
        //遍历密码组，根据密码在密码表中的次序翻译为字符编码
        for(let j=0;j<temporaryString.length;j++){
            temporaryNum+=codes.indexOf(temporaryString[j])*(base**j);
        };
        //将字符编码存入数组
        temporaryArray.push(temporaryNum);
    }
    //将 temporaryArray 中的数据编码为明文字符串
    let plainText=uint8Array2String(temporaryArray);
    //返回明文
    return plainText;
}

/*
    调用 encryption("你好，世界","0123456789ABCDEF") 方法，应当返回密文
"4EDB0A5E5ADBFECBC84E8B697E59C8" 。
    调用 decryption("4EDB0A5E5ADBFECBC84E8B697E59C8","0123456789ABCDEF") 方
法，应当返回明文 "你好，世界" 。
*/

//以下代码请勿删除，否则可能导致检测不通过
export {defaultCodes,encryption,decryption}
```