- 师资培训课件（DOM 操作题目）
  - 1. 十三届省赛健身大调查
  - 2. 十三届省赛课程列表
  - (3. 十三届国赛新增地址)[]
  - 4. 十四届省赛图片水印生成
  - 5. 十四届省赛视频弹幕
  - 6. 十四届省赛组课神器
  - 7. 十四届国赛恶龙与公主
  - 8. 文本查重小助手
    - 高频考点知识点讲解：
    - 总结： DOM 操作 20 - 50 分。题目数量：1 - 4。高频考点：class 操作，获取 dom ， 设置 DOM 内容，显示和隐藏，新增 DOM ,删除 DOM 节点。
    - 备考建议： DOM 操作题目占分均衡，大部分 DOM 操作题在难度在中等，所有一定要尽量拿分。
  - axios、ajax、fetch 考点
    - 请求拦截器（Request Interceptors）
      - 设置请求拦截器
    - 响应拦截器（Response Interceptors）
      - 设置响应拦截器
    - 示例：GET 和 POST 请求
    - 备考建议: ajax 这里的考点比较简单，占分比例 5-10，一定要拿到。重点就是熟悉 axios 的 API,以及 fetch 的使用。
  - 想拿更高的名次，二维数组出现频率很高，80% 的可能会出现二维数组相关的题目。对于分数较高中较难的题目一般的题型有 二维数组，复杂数据操作，需要书写代码比较多的 DOM 操作。做题的正确率很重要。

师资培训课件（DOM 操作题目）

1. 十三届省赛健身大调查

```
let placeValuesMap = new Map([
  ["1", "公园"],
  ["2", "健身房"],
  ["3", "户外"],
]);
const males = ["男", "女"];
```

```
function formSubmit() {
  var formData = new FormData(myForm);
  let height = formData.get("height");
  let weight = formData.get("weight");
  let male = formData.get("male");
  let places = formData.getAll("place");
  places = places.map((placeValue) => placeValuesMap.get(placeValue));
  placeStr = places.join(",");
  // 表单项隐藏
  quescontent.style.display = "none";
  // 结果项目显示
  result.style.display = "block";
  resultStr = `身高${height}cm，体重 ${weight}kg ，性别 ${males[male]}，会在
${placeStr}健身锻炼。`;
  result.innerText += `\r\n${resultStr}`;
}
```

2. 十三届省赛课程列表

```
let pageNum = 1; // 默认页码
let maxPage; // 最大页数

// TODO : start

let pageSize = 5; // 每页条数
let pagereuslt; // 接收请求返回的数据

let pageData; // 当前页显示的数据
// 显示最大页数和当前是第几页
function pagefun(maxPage, pageNum) {
  pagination.innerHTML = `共 ${maxPage}  页，当前 ${pageNum} 页`;
}
// 将数据渲染到页面上
function renderHtml(pageData) {
  list.innerHTML = pageData
    .map((item) => {
      return `<div class="list-group">
                                                <a href="#" class="list-
group-item list-group-item-action">
                                                        <div class="d-flex
w-100 justify-content-between">
                                                        <h5 class="mb-
1">${item.name}</h5>

<small>${(item.price / 100).toFixed(2)}元</small>
                                                        </div>
                                                        <p class="mb-
1">${item.description}</p>
                                                </a>
                                </div>`;
    })
```

```
    .join("");
}
// 通过 axios 获取数据
axios.get("./js/carlist.json").then((res) => {
  pagereuslt = res.data;
  pageData = pagereuslt.slice((pageNum - 1) * pageSize, pageNum * pageSize);
  renderHtml(pageData);
  maxPage = Math.ceil(pagereuslt.length / pageSize);
  pagefun(maxPage, pageNum);
});
// 点击获取下一页数据
prev.onclick = function () {
  next.classList.remove("disabled");
  pageNum = pageNum - 1;
  if (pageNum <= 1) {
    prev.classList.add("disabled");
    pageNum = 1;
  }
  pageData = pagereuslt.slice((pageNum - 1) * pageSize, pageNum * pageSize);
  renderHtml(pageData);
  pagefun(maxPage, pageNum);
};
// 点击下一页
next.onclick = function () {
  prev.classList.remove("disabled");
  pageNum = pageNum + 1;
  if (pageNum >= maxPage) {
    pagnum = maxPage;
    next.classList.add("disabled");
  }
  pageData = pagereuslt.slice((pageNum - 1) * pageSize, pageNum * pageSize);
  renderHtml(pageData);
  pagefun(maxPage, pageNum);
};

// TODO:END
```

(3. 十三届国赛新增地址)[]

```
// 初始化省份下拉列表内容
function provinceInit() {
  var province = document.getElementById("param_province");
  province.length = provinces.length;
  for (var i = 0; i < provinces.length; i++) {
    province.options[i].text = provinces[i];
    province.options[i].value = provinces[i];
  }
}

// 选择省份后对应城市下拉列表内容渲染
function provincechange() {
  // TODO: start
```

```
  var province = document.getElementById("param_province");
  var num = province.selectedIndex; // 选中 option 索引
  var city = document.getElementById("param_city");
  var citystemp = citys[num];
  city.length = citystemp.length;
  for (var i = 0; i < citystemp.length; i++) {
    city.options[i].text = citystemp[i];
    city.options[i].value = citystemp[i];
  }
  city.options[0].selected = true;
  // TODO: END
}

/**
 * 为标签绑定单击事件。
 * 事件效果为:
 * 1、鼠标点击该标签后该标签样式显示 class=active;
 * 2、其他已选标签的 active 样式被移除;
 * 3、将选中的标签对应下标（即选择器为 “mark a” 选中的标签对应的下标）更新到
id=param_mark 的隐藏的 input 中。
 */
function addClick() {
  // TODO
  var markList = document.querySelectorAll(".mark a");
  for (let i = 0; i < markList.length; i++) {
    markList[i].onclick = function () {
      this.className = "active";
      for (var j = 0; j < markList.length; j++) {
        if (j != index) {
          markList[j].className = "";
        }
      }
      document.getElementById("param_mark").value = index;
    };
  }
  // TODO: END
}
// 提交信息后，读取并显示在页面中
function saveInfo() {
  // TODO
  var province = document.getElementById("param_province").value;
  var city = document.getElementById("param_city").value;
  var address = document.getElementById("param_address").value;
  var mark = document.getElementById("param_mark").value;
  var name = document.getElementById("param_name").value;
  var phone = document.getElementById("param_phone").value;

  if (!province || !city || !address || !name || !phone) {
    document.querySelector(".warning-dialog").style.display = "block";
    return false;
  }

  document.getElementById("main_title").innerHTML = "地址管理";
  document.querySelector(".address").style.display = "none";
  document.querySelector(".user-info").style.display = "none";

  var markClass = "";
```

```
  var markInfo = "";
  switch (mark) {
    case "0":
      markClass = "home";
      markInfo = "家";
      break;
    case "1":
      markClass = "company";
      markInfo = "公司";
      break;
    case "2":
      markClass = "school";
      markInfo = "学校";
      break;
  }

  var addressInfoStr = `<li>
    <div class="show-area">
      ${markInfo ? `<label class="${markClass}">${markInfo}</label>` : ""}
      <span>${province + city}</span>
    </div>
    <div class="show-address">
      <span>${address}</span>
      <a><img src="./images/edit.png" /></a>
    </div>
    <div class="show-info">
      <span>${name}</span>
      <span>${phone}</span>
    </div>
  </li>`;
  document.querySelector(".address-list").innerHTML =
    addressInfoStr + document.querySelector(".address-list").innerHTML;
  document.querySelector(".address-list").style.display = "block";

  // TODO: END
}

// 切换新增地址和地址管理的显隐
function back() {
  if (document.getElementById("main_title").innerHTML == "地址管理") {
    document.getElementById("main_title").innerHTML = "新增地址";
    document.querySelector(".address-list").style.display = "none";
    document.querySelector(".address").style.display = "block";
    document.querySelector(".user-info").style.display = "block";
  }
}
// 页面加载后的初始化操作
function init() {
  // 初始化省份下拉列表内容
  provinceInit();
  // 为标签绑定单击事件
  addClick();
}

window.onload = function () {
  // 初始化
```

```
  init();
};
```

4. 十四届省赛图片水印生成

- 考察： dom 操作、css 、 css3 常见属性

```
// 通过 for 循环生成指定数量的 span 元素，并将它们添加到容器中
for (let index = 0; index < count; index++) {
  // 使用传入的参数设置 span 元素的文本内容、颜色、旋转角度和透明度，并添加到容器中
  container.innerHTML += `<span style="color:${color};transform:
rotate(${deg}deg);opacity:${opacity}">${text}</span>`;
  // 返回添加了 span 元素的容器
}
```

答案 2：

```
// 使用 repeat() 方法创建一个包含指定数量 span 元素的字符串，并为每个元素设置文本内容、
颜色、旋转角度和
const spans =
  `<span style="color:${color};transform:
rotate(${deg}deg);opacity:${opacity}">${text}</span>`.repeat(
    count
  );
// 将包含所有 span 元素的字符串添加到容器中
container.innerHTML = spans;
// 返回添加了 span 元素的容器
return container;
```

5. 十四届省赛视频弹幕

```
// TODO: 控制弹幕的显示颜色和移动，每隔 bulletConfig.time 时间，弹幕移动的距离
bulletConfig.speed
// 创建 span 元素用于展示弹幕
let spanEle = document.createElement("span");
// 将弹幕内容添加到 span 元素中
spanEle.innerHTML = `${bulletConfig.value}`;
// 显示弹幕
spanEle.style.display = "block";
// 获取视频元素的宽高
let { width: vWidth, height: vHeight } = getEleStyle(videoEle);
// 设置弹幕初始位置在视频最右侧，并在垂直方向随机分布
spanEle.style.left = vWidth + "px";
spanEle.style.top = getRandomNum(vHeight) + "px";
```

```
// 将弹幕元素添加到视频元素中
videoEle.appendChild(spanEle);
// 设置弹幕定时器，每隔一段时间将弹幕左移，当弹幕移出屏幕后清除弹幕元素及定时器
let timer = setInterval(() => {
  spanEle.style.left = parseInt(spanEle.style.left) - bulletConfig.speed +
"px";
  if (parseInt(spanEle.style.left) <= -64) {
    if (spanEle) {
      videoEle.removeChild(spanEle);
    }
    clearInterval(timer);
  }
}, bulletConfig.time);

// TODO:点击发送按钮，输入框中的文字出现在弹幕中
// 从输入框中获取弹幕内容
let val = document.querySelector("#bulletContent").value;
// 使用 renderBullet() 函数渲染弹幕
renderBullet(
  {
    ...bulletConfig, // 使用扩展运算符合并弹幕配置信息
    isCreate: true, // 设置为创建新的弹幕
    value: val, // 覆盖原来的弹幕内容
  },
  videoEle, // 视频元素
  true // 是否创建新的弹幕
);
```

6. 十四届省赛组课神器

```
/**
 * @description 模拟 ajax 请求，拿到树型组件的数据 treeData
 * @param {string} url 请求地址
 * @param {string} method 请求方式，必填，默认为 get
 * @param {string} data 请求体数据，可选参数
 * @return {Array}
 * */
async function ajax({ url, method = "get", data }) {
  let result;
  // TODO：根据请求方式 method 不同，拿到树型组件的数据
  // 当method === "get" 时，localStorage 存在数据从 localStorage 中获取，不存在则
从 /js/data.json 中获取
  // 当method === "post" 时，将数据保存到localStorage 中
  if (method === "get") {
    let res;
    if (localStorage.getItem("data")) {
      result = JSON.parse(localStorage.getItem("data"));
    } else {
      data = await axios({ url, method, data });
      res = data.data;
      result = res.data;
    }
```

```
  }
  if (method === "post") {
    localStorage.setItem("data", JSON.stringify(data));
  }
  // TODO: END
  return result;
}

/**



/**
 * @description 根据 dragElementId, dropElementId 重新生成拖拽完成后的树型组件的数据 treeData
 * @param {number} dragGrade 被拖拽的元素的等级，值为 dragElement data-grade属性对应的值
 * @param {number} dragElementId 被拖拽的元素的id，值为当前数据对应在 treeData 中的id
 * @param {number} dropGrade 放入的目标元素的等级，值为 dropElement data-grade属性对应的值
 * @param {number} dropElementId 放入的目标元素的id，值为当前数据对应在 treeData 中的id
 */
function treeDataRefresh(
  { dragGrade, dragElementId }, // 拖拽的等级和元素 ID
  { dropGrade, dropElementId } // 放置的等级和元素 ID
) {
  if (dragElementId === dropElementId) return; // 如果拖拽元素与放置元素相同，直接返回
  // TODO: 根据 `dragElementId, dropElementId` 重新生成拖拽完成后的树型组件的数据 `treeData`

  // 函数：获取并删除拖拽标签对象
  const getAndDeleteDragLabelObj = (dragElementId, data) => {
    let result;
    if (dragGrade - dropGrade > 1 || dragGrade - dropGrade < 0) return result; // 如果拖拽等级和放置等级差值大于 1 或小于 0，直接返回
    const innerFn = (dragElementId, data) => {
      data.forEach((treeObj, index, array) => {
        if (treeObj.id === Number(dragElementId)) { // 如果当前节点的 ID 与拖拽元素 ID 相同
          array.splice(index, 1); // 从数据中删除该节点
          result = treeObj; // 将删除的节点保存到 result 中
        } else {
          treeObj.children && innerFn(dragElementId, treeObj.children); // 如果有子节点，递归查找
        }
      });
    };
    innerFn(dragElementId, data);
    return result; // 返回删除的节点对象
  };

  // 函数：将拖拽标签对象设置到树形数据中
  const setDragLabelObjToTreeData = (dragLabelObj, dropElementId, data) => {
    for (let i = 0; i < data.length; i++) {
      const treeObj = data[i];
```

```
      if (treeObj.id === Number(dropElementId)) { // 如果当前节点的 ID 与放置元素 ID 相同
        if (dragGrade - dropGrade == 1) { // 如果拖拽等级比放置等级低一级
          treeObj.children ? treeObj.children.unshift(dragLabelObj) : (treeObj["children"] = [dragLabelObj]); // 将拖拽标签对象添加到放置元素的子节点中
        } else if (dragGrade - dropGrade == 0) { // 如果拖拽等级与放置等级相同
          data.splice(i + 1, 0, dragLabelObj); // 将拖拽标签对象插入到当前节点的后面
          break;
        }
      } else {
        treeObj.children && setDragLabelObjToTreeData(dragLabelObj, dropElementId, treeObj.children); // 如果有子节点，递归设置
      }
    }
  };

  let dragLabelObj = getAndDeleteDragLabelObj(dragElementId, treeData); // 获取并删除拖拽标签对象
  if (dropElementId) { // 如果有放置元素
    dragLabelObj && setDragLabelObjToTreeData(dragLabelObj, dropElementId, treeData); // 将拖拽标签对象设置到树形数据中
  } else {
    treeData.unshift(dragLabelObj); // 如果没有放置元素，则将拖拽标签对象添加到树形数据的开头
  }
  // TODO:END
}


/**
 * @description 根据 treeData 的数据生成树型组件的模板字符串，在包含 .tree-node 的元素节点需要加上 data-grade=${index}表示菜单的层级 data-index="${id}" 表示菜单的唯一 id
 * @param {array} data treeData 数据
 * @param {number} grade 菜单的层级
 * @return 树型组件的模板字符串
 *
 * */
function treeMenusRender(data, grade = 0) {
  let treeTemplate = "";
  // TODO: 根据传入的 treeData 的数据生成树型组件的模板字符串
  grade++;
  return data.reduce((prev, cur) => {
    let isContainChild = !!cur.children;
    return (treeTemplate += `
        <div class="tree-node"  data-grade=${grade} data-index="${cur.id}">
            <div class="tree-node-content" style="margin-left:${
              (grade - 1) * 15
            }px">
                <div class="tree-node-content-left">
                    <img src="https://static.shiyanlou.com/opcenter-fe/frontend/dist/img/icon-dragger.2c0734b.svg"  class="point-svg">
                    ${
                      cur.tag
                        ? `<span class="tree-node-tag" >${cur.tag}</span>`
                        : ""
```

```
                    }
                    <span class="tree-node-label">${cur.label}</span>
                    ${
                      isContainChild
                        ? `<img class="config-svg"
src="https://static.shiyanlou.com/opcenter-fe/frontend/dist/img/icon-
setting.4ab65af.svg">`
                        : ""
                    }
                </div>

                ${
                  !isContainChild
                    ? `<div class="tree-node-content-right">
                        <div class="students-count">
                            <span class="number">0人完成</span>
                            <span class="line">|</span>
                            <span class="number">0人提交报告</span>
                        </div>
                        <div class="config">
                            <img class="config-svg"
                                src="https://static.shiyanlou.com/opcenter-
fe/frontend/dist/img/icon-setting.4ab65af.svg"
                                alt="">
                            <button class="doc-link">编辑文档</button>
                        </div>
                    </div>`
                    : ""
                }
            </div>
            ${
              isContainChild
                ? `<div class="tree-node-children">
                    ${isContainChild && treeMenusRender(cur.children,
grade)}
                </div>`
                : ""
            }
        </div>
        `);
  }, "");
}
```

7. 十四届国赛恶龙与公主

```
let game = {
  element: {
    boxs: document.querySelectorAll(".container .box"),
    bloodEle: document.querySelector("#blood span"),
    tipEle: document.querySelector(".tip"),
    gameCtl: document.querySelector("#control .btn"),
```

```
    gameStep: document.querySelector("#control .ctl-input"),
    gamePath: document.querySelector("#game-path"),
  },
  gameData: {
    step: 0,
    dragonPos: [
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 21, 22,
23,
      24,
    ].filter((i) => Math.random() > 0.8), // 恶龙的位置
    angelPos: 20, // 天使的位置
    randomArr: [
      0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20,
      21, 22, 23, 24,
    ],
    pathArr: [], // 表示行走的路径数组
    curPos: 0, // 表示当前的位置
    curBlood: 3, // 表示当前的血量
  },
  events: {
    moveHandlerDebounce: null,
  },
  randomNum(start, end) {
    return Math.floor(Math.random() * (end - start + 1)) + start;
  },
  getStep(step) {
    return step ? step : Number(this.randomNum(1, 3));
  },
  // 游戏的所有格子的 data-index 属性按照摆放的位置组成二维数组，起始和最中间的格子单独
赋值
  getDyadicArray() {
    const dyadicArray = [];
    let tempArr = [];
    Array.from(this.element.boxs).forEach((box, index) => {
      let i = index % 5;
      tempArr.push(Number(box.dataset.index));
      if (i == 4) {
        dyadicArray.push(tempArr);
        tempArr = [];
      }
    });
    dyadicArray[0][0] = "start";
    dyadicArray[2][2] = "end";
    return dyadicArray;
  },

  debounce(handle, isImmediate = false, wait = 500) {
    let timer = null;
    let isDone;
    return function (...args) {
      let self = this;
      isDone = isImmediate && !timer;
      timer && clearTimeout(timer);
      timer = setTimeout(() => {·
        timer = null;
        !isImmediate && handle.apply(self, args);
```

```
      }, wait);
      isDone && handle.apply(self, args);
    };
  },
  tipRender(tip) {
    const tipEle = this.element.tipEle;
    tipEle.classList.remove("hide");
    tipEle.classList.add("show");
    if (tip === "success") {
      tipEle.style.backgroundColor = "green";
      tipEle.innerText = "营救成功";
    }
    if (tip === "warning") {
      tipEle.style.backgroundColor = "red";
      tipEle.innerText = "重伤不治";
    }
  },
  bloodCalculator(boxEle) {
    curBlood = this.gameData.curBlood;
    if (Array.from(boxEle.classList).includes("dragon")) {
      curBlood -= 2;
    }
    if (Array.from(boxEle.classList).includes("angel")) {
      curBlood += 3;
    }
    if (curBlood <= 0) {
      this.tipRender("warning");
      curBlood = 0;
      this.element.gameCtl.removeEventListener(
        "click",
        this.events.moveHandlerDebounce
      );
    }
    this.element.bloodEle.innerText = curBlood;
    this.gameData.curBlood = curBlood;
  },
  /**
   * @param {array}: getDyadicArray 得到的二维数据，存储地图上每个box的 data-
index 值
   * @return {array}: 一维数组，保存经过的路径
   */
  mazePath(arr) {
    // TODO:得到起点到终点经过的每个 box 元素的 data-index 的值并依次保存在数组中
    let bottom = arr.length,
      right = arr[0].length;
    let total = bottom * right;
    const result = [];
    let left = 0,
      top = 0,
      count = 0;
    while (left < right && top < bottom) {
      //l-> r
      for (let j = left; j < right; j++) {
        result.push(arr[top][j]);
        j == right - 1 && top++;
      }
      // t-> b
```

```
      for (let i = top; i < bottom; i++) {
        result.push(arr[i][right - 1]);
        i == bottom - 1 && right--;
      }
      // r -> l
      for (let j = right - 1; j >= left; j--) {
        result.push(arr[bottom - 1][j]);
        j == left && bottom--;
      }
      // b -> t
      for (let i = bottom - 1; i >= top; i--) {
        result.push(arr[i][left]);
        i == top && left++;
      }
    }
    console.log(JSON.stringify(result.slice(0, total)));
    return result.slice(0, total);
  },
  moveHandler() {
    let step = (this.element.gameStep.value =
this.getStep(this.gameData.step));
    // TODO: 根据点击营救后获得的点数，正确到达指定位置调用bloodCalculator函数计算当前血量，到达公主处调用 tipRender 函数，每步的时间间隔在 200ms内（大于此时间会导致判题失败）。
    curPos = this.gameData.curPos;
    let start = curPos;
    curPos += step;
    curPos = curPos >= 24 ? 24 : curPos;
    this.gameData.curPos = curPos;
    for (let i = start; i <= curPos; i++) {
      let timer = setTimeout(() => {
        this.element.boxs.forEach((box) => {
          if (this.gameData.pathArr[i] == box.dataset.index) {
            box.classList.add("active");
            if (i == curPos) {
              this.bloodCalculator(box);
            }
          } else {
            box.classList.remove("active");
          }
        });
        if (i === 24) {
          this.tipRender("success");
        }
        timer !== null && (timer = null);
      }, (i - start) * 200);
    }
  },
  initPlay() {
    Array.from(this.element.boxs).forEach((box, index, boxs) => {
      if (index === 0) {
        box.innerText = "start";
        box.setAttribute("data-index", "start");
      } else if (index === Math.floor(boxs.length / 2)) {
        box.classList.add("princess");
        box.setAttribute("data-index", "end");
      } else {
```

```
        box.setAttribute("data-index", this.gameData.randomArr[index]);
        if (this.gameData.dragonPos.includes(index)) {
          box.classList.add("dragon");
        } else if (index == this.gameData.angelPos) {
          box.classList.add("angel");
        } else {
          box.style.backgroundColor = "#f5f5f5";
          box.innerText = `无危险`;
        }
      }
    });
    this.element.bloodEle.innerText = this.gameData.curBlood;
    this.gameData.pathArr = this.mazePath(this.getDyadicArray());
    this.element.gamePath.innerText = JSON.stringify(this.gameData.pathArr);
    this.events.moveHandlerDebounce = this.debounce(
      this.moveHandler.bind(this)
    );
    this.element.gameCtl.addEventListener(
      "click",
      this.events.moveHandlerDebounce
    );
  },
};
```

8. 文本查重小助手

高频考点知识点讲解：

1. form 表单数据 (必须要掌握的内容)

- `let formData = new FormData(myForm);`：创建了一个 FormData 对象，它用于将表单数据编码成键/值对集合以便于使用。`myForm` 是一个表单元素的引用，用于指定要从哪个表单中获取数据。

- `formData.get('name')` // 获取表单中 name 属性的值

2. `class` 操作（必考点，一定要拿的分数）

```
<!DOCTYPE html>
<html>
  <head>
    <title>Class操作示例</title>
    <style>
      .highlight {
        background-color: yellow;
      }
    </style>
```

```
  </head>
  <body>
    <div id="myDiv" class="highlight">这是一个div元素</div>

    <script>
      // 切换类名
      document.getElementById("myDiv").classList.toggle("highlight");

      // 替换类名
      document
        .getElementById("myDiv")
        .classList.replace("highlight", "newHighlight");
    </script>
  </body>
</html>
```

3. 获取 DOM 元素、设置 DOM 内容、显示和隐藏

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM操作示例</title>
  </head>
  <body>
    <button onclick="changeContent()">改变内容</button>
    <div id="myElement">这是一个可改变内容的元素</div>

    <script>
      function changeContent() {
        document.getElementById("myElement").innerHTML = "内容已改变! ";
      }

      // 显示元素
      document.getElementById("myElement").style.display = "block";

      // 隐藏元素
      document.getElementById("myElement").style.display = "none";
    </script>
  </body>
</html>
```

4. 新增 DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>新增DOM示例</title>
  </head>
  <body>
    <div id="container">
      <h2>原始内容</h2>
```

```
    <p>这是原始的段落。</p>
  </div>

  <button onclick="addParagraph()">添加段落</button>

  <script>
    function addParagraph() {
      var newParagraph = document.createElement("p");
      var text = document.createTextNode("这是新添加的段落。");
      newParagraph.appendChild(text);

      // 将新段落插入到容器中的第一个子元素之前
      document
        .getElementById("container")
        .insertBefore(
          newParagraph,
          document.getElementById("container").firstChild
        );
    }
  </script>
 </body>
</html>
```

5. 删除 DOM 节点

```
<!DOCTYPE html>
<html>
  <head>
    <title>删除DOM示例</title>
  </head>
  <body>
    <div id="container">
      <h2>原始内容</h2>
      <p class="paragraph">这是要删除的段落。</p>
      <p class="paragraph">这是另一个要删除的段落。</p>
    </div>

    <button onclick="deleteParagraph()">删除第一个段落</button>

    <script>
      function deleteParagraph() {
        var elements = document.getElementsByClassName("paragraph");
        if (elements.length > 0) {
          elements[0].parentNode.removeChild(elements[0]);
        }
      }
    </script>
  </body>
</html>
```

6. 模板字符串

- 第一个写个 “`” 表示模板字符串
- 模板字符串中的变量 用 `${}` 变量

这些示例演示了更多的 `class` 操作、获取 DOM 元素、设置 DOM 内容、显示和隐藏、新增 DOM、删除 DOM 节点的方法。

总结： DOM 操作 20 - 50 分。题目数量：1 - 4。高频考点：`class` 操作，获取 dom ，设置 DOM 内容，显示和隐藏，新增 DOM ,删除 DOM 节点。

备考建议： DOM 操作题目占分均衡，大部分 DOM 操作题在难度在中等，所有一定要尽量拿分。

axios、ajax、fetch 考点

难度：⭐ 到 ⭐⭐

总结：`axios` 考点不会单独出现，而是在某道题目中，axios 的考点相对简单，属于必须拿下的分数。

当你在使用 Axios 发送请求时，你可以设置请求拦截器（Request Interceptors）和响应拦截器（Response Interceptors），这些拦截器可以让你在请求发送前和响应返回前进行一些额外的操作。下面我会详细讲解这两种拦截器，并提供一些相关的示例。

请求拦截器（Request Interceptors）

请求拦截器允许你在发送请求之前对其进行修改或添加一些自定义逻辑。常见的用例包括添加认证信息、在每个请求上添加特定的头部、转换请求数据等。

当涉及 Axios 的详细讲解时，我们将覆盖以下几个方面：

1. 发送 GET 请求和 POST 请求的例子。
2. 请求拦截器和响应拦截器的使用。

3. 设置请求头和响应头。

首先，我们将安装 Axios，并使用一个示例 API 来演示这些功能。

然后，我们可以编写一些 Axios 的代码：

```
const axios = require("axios");

// 设置基础 URL
axios.defaults.baseURL = "https://jsonplaceholder.typicode.com";

// 请求拦截器
axios.interceptors.request.use(
  (config) => {
    console.log("Request Interceptor");
    // 在发送请求之前做些什么
    return config;
  },
  (error) => {
    // 对请求错误做些什么
    return Promise.reject(error);
  }
);

// 响应拦截器
axios.interceptors.response.use(
  (response) => {
    console.log("Response Interceptor");
    // 对响应数据做些什么
    return response;
  },
  (error) => {
    // 对响应错误做些什么
    return Promise.reject(error);
  }
);

// 发送 GET 请求
axios
  .get("/posts")
  .then((response) => {
    console.log("GET Request Response:", response.data);
  })
  .catch((error) => {
    console.error("GET Request Error:", error);
  });

// 发送 POST 请求
axios
  .post("/posts", {
    title: "foo",
    body: "bar",
    userId: 1,
  })
  .then((response) => {
```

```
    console.log("POST Request Response:", response.data);
  })
  .catch((error) => {
    console.error("POST Request Error:", error);
  });
```

在这个例子中：

- 我们首先导入了 Axios 模块。
- 设置了 Axios 的默认基础 URL。
- 我们定义了请求拦截器和响应拦截器，以便在请求发送前和响应返回前进行一些操作。
- 发送了一个 GET 请求和一个 POST 请求。
- 对于 GET 请求，我们打印了返回的数据。
- 对于 POST 请求，我们打印了返回的数据。
- 在控制台中，你会看到请求拦截器和响应拦截器的输出。

在上述例子中，我们还没有设置请求头或响应头。现在，我们将添加这部分内容。

```
// 发送 GET 请求，并设置请求头
axios
  .get("/posts", {
    headers: {
      Authorization: "Bearer token",
      "Content-Type": "application/json",
    },
  })
  .then((response) => {
    console.log("GET Request Response:", response.data);
  })
  .catch((error) => {
    console.error("GET Request Error:", error);
  });

// 发送 POST 请求，并设置请求头
axios
  .post(
    "/posts",
    {
      title: "foo",
      body: "bar",
      userId: 1,
    },
    {
      headers: {
        Authorization: "Bearer token",
        "Content-Type": "application/json",
      },
    }
  )
```

```
  .then((response) => {
    console.log("POST Request Response:", response.data);
  })
  .catch((error) => {
    console.error("POST Request Error:", error);
  });
```

在这个例子中，我们为每个请求设置了请求头，其中包括授权令牌（Authorization）和 Content-Type。你可以根据自己的需要添加或修改请求头。

综上所述，我们通过这个示例详细讲解了 Axios 的使用方法，包括发送 GET 请求和 POST 请求、设置请求拦截器和响应拦截器、以及设置请求头和响应头。这些功能使得 Axios 成为一个非常强大且灵活的 HTTP 客户端库。

设置请求拦截器

你可以通过 `axios.interceptors.request.use()` 方法来设置请求拦截器。这个方法接受两个函数作为参数：一个用于处理请求成功时的回调函数，另一个用于处理请求失败时的回调函数。

```
// 设置请求拦截器
axios.interceptors.request.use(
  function (config) {
    // 在请求发送之前做一些操作
    console.log("Request Interceptor - Request Sent:", config);

    // 示例：添加请求头
    config.headers.Authorization = "Bearer your_token_here";

    return config;
  },
  function (error) {
    // 对请求错误做些什么
    console.error("Request Interceptor - Request Error:", error);
    return Promise.reject(error);
  }
);
```

响应拦截器（Response Interceptors）

响应拦截器允许你在接收到响应数据之前对其进行修改或添加一些自定义逻辑。常见的用例包括全局错误处理、对响应数据进行预处理等。

设置响应拦截器

你可以通过 `axios.interceptors.response.use()` 方法来设置响应拦截器。这个方法接受两个函数作为参数：一个用于处理成功响应的回调函数，另一个用于处理失败响应的回调函数。

```
// 设置响应拦截器
axios.interceptors.response.use(
  function (response) {
    // 对响应数据做些事
    console.log("Response Interceptor - Response Received:", response);

    // 示例：对响应数据进行处理
    response.data = response.data.results; // 假设响应数据在 results 字段中

    return response;
  },
  function (error) {
    // 对响应错误做些什么
    console.error("Response Interceptor - Response Error:", error);
    return Promise.reject(error);
  }
);
```

示例：GET 和 POST 请求

下面是一个包含 GET 和 POST 请求的示例，同时演示了请求拦截器和响应拦截器的使用。

```
// 设置请求拦截器
axios.interceptors.request.use(
  (config) => {
    // 在请求发送之前做一些操作
    console.log("Request Interceptor - Request Sent:", config);

    // 示例：添加请求头
    config.headers.Authorization = "Bearer your_token_here";

    return config;
  },
  (error) => {
    // 对请求错误做些什么
    console.error("Request Interceptor - Request Error:", error);
    return Promise.reject(error);
  }
);

// 设置响应拦截器
axios.interceptors.response.use(
  (response) => {
```

```
    // 对响应数据做些事
    console.log("Response Interceptor - Response Received:", response);

    // 示例：对响应数据进行处理
    response.data = response.data.results; // 假设响应数据在 results 字段中

    return response;
  },
  (error) => {
    // 对响应错误做些什么
    console.error("Response Interceptor - Response Error:", error);
    return Promise.reject(error);
  }
);

// 发送 GET 请求
axios
  .get("https://api.example.com/data")
  .then((response) => {
    console.log("GET Request Response:", response.data);
  })
  .catch((error) => {
    console.error("GET Request Error:", error);
  });

// 发送 POST 请求
axios
  .post("https://api.example.com/post", { key: "value" })
  .then((response) => {
    console.log("POST Request Response:", response.data);
  })
  .catch((error) => {
    console.error("POST Request Error:", error);
  });
```

这个示例展示了如何设置请求拦截器和响应拦截器，并进行了 GET 和 POST 请求。在实际应用中，你可以根据自己的需求定制拦截器的行为，以满足特定的业务逻辑和需求。

2. fetch 请求（和原生 ajax 请求掌握其中一种写法即可，学习的时候建议都学习）

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Simple Fetch Example</title>
  </head>
  <body>
    <h2>Simple Fetch Example</h2>
    <div id="response"></div>

    <script>
      // 发起Fetch请求
```

```
      fetch("https://jsonplaceholder.typicode.com/posts/1")
        .then((response) => {
          // 检查响应状态
          if (!response.ok) {
            throw new Error("Network response was not ok");
          }
          // 将响应解析为JSON格式
          return response.json();
        })
        .then((data) => {
          // 将响应数据显示在页面上
          document.getElementById("response").textContent =
            JSON.stringify(data);
        })
        .catch((error) => {
          // 处理请求错误
          document.getElementById("response").textContent =
            "Error: " + error.message;
        });
    </script>
  </body>
</html>
```

1. 原生 ajax 请求

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>AJAX Example</title>
  </head>
  <body>
    <button id="loadDataBtn">Load Data</button>
    <div id="output"></div>

    <script>
      document
        .getElementById("loadDataBtn")
        .addEventListener("click", function () {
          var xhr = new XMLHttpRequest();

          xhr.onload = function () {
            if (xhr.status >= 200 && xhr.status < 300) {
              document.getElementById("output").innerHTML =
xhr.responseText;
            } else {
              document.getElementById("output").innerHTML =
                "Error: " + xhr.status;
            }
          };

          xhr.onerror = function () {
            document.getElementById("output").innerHTML = "Request failed";
```

```
            };

            xhr.open("GET", "example-data.txt"); // Replace "example-data.txt"
with your actual URL
            xhr.send();
          });
      </script>
    </body>
</html>
```

备考建议: ajax 这里的考点比较简单，占分比例 5-10，一定要拿到。重点就是熟悉 axios 的 API,以及 fetch 的使用。

想拿更高的名次，二维数组出现频率很高，80% 的可能会出现二维数组相关的题目。对于分数较高中较难的题目一般的题型有 二维数组，复杂数据操作，需要书写代码比较多的 DOM 操作。做题的正确率很重要。