

.net 架构第三次大作业

目录

第一题：	2
一、要求：	2
二、项目文件简要概述	3
三、JavaScript 详解：	4
四、程序运行截图	6
五、遇到的问题与解决	7
六、代码附录	8
第二题：	13
一、要求：	13
二、项目文件简要概述	14
三、CSS 和 JavaScript 详解	15
四、程序运行截图	19
五、遇到的问题与解决	20
六、代码附录	21
第三题：	25
一、要求：	25
二、项目文件简要概述	26
三、JavaScript 详解	27
四、程序运行截图	28
五、遇到的问题与解决	29
六、代码附录	30

第一题：

一、要求：

实现表格排序功能（表格见课堂示例）

测试

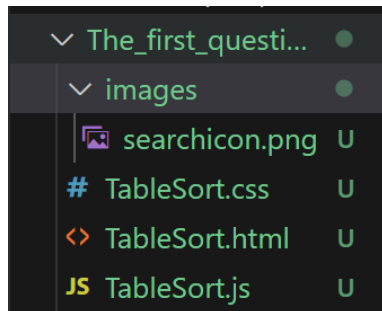
文件 | C:/Users/wustzz/Desktop/前端开发/JS学习/04表格数据搜索/index.html

搜索公司...


公司名称	国家
Alfreds Futterkiste	Germany
Berglunds snabbkop	Sweden
Island Trading	UK
Koniglich Essen	Germany

要求：点击表头（“公司名称”或“国家”）对表格进行排序，第一次点击时，排序为升序（A 到 Z），再次点击，排序为降序（Z 到 A）。

二、项目文件简要概述



HTML 和 CSS 实现基本的一个“公司名称”或“国家”表格

另：加入一个精细小图标, 让界面更高级

```
#myInput {  
  background-image: url("images/searchicon.png");  
  background-position: 10px 12px;  
  background-repeat: no-repeat;  
  font-size: 16px;  
  padding: 12px 20px 12px 40px;  
  border: 1px solid #ddd;  
  margin-bottom: 12px;  
}
```

三、JavaScript 详解：

这段代码实现了一个简单的表格搜索功能，
用户在输入框中输入文本时，将会过滤表格中的行，只显示匹配搜索关键词的行。
定义一个搜索函数

```
function search() {  
    // 获取输入框元素  
    let input = document.getElementById("myInput");  
    // 获取输入框的值并转换为大写以进行不区分大小写的搜索  
    let filter = input.value.toUpperCase();  
    // 获取包含表格的元素  
    let table = document.getElementById("myTable");  
    // 获取表格中的所有行  
    let tr = table.querySelectorAll("tr");  
    // 遍历表格中的每一行  
    for (let i = 0; i < tr.length; i++) {  
        // 获取当前行的第一个单元格  
        let td = tr[i].querySelectorAll("td")[0];  
        // 检查单元格是否存在  
        if (td) {  
            // 检查单元格中的文本是否包含搜索关键词  
            if (td.innerHTML.toUpperCase().indexOf(filter) > -1) {  
                // 如果包含，显示该行  
                tr[i].style.display = "";  
            } else {  
                // 如果不包含，隐藏该行  
                tr[i].style.display = "none";  
            }  
        }  
    }  
}
```

接着获取输入框元素并且添加输入事件监听器，当输入框的内容发生变化时执行搜索函数

```
// 获取输入框元素  
let input = document.getElementById("myInput");  
  
// 添加输入事件监听器，当输入框的内容发生变化时执行搜索函数  
input.addEventListener("input", search);
```

这段代码实现了一个用于表格排序的函数，
它可以根据用户点击表头来对表格的特定列进行升序或降序排序。
定义用于表格排序的函数，参数 n 代表列号

```
function sortTable(n) {  
    var table,  
        rows,  
        switching,  
        i,  
        x,  
        y,  
        shouldSwitch,  
        dir,  
        switchcount = 0;  
    // 获取包含表格的元素  
    table = document.getElementById("myTable");  
    // 开始排序过程  
    switching = true;
```

默认排序方向为升序

```
// 默认排序方向为升序  
dir = "asc";  
while (switching) {  
    switching = false;  
    // 获取表格的所有行  
    rows = table.rows;  
    // 遍历表格的行，从第二行开始（第一行通常是表头）  
    for (i = 1; i < rows.length - 1; i++) {  
        shouldSwitch = false;  
        // 获取当前行和下行的要比较的单元格  
        x = rows[i].getElementsByTagName("TD")[n];  
        y = rows[i + 1].getElementsByTagName("TD")[n];  
        // 根据排序方向和单元格内容比较决定是否交换行  
        if (dir == "asc") {  
            if (x.innerHTML.toLowerCase() > y.innerHTML.toLowerCase()) {  
                shouldSwitch = true;  
                break;  
            }  
        } else if (dir == "desc") {  
            if (x.innerHTML.toLowerCase() < y.innerHTML.toLowerCase()) {  
                shouldSwitch = true;  
                break;  
            }  
        }  
    }  
}
```

如果需要交换行，则进行交换并标记为已交换

```
// 如果需要交换行，则进行交换并标记为已交换  
if (shouldSwitch) {  
    rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);  
    switching = true;  
    switchcount++;  
} else {  
    // 如果没有发生交换并且排序方向为升序，切换为降序重新进行排序  
    if (switchcount == 0 && dir == "asc") {  
        dir = "desc";  
        switching = true;  
    }  
}
```

四、程序运行截图

公司名称顺序排列：

Q 搜索公司...

公司名称	国家
Alfreds Futterkiste	Germany
Berglunds snabbkop	Sweden
Island Trading	UK
Koniglich Essen	Germany

公司名称倒序排列：

Q 搜索公司...

公司名称	国家
Koniglich Essen	Germany
Island Trading	UK
Berglunds snabbkop	Sweden
Alfreds Futterkiste	Germany

国家名称顺序排列：

Q 搜索公司...

公司名称	国家
Koniglich Essen	Germany
Alfreds Futterkiste	Germany
Berglunds snabbkop	Sweden
Island Trading	UK

国家名称倒序排列：

Q 搜索公司...

公司名称	国家
Island Trading	UK
Berglunds snabbkop	Sweden
Koniglich Essen	Germany
Alfreds Futterkiste	Germany

搜索功能：

Q Island

公司名称	国家
Island Trading	UK

五、遇到的问题与解决

已经解决的问题

(1) 识别点击的表头和所需的排序列：

问题：需要确定用户点击的表头是哪一列，以及当前的排序状态（升序或降序）。

解决方案：使用 HTML 的 `data-` 属性来存储列名称和当前排序状态。在 JavaScript 中，使用事件监听器来捕获表头的点击事件，然后解析 `data-` 属性以获得所需信息。

(2) 排序算法的选择：

问题：需要编写有效的排序算法来排序表格行。

解决方案：常用的排序算法包括冒泡排序、快速排序、插入排序等。选择一个合适的排序算法，根据当前的排序状态（升序或降序）对表格数据进行排序。

(2) 表格行的重新排序：

问题：如何将排序后的行重新渲染到表格中？

解决方案：您可以通过删除当前表格中的所有行，然后按照排序后的顺序重新插入它们。这可以通过 JavaScript 操作 DOM 元素来实现。

进一步可以改进的方面

(1) 用户界面反馈：

问题：如何向用户提供反馈，以显示当前的排序状态？

解决方案：通常，可以使用 CSS 类或图标来指示当前排序状态（升序或降序）。例如，通过在表头添加箭头图标或更改文本颜色来表示排序状态。

(2) 性能考虑：

问题：对大型表格进行排序可能会导致性能问题。

解决方案：对于大型表格，考虑使用更高效的排序算法，并在需要时对可见部分进行排序，而不是对整个表格进行排序。

(3) 浏览器兼容性：

问题：不同浏览器可能对 JavaScript 和 CSS 的某些特性支持不同。

解决方案：确保您的代码在不同主流浏览器上进行测试，并根据需要提供浏览器兼容性的解决方案。

六、代码附录

TableSort.html

```
<!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="/TableSort.css" />
    <title>TableSort</title>
  </head>
  <body>
    <input type="text" id="myInput" placeholder="搜索公司..." />
    <table id="myTable">
      <tr class="header">
        <th style="width: 60%" onclick="sortTable(0)">公司名称</th>
        <th style="width: 40%" onclick="sortTable(1)">国家</th>
      </tr>
      <tr>
        <td>Alfreds Futterkiste</td>
        <td>Germany</td>
      </tr>
      <tr>
        <td>Berglunds snabbkop</td>
        <td>Sweden</td>
      </tr>
      <tr>
        <td>Island Trading</td>
        <td>UK</td>
      </tr>
      <tr>
        <td>Koniglich Essen</td>
        <td>Germany</td>
      </tr>
    </table>
    <script src="/TableSort.js"></script>
  </body>
</html>
```


TableSort.css:

```
#myInput {
  background-image: url("images/searchicon.png");
  background-position: 10px 12px;
  background-repeat: no-repeat;
  font-size: 16px;
  padding: 12px 20px 12px 40px;
  border: 1px solid #ddd;
  margin-bottom: 12px;
}
#myTable {
  border-collapse: collapse;
  width: 100%;
  border: 1px solid #ddd;
  font-size: 18px;
}
#myTable th,
#myTable td {
  text-align: left;
  padding: 12px;
}
#myTable tr {
  border-bottom: 1px solid #ddd;
}
#myTable tr.header,
#myTable tr:hover {
  background-color: #f1f1f1;
}
```

TableSort.js:

//这段代码实现了一个简单的表格搜索功能,

//用户在输入框中输入文本时, 将会过滤表格中的行, 只显示匹配搜索关键词的行。

// 定义一个搜索函数

```
function search() {
```

```
  // 获取输入框元素
```

```
  let input = document.getElementById("myInput");
```

```
  // 获取输入框的值并转换为大写以进行不区分大小写的搜索
```

```
  let filter = input.value.toUpperCase();
```

```
  // 获取包含表格的元素
```

```
  let table = document.getElementById("myTable");
```

```
  // 获取表格中的所有行
```

```
  let tr = table.querySelectorAll("tr");
```

```
  // 遍历表格中的每一行
```

```
  for (let i = 0; i < tr.length; i++) {
```

```
    // 获取当前行的第一个单元格
```

```
    let td = tr[i].querySelectorAll("td")[0];
```

```
    // 检查单元格是否存在
```

```
    if (td) {
```

```
      // 检查单元格中的文本是否包含搜索关键词
```

```
      if (td.innerHTML.toUpperCase().indexOf(filter) > -1) {
```

```
        // 如果包含, 显示该行
```

```
        tr[i].style.display = "";
```

```
      } else {
```

```
        // 如果不包含, 隐藏该行
```

```
        tr[i].style.display = "none";
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

```
// 获取输入框元素
```

```
let input = document.getElementById("myInput");
```

```
// 添加输入事件监听器, 当输入框的内容发生变化时执行搜索函数
```

```
input.addEventListener("input", search);
```

//这段代码实现了一个用于表格排序的函数,

//它可以根据用户点击表头来对表格的特定列进行升序或降序排序。

// 定义用于表格排序的函数，参数 n 代表列号

```
function sortTable(n) {  
    var table,  
        rows,  
        switching,  
        i,  
        x,  
        y,  
        shouldSwitch,  
        dir,  
        switchcount = 0;  
  
    // 获取包含表格的元素  
    table = document.getElementById("myTable");  
  
    // 开始排序过程  
    switching = true;  
  
    // 默认排序方向为升序  
    dir = "asc";  
  
    while (switching) {  
        switching = false;  
  
        // 获取表格的所有行  
        rows = table.rows;  
  
        // 遍历表格的行，从第二行开始（第一行通常是表头）  
        for (i = 1; i < rows.length - 1; i++) {  
            shouldSwitch = false;  
  
            // 获取当前行和下一行的要比较的单元格  
            x = rows[i].getElementsByTagName("TD")[n];  
            y = rows[i + 1].getElementsByTagName("TD")[n];  
  
            // 根据排序方向和单元格内容比较决定是否交换行  
            if (dir == "asc") {  
                if (x.innerHTML.toLowerCase() > y.innerHTML.toLowerCase()) {  
                    shouldSwitch = true;  
                    break;  
                }  
            }  
            } else if (dir == "desc") {
```

```

        if (x.innerHTML.toLowerCase() < y.innerHTML.toLowerCase()) {
            shouldSwitch = true;
            break;
        }
    }
}

// 如果需要交换行，则进行交换并标记为已交换
if (shouldSwitch) {
    rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
    switching = true;
    switchcount++;
} else {
    // 如果没有发生交换并且排序方向为升序，切换为降序重新进行排序
    if (switchcount == 0 && dir == "asc") {
        dir = "desc";
        switching = true;
    }
}
}
}

```

第二题：

一、要求：

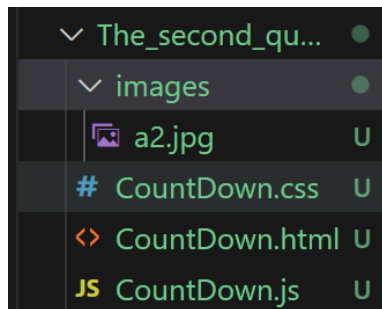
用 JS+CSS 实现高考倒计时



要求：对 JS、CSS 关键代码添加注释。

二、项目文件简要概述

项目基本框架如下：



HTML 实现基本的一个高考倒计时的内容

另：加入一个精细背景图片，让界面更高级



三、CSS 和 JavaScript 详解

CSS 部分：

引入 Google Fonts 中的 Poppins 字体，提供多种字重的选项并重置所有元素的外边距和内边距，使用 Poppins 字体作为默认字体

```
/* 引入 Google Fonts 中的 Poppins 字体，提供多种字重的选项 */
@import url("https://fonts.googleapis.com/css?family=Poppins:100,200,300,400,500,600,700,800,900");

/* 重置所有元素的外边距和内边距，使用 Poppins 字体作为默认字体 */
* {
  margin: 0; /* 清除所有元素的外边距 */
  padding: 0; /* 清除所有元素的内边距 */
  font-family: "Poppins", sans-serif; /* 设置所有文本的字体为 Poppins，如未定义则使用 sans-serif 作为备用字体 */
}
```

设置页面背景样式，包括背景颜色和图像，固定背景并应用混合模式

```
/* 设置页面背景样式，包括背景颜色和图像，固定背景并应用混合模式 */
body {
  background: #000 url(/images/a2.jpg); /* 设置背景颜色为黑色，同时加载背景图片 */
  background-attachment: fixed; /* 固定背景，使其不随页面滚动而滚动 */
  background-blend-mode: hard-light; /* 应用混合模式，以实现图像与背景的混合效果 */
}
```

定义容器样式，通过绝对定位实现定位和居中显示内容

```
/* 定义容器样式，通过绝对定位实现定位和居中显示内容 */
.container {
  position: absolute; /* 使用绝对定位，将容器相对于其最近的具有定位的祖先元素定位 */
  top: 80px; /* 与顶部的距离为80像素 */
  left: 100px; /* 与左侧的距离为100像素 */
  right: 100px; /* 与右侧的距离为100像素 */
  bottom: 80px; /* 与底部的距离为80像素 */
  background: url(/images/a2.jpg); /* 设置容器的背景图片 */
  background-attachment: fixed; /* 固定容器的背景 */
  display: flex; /* 使用弹性布局容器 */
  justify-content: center; /* 在水平方向上居中内容 */
  align-items: center; /* 在垂直方向上居中内容 */
  flex-direction: column; /* 设置内容的排列方向为纵向（垂直） */
  box-shadow: 0 50px 50px #000, 0 0 100px #000; /* 创建内容区域的阴影效果，包括内阴影和外阴影 */
}
```

定义容器内部标题的样式

```
/* 定义容器内部标题的样式 */
.container h2 {
  text-align: center; /* 设置文本居中对齐 */
  font-size: 10em; /* 设置字体大小为10倍（1000%） */
  line-height: 0.7em; /* 设置行高为0.7倍（70%） */
  color: #333; /* 设置字体颜色为深灰色 */
  margin-top: -80px; /* 与顶部的距离为-80像素，向上偏移标题以实现视觉居中 */
}
```

定义 .container 内部的 h2 元素中的 span 元素样式

```
/* 定义 .container 内部的 h2 元素中的 span 元素样式 */
.container h2 span {
  display: block;
  color: #00FFFF; /* 设置字体颜色为aliceblue */
  font-weight: 300; /* 设置字体粗细为300 */
  letter-spacing: 6px; /* 设置字符间距为6像素 */
  font-size: 0.2em; /* 设置字体大小为0.2倍（20%） */
}
```

定义具有 .countdown 类的元素样式

```
/* 定义具有 .countdown 类的元素样式 */
.countdown {
  display: flex; /* 使用弹性布局 */
  margin-top: 50px; /* 顶部外边距为50像素 */
}
```

定义具有 .countdown 类的子元素样式

```
/* 定义具有 .countdown 类的子元素样式 */
.countdown div {
  position: relative; /* 使用相对定位 */
  width: 100px; /* 宽度为100像素 */
  height: 100px; /* 高度为100像素 */
  line-height: 100px; /* 行高为100像素 */
  text-align: center; /* 文本居中对齐 */
  background: □ #333; /* 设置背景颜色为#333 */
  color: ■ #fff; /* 设置字体颜色为白色 */
  margin: 0 15px; /* 外边距，左右各为15像素 */
  font-size: 3em; /* 设置字体大小为3倍（300%） */
  font-weight: 500; /* 设置字体粗细为500 */
}
```

定义 .countdown div 元素的伪元素样式

```
/* 定义 .countdown div 元素的伪元素样式 */
.countdown div:before {
  content: ""; /* 清除伪元素内容 */
  position: absolute; /* 使用绝对定位 */
  bottom: -30px; /* 从底部偏移-30像素 */
  left: 0; /* 从左侧对齐 */
  width: 100%; /* 宽度100% */
  height: 35px; /* 高度35像素 */
  background: ■ #ff0; /* 设置背景颜色为#ff0 */
  color: □ #333; /* 设置字体颜色为#333 */
  font-size: 0.35em; /* 设置字体大小为0.35倍（35%） */
  line-height: 35px; /* 行高为35像素 */
  font-weight: 300; /* 设置字体粗细为300 */
}
```


定义 .container 中具有特定 ID 的元素的伪元素样式

```
/* 定义 .container 中具有特定 ID 的元素的伪元素样式 */  
.container #day:before {  
  content: "Days"; /* 内容为 "Days" */  
}  
  
.container #hour:before {  
  content: "Hours"; /* 内容为 "Hours" */  
}  
  
.container #minute:before {  
  content: "Minutes"; /* 内容为 "Minutes" */  
}  
  
.container #second:before {  
  content: "Seconds"; /* 内容为 "Seconds" */  
}
```

JavaScript 部分:

获取倒计时结束时间

```
// 获取倒计时结束时间
function getEndTime(myYear) {
    // 创建一个指定年份的日期对象，月份和时间设置为固定值
    let myEndTime = new Date("" + myYear + "/06/07 09:00:00");
    return myEndTime; // 返回结束时间
}
```

开始执行倒计时

```
// 开始执行倒计时
function countdown() {
    // 获取当前日期和时间
    let mydate = new Date();
    // 设置目标年份
    let year = "2024";
    // 获取倒计时结束时间
    let EndTime = getEndTime(year);
    // 获取当前时间
    let NowTime = new Date();
    // 检查是否已过倒计时结束时间
    if (EndTime.getTime() - NowTime.getTime() < 0) {
        // 如果已经过了，将目标年份增加 1
        year = mydate.getFullYear() + 1;
        // 获取新的倒计时结束时间
        EndTime = getEndTime(year);
    }
    // 计算时间差
    let t = EndTime.getTime() - NowTime.getTime();
    // 计算剩余的天数、小时、分钟和秒数
    let d = Math.floor(t / 1000 / 60 / 60 / 24);
    let h = Math.floor((t / 1000 / 60 / 60) % 24);
    let m = Math.floor((t / 1000 / 60) % 60);
    let s = Math.floor((t / 1000) % 60);
    // 更新页面上的倒计时显示
    document.getElementById("day").innerText = d;
    document.getElementById("hour").innerText = h;
    document.getElementById("minute").innerText = m;
    document.getElementById("second").innerText = s;
}
```

每隔 1 秒执行一次倒计时函数

```
// 每隔 1 秒执行一次倒计时函数
setInterval(countDown, 1000);
```

四、程序运行截图



五、遇到的问题与解决

已经解决的问题

(1) 获取目标日期和时间：

问题：需要确定高考的日期和时间。

解决方案：您可以在 JavaScript 中创建一个日期对象，设置高考的确切日期和时间。例如，使用 `new Date("2023-06-07T09:00:00")`。

(2) 实时更新倒计时：

问题：如何实时更新剩余时间并显示在页面上？

解决方案：使用 JavaScript 的 `setInterval` 函数来定期更新倒计时并将结果显示在页面上。在每个时间间隔（例如 1 秒），更新剩余时间并更新页面上的倒计时显示。

(3) 显示格式化的倒计时：

问题：如何将剩余时间以友好的格式显示给用户？

解决方案：您可以计算剩余的天、小时、分钟和秒，并将它们格式化为易于阅读的文本。例如，将它们显示为 "X 天 X 小时 X 分钟 X 秒"。

进一步可以改进的方面

(1) 处理过期情况：

问题：如何处理高考日期已经过去的情况？

解决方案：在每次更新倒计时时，检查当前日期是否超过了高考日期。如果是，可以显示一个适当的消息，如 "高考已经结束"。

(2) 浏览器兼容性：

问题：不同浏览器可能对某些 JavaScript 和 CSS 特性的支持有所不同。

解决方案：确保您的代码在不同主流浏览器上进行测试，并根据需要提供浏览器兼容性的解决方案，例如使用浏览器前缀。

(3) 更新年份：

问题：如果需要在每年都更新高考日期，如何自动处理？

解决方案：您可以在 JavaScript 中获取当前年份，然后将其与高考日期进行比较，以自动确定高考日期。这样，您无需手动更改日期。

六、代码附录

CountDown.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Countdown to NCEE</title>
    <link rel="stylesheet" href="./CountDown.css" />
  </head>
  <body>
    <div class="container">
      <h2><span>Countdown to NCEE</span> 2024</h2>
      <div class="countdown">
        <div id="day">NA</div>
        <div id="hour">NA</div>
        <div id="minute">NA</div>
        <div id="second">NA</div>
      </div>
    </div>
    <script src="./CountDown.js"></script>
  </body>
</html>
```

CountDown.css:

```
/* 引入 Google Fonts 中的 Poppins 字体，提供多种字重的选项 */
@import url("https://fonts.googleapis.com/css?family=Poppins:100,200,300,400,500,600,700,800,900");

/* 重置所有元素的外边距和内边距，使用 Poppins 字体作为默认字体 */
*{
  margin: 0;          /* 清除所有元素的外边距 */
  padding: 0;         /* 清除所有元素的内边距 */
  font-family: "Poppins", sans-serif; /* 设置所有文本的字体为 Poppins，如未定义则使用 sans-serif 作为备用字体 */
}

/* 设置页面背景样式，包括背景颜色和图像，固定背景并应用混合模式 */
body {
  background: #000 url(/images/a2.jpg); /* 设置背景颜色为黑色，同时加载背景图片 */
  background-attachment: fixed;          /* 固定背景，使其不随页面滚动而滚动 */
  background-blend-mode: hard-light;     /* 应用混合模式，以实现图像与背景的混合效果 */
}

/* 定义容器样式，通过绝对定位实现定位和居中显示内容 */
.container {
  position: absolute; /* 使用绝对定位，将容器相对于其最近的具有定位的祖先元素定位 */
  top: 80px;          /* 与顶部的距离为 80 像素 */
  left: 100px;        /* 与左侧的距离为 100 像素 */
  right: 100px;       /* 与右侧的距离为 100 像素 */
  bottom: 80px;       /* 与底部的距离为 80 像素 */
  background: url(/images/a2.jpg); /* 设置容器的背景图片 */
  background-attachment: fixed;     /* 固定容器的背景 */
  display: flex;                    /* 使用弹性布局容器 */
  justify-content: center;          /* 在水平方向上居中内容 */
  align-items: center;              /* 在垂直方向上居中内容 */
  flex-direction: column;          /* 设置内容的排列方向为纵向（垂直） */
  box-shadow: 0 50px 50px rgba(0, 0, 0, 0.5), 0 0 0 100px rgba(0, 0, 0, 0.1);
  /* 创建内容区域的阴影效果，包括内阴影和外阴影 */
}

/* 定义容器内部标题的样式 */
.container h2 {
  text-align: center; /* 设置文本居中对齐 */
  font-size: 10em;    /* 设置字体大小为 10 倍（1000%） */
  line-height: 0.7em; /* 设置行高为 0.7 倍（70%） */
  color: #333;        /* 设置字体颜色为深灰色 */
  margin-top: -80px;   /* 与顶部的距离为-80 像素，向上偏移标题以实现视觉居中 */
}

/* 定义 .container 内部的 h2 元素中的 span 元素样式 */
.container h2 span {
  display: block;
  color: aliceblue; /* 设置字体颜色为 aliceblue */
  font-weight: 300; /* 设置字体粗细为 300 */
  letter-spacing: 6px; /* 设置字符间距为 6 像素 */
  font-size: 0.2em; /* 设置字体大小为 0.2 倍（20%） */
}

/* 定义具有 .countdown 类的元素样式 */
.countdown {
  display: flex; /* 使用弹性布局 */
}
```

```

    margin-top: 50px;    /* 顶部外边距为 50 像素 */
}

/* 定义具有 .countdown 类的子元素样式 */
.countdown div {
    position: relative; /* 使用相对定位 */
    width: 100px;       /* 宽度为 100 像素 */
    height: 100px;      /* 高度为 100 像素 */
    line-height: 100px; /* 行高为 100 像素 */
    text-align: center; /* 文本居中对齐 */
    background: #333;    /* 设置背景颜色为#333 */
    color: #fff;         /* 设置字体颜色为白色 */
    margin: 0 15px;      /* 外边距，左右各为 15 像素 */
    font-size: 3em;      /* 设置字体大小为 3 倍（300%） */
    font-weight: 500;    /* 设置字体粗细为 500 */
}

/* 定义 .countdown div 元素的伪元素样式 */
.countdown div:before {
    content: "";         /* 清除伪元素内容 */
    position: absolute;  /* 使用绝对定位 */
    bottom: -30px;       /* 从底部偏移-30 像素 */
    left: 0;             /* 从左侧对齐 */
    width: 100%;         /* 宽度 100% */
    height: 35px;        /* 高度 35 像素 */
    background: #ff0;     /* 设置背景颜色为#ff0 */
    color: #333;         /* 设置字体颜色为#333 */
    font-size: 0.35em;    /* 设置字体大小为 0.35 倍（35%） */
    line-height: 35px;    /* 行高为 35 像素 */
    font-weight: 300;     /* 设置字体粗细为 300 */
}

/* 定义 .container 中具有特定 ID 的元素的伪元素样式 */
.container #day:before {
    content: "Days";     /* 内容为 "Days" */
}

.container #hour:before {
    content: "Hours";    /* 内容为 "Hours" */
}

.container #minute:before {
    content: "Minutes";  /* 内容为 "Minutes" */
}

.container #second:before {
    content: "Seconds";  /* 内容为 "Seconds" */
}

```

CountDown.js:

```
// 获取倒计时结束时间
function getEndTime(myYear) {
    // 创建一个指定年份的日期对象，月份和时间设置为固定值
    let myEndTime = new Date("" + myYear + "/06/07 09:00:00");
    return myEndTime; // 返回结束时间
}

// 开始执行倒计时
function countDown() {
    // 获取当前日期和时间
    let mydate = new Date();
    // 设置目标年份
    let year = "2024";
    // 获取倒计时结束时间
    let EndTime = getEndTime(year);
    // 获取当前时间
    let NowTime = new Date();

    // 检查是否已过倒计时结束时间
    if (EndTime.getTime() - NowTime.getTime() < 0) {
        // 如果已经过了，将目标年份增加 1
        year = mydate.getFullYear() + 1;
        // 获取新的倒计时结束时间
        EndTime = getEndTime(year);
    }

    // 计算时间差
    let t = EndTime.getTime() - NowTime.getTime();

    // 计算剩余的天数、小时、分钟和秒数
    let d = Math.floor(t / 1000 / 60 / 60 / 24);
    let h = Math.floor((t / 1000 / 60 / 60) % 24);
    let m = Math.floor((t / 1000 / 60) % 60);
    let s = Math.floor((t / 1000) % 60);

    // 更新页面上的倒计时显示
    document.getElementById("day").innerText = d;
    document.getElementById("hour").innerText = h;
    document.getElementById("minute").innerText = m;
    document.getElementById("second").innerText = s;
}

// 每隔 1 秒执行一次倒计时函数
setInterval(countDown, 1000);
```


第三题：

一、要求：

完成模板渲染和分页功能，实现后的结果如下：



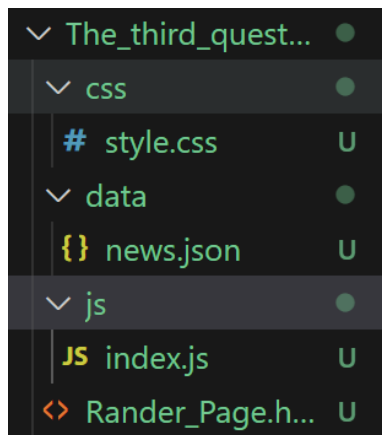
说明：程序主体代码在“作业 3 第三题”文件夹中，数据是“news.json”，阅读“index.js”，在指定位置完成 JS 代码 2 处：

完成代码 1：实现新闻模板内容的渲染

完成代码 2：实现输入页码回车跳转

二、项目文件简要概述

项目基本框架如下：



项目使用 html 和 css 构建了一个简单的消息界面

三、JavaScript 详解

渲染新闻函数

```
💡 渲染新闻函数
function renderNews() {
  news.innerHTML = " "; // 清空新闻容器以便重新渲染
  newsDataRender = newsData.slice((p - 1) * 5, 5 * p); // 每页要显示的数据,一页显示5条, p为当前页码
  console.log(newsDataRender); // 在控制台输出当前页面要显示的新闻数据

  let tpl = document.getElementById("tpl").innerHTML;
  //此处实现新闻模板内容的渲染

  // 使用 map 函数将每条新闻数据应用到模板中, 生成HTML字符串数组
  let html = newsDataRender
    .map((ndr) => {
      // 将模板中的占位符 {{title}} 和 {{content}} 替换为实际新闻数据
      let result = tpl
        .replace("{{title}}", ndr.title)
        .replace("{{content}}", ndr.content);
      return result;
    })
    .join(""); // 将生成的HTML字符串数组连接成一个完整的HTML字符串

  // 将生成的HTML字符串插入到新闻容器中, 用于渲染新闻内容
  news.innerHTML = html;
}
```

页码跳转

```
//页码跳转
let skipInput = document.querySelector(".skip input");
//此处实现输入页码回车跳转
skipInput.addEventListener("keyup", function (e) {
  p = skipInput.value; // 从输入框中获取用户输入的页码
  if (e.key === "Enter" && skipInput.value !== "") {
    // 如果用户按下回车键并且输入框的值不为空
    for (let j = 0; j < asAll.length; j++) {
      asAll[j].classList.remove("page-current"); // 移除所有页码的 "page-current" 类
    }
    renderNews(); // 渲染新闻内容
    asAll[p - 1].classList.add("page-current"); // 将当前页码的 "page-current" 类添加到页码按钮
  }
  if (p > asAll.length) {
    return; // 如果页码超出范围, 则不执行后续操作
  }
})
```

四、程序运行截图

实现新闻模板内容的渲染：



实现输入页码回车跳转：



五、遇到的问题与解决

已经解决的问题

问题 1: 实现新闻模板内容的渲染 (完成代码 1)

问题: 如何将新闻数据动态地渲染到页面上的模板中?

解决方案: 您可以使用模板字符串或模板引擎, 将新闻数据插入到模板中的占位符中。使用 JavaScript 来遍历新闻数据数组, 然后生成模板内容并插入到指定位置。

问题 2: 实现输入页码回车跳转 (完成代码 2)

问题: 如何捕获用户输入页码并在按下回车键时执行跳转操作?

解决方案: 您可以使用事件监听器来监视用户在输入框中的输入, 并在用户按下回车键时触发跳转操作。在事件处理程序中, 获取输入框的值, 然后执行相应的操作。

六、代码附录

Render_Page.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Module rendering and paging</title>

    <link rel="stylesheet" href="/css/style.css" />
  </head>
  <body>
    <!-- 新闻容器 -->
    <div class="news-container">
      <div class="news"></div>
    </div>
    <!-- 分页容器 -->
    <div class="page-container">
      <a class="page-item prev">&lt;</a>
      <div class="pagination"></div>
      <a class="page-item next">&gt;</a>
    </div>

    <!-- 定义新闻模板 -->
    <script type="text/template" id="tpl">
      <div class="news-item">
        <div class="news-title">{{title}}</div>
        <div class="news-content">{{content}}</div>
      </div>
    </script>

    <script src="/js/index.js"></script>
  </body>
</html>
```

Rander_Page.css:

```
body {  
    background-color: #f8f8f8;  
}
```

```
.news-container {  
    width: 75%;  
    margin: 50px auto;  
}
```

```
.news {  
    padding: 15px;  
    background-color: #fff;  
    height: 400px;  
}
```

```
.news-item {  
    margin: 20px 0px;  
}
```

```
.news-title {  
    text-align: left;  
    font-weight: bold;  
    font-size: 1rem;  
    padding-bottom: 10px;  
}
```

```
.page-container {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}
```

```
.page-item {  
    display: inline-block;  
    color: #a2a2a2;  
    line-height: 50px;  
    padding: 0 20px;  
    margin: 10px;  
    font-size: 16px;  
}
```

```
.page-current,  
.page-item:hover {
```

```
background: #31c27c;  
color: white;  
cursor: pointer;  
}
```

```
.skip input {  
width: 50px;  
}
```


Render_Page.js:

```
async function get(url) {
  let res = await fetch(url); // 调用 fetch(), 前面要加 await
  let json = await res.json(); // 将 res(服务器响应)解析为 JSON 格式
  return json;
}

let url = "./data/news.json";
let newsData = []; //存放新闻数据
let news = document.querySelector(".news"); //新闻容器
let pagination = document.querySelector(".pagination"); //分页容器
let asAll = []; //所有分页 a 链接
let pageCount = 0; //根据数据的长度计算总共几页
let newsDataRender = []; //每页要显示的数据
let p = 1; //根据 p 值显示每页的数据(p 为当前页码, 从 1 开始)

// 渲染新闻函数
function renderNews() {
  news.innerHTML = ""; // 清空新闻容器以便重新渲染
  newsDataRender = newsData.slice((p - 1) * 5, 5 * p); // 每页要显示的数据,一页显示 5 条, p 为当前页码
  console.log(newsDataRender); // 在控制台输出当前页面要显示的新闻数据

  let tpl = document.getElementById("tpl").innerHTML;
  //此处实现新闻模板内容的渲染

  // 使用 map 函数将每条新闻数据应用到模板中, 生成 HTML 字符串数组
  let html = newsDataRender
    .map((ndr) => {
      // 将模板中的占位符 {{title}} 和 {{content}} 替换为实际新闻数据
      let result = tpl
        .replace("{{title}}", ndr.title)
        .replace("{{content}}", ndr.content);
      return result;
    })
    .join(""); // 将生成的 HTML 字符串数组连接成一个完整的 HTML 字符串

  // 将生成的 HTML 字符串插入到新闻容器中, 用于渲染新闻内容
  news.innerHTML = html;
}

// 渲染分页器函数
function renderPager() {
  for (let i = 1; i <= pageCount; i++) {
    pagination.innerHTML += `<a class="page-item">${i}</a>`; //模板字符串
  }
  pagination.innerHTML += `<span class="skip">跳转至 <input type="text"> </span>页`;

  asAll = pagination.querySelectorAll("a"); //所有分页链接
  console.log(asAll);
  //页面刚进来时第一页高亮
  asAll[0].classList.add("page-current");

  //遍历所有分页
  asAll.forEach((item, index) => {
    //点击页数
    item.onclick = function () {

```

```

        for (let j = 0; j < asAll.length; j++) {
            asAll[j].classList.remove("page-current"); //去除所有选中项
        }
        this.classList.add("page-current");
        p = index + 1; //点击页数，改变 p 的值，以改变这个页面要显示的数据，达到分页的效果
        renderNews(); //重新渲染页面
    };
});
}

```

//改变选中页高亮函数(点击上一页下一页时调用)

```

function changePageClass(p) {
    for (let j = 0; j < asAll.length; j++) {
        asAll[j].classList.remove("page-current");
    }
    asAll[p - 1].classList.add("page-current");
}

```

//主函数

```

async function run() {
    //读取 json
    newsData = await get(url); //调用 get
    console.log(newsData);
    pageCount = Math.ceil(newsData.length / 5); //根据数据的长度计算总共几页

```

renderNews(); //渲染新闻模板

renderPager(); //渲染分页器

//上一页

```

let prev = document.querySelector(".prev");
prev.onclick = function (e) {
    if (p <= 1) {
        return;
    } else {
        p = p - 1;
        changePageClass(p);
        renderNews();
    }
};

```

//下一页

```

let next = document.querySelector(".next");
next.onclick = function () {
    if (p >= asAll.length) {
        return;
    }
    p = p + 1;
    changePageClass(p);
    renderNews();
};

```

//页码跳转

```

let skipInput = document.querySelector(".skip input");
//此处实现输入页码回车跳转
skipInput.addEventListener("keyup", function (e) {
    p = skipInput.value; // 从输入框中获取用户输入的页码
    if (e.key === "Enter" && skipInput.value !== "") {
        // 如果用户按下回车键并且输入框的值不为空

```

```
    for (let j = 0; j < asAll.length; j++) {
        asAll[j].classList.remove("page-current"); // 移除所有页码的 "page-current" 类
    }
    renderNews(); // 渲染新闻内容
    asAll[p - 1].classList.add("page-current"); // 将当前页码的 "page-current" 类添加到页码按钮
}
if (p > asAll.length) {
    return; // 如果页码超出范围，则不执行后续操作
}
});
}

//执行
run();
```