河南中医药大学信息技术学院（智能医疗行业学院）智能医学工程专业《互联网医疗服务开发》课程

第06章：JavaScript应用

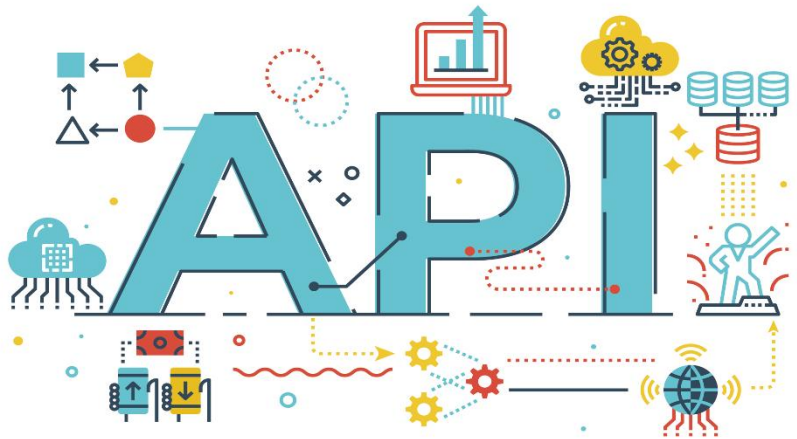
冯顺磊

河南中医药大学信息技术学院（智能医疗行业学院）
河南中医药大学信息技术学院智能医疗教研室
https://aitcm.hactcm.edu.cn

2025/3/7

本章概要

- AJAX
- 文件
- 绘图
- 本地存储
- 地理定位



1. AJAX

- AJAX（Asynchronous JavaScript and XML）即异步的 JavaScript 和 XML，它并不是一种新的编程语言，而是一种在无需重新加载整个网页的情况下，与服务器进行异步通信并更新部分网页的技术。虽然名字中包含 XML，但现在更多使用 JSON 作为数据交换格式。

- 原理
  - 传统的网页（不使用 AJAX）如果需要更新内容，必须重载整个网页页面。而 AJAX 允许在不刷新整个页面的前提下，与服务器进行异步通信并更新部分网页。其核心是利用浏览器提供的 XMLHttpRequest 对象（现代也可用 fetch API）来发送 HTTP 请求，在后台与服务器进行数据交换，然后根据服务器返回的数据更新网页的部分内容。

- 工作流程
  1. 创建 XMLHttpRequest 对象：这是 AJAX 的基础，通过该对象可以与服务器进行通信。
  2. 打开连接：指定请求的方法（如 GET、POST）、请求的 URL 等信息。
  3. 发送请求：将请求发送到服务器。
  4. 监听状态变化：通过监听 XMLHttpRequest 对象的状态变化，判断请求是否完成。
  5. 处理响应：当请求完成后，根据服务器返回的状态码和数据进行相应的处理。

1. AJAX

- 使用 XMLHttpRequest 对象实现 AJAX

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

<body>
  <button id="fetchDataButton">获取数据</button>
  <div id="result"></div>
  <script>
    const fetchDataButton = document.getElementById('fetchDataButton');
    const resultDiv = document.getElementById('result');

    fetchDataButton.addEventListener('click', function () {
      // 1. 创建 XMLHttpRequest 对象
      const xhr = new XMLHttpRequest();

      // 2. 打开连接
      xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1', true);

      // 3. 监听状态变化
      xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
          // 4. 处理响应
          const response = JSON.parse(xhr.responseText);
          resultDiv.innerHTML = `标题: ${response.title}`;
        }
      };

      // 5. 发送请求
      xhr.send();
    });
  </script>
</body>

</html>
```

1. AJAX

- 使用 fetch API 实现 AJAX：fetch 是现代 JavaScript 提供的更简洁、更强大的 API，用于发起网络请求。

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

<body>
  <button id="fetchDataButton">获取数据</button>
  <div id="result"></div>
  <script>
    const fetchDataButton = document.getElementById('fetchDataButton');
    const resultDiv = document.getElementById('result');

    fetchDataButton.addEventListener('click', function () {
      // 发送请求
      fetch('https://jsonplaceholder.typicode.com/todos/1')
        .then(response => {
          if (!response.ok) {
            throw new Error('网络响应不正常');
          }
          return response.json();
        })
        .then(data => {
          // 处理响应数据
          resultDiv.innerHTML = `标题: ${data.title}`;
        })
        .catch(error => {
          // 处理错误
          console.error('请求出错:', error);
        });
    });
  </script>
</body>

</html>
```

1. AJAX

- AJAX 的优点
  - 无刷新更新页面：用户可以在不刷新整个页面的情况下与服务器进行交互，提高了用户体验。
  - 异步通信：在请求服务器数据时，不会阻塞用户的其他操作，页面仍然可以响应用户的交互。
  - 减少数据传输量：只请求和更新需要的数据，减少了服务器和客户端之间的数据传输量，提高了性能。

- AJAX 的缺点
  - 浏览器兼容性问题：早期的浏览器对 AJAX 的支持存在差异，需要进行兼容性处理。
  - 安全问题：由于 AJAX 可以在后台与服务器进行通信，可能存在跨站请求伪造（CSRF）等安全风险。
  - 搜索引擎优化（SEO）困难：由于 AJAX 动态加载的内容可能无法被搜索引擎爬虫抓取，对网站的 SEO 有一定影响。

示例6-1-1：AJAX示例

1. AJAX

- 实际应用案例
  - 实时搜索建议：在搜索引擎或电商网站的搜索框中，当用户输入关键词时，会实时显示相关的搜索建议。这一过程利用 AJAX 实现，用户在输入过程中，浏览器会不断向服务器发送异步请求，服务器根据输入内容返回相关的搜索建议，然后在不刷新页面的情况下更新到搜索框下方。
  - 动态加载内容：在社交媒体网站、新闻网站等，当用户滚动页面到底部时，会自动加载更多的内容，而不需要用户手动点击“加载更多”按钮或刷新页面。这是通过 AJAX 异步请求服务器获取新的内容，然后将其添加到页面中实现的。
  - 表单异步提交：在用户提交表单时，使用 AJAX 可以在不刷新整个页面的情况下将表单数据发送到服务器，并根据服务器的响应更新页面的部分内容，例如显示提交结果。
  - 实时数据更新：在金融交易网站、股票行情网站等，需要实时显示股票价格、汇率等数据。AJAX 可以定期向服务器发送请求，获取最新的数据，并更新页面上的显示内容。
  - 在线聊天应用：在在线聊天应用中，用户发送的消息可以通过 AJAX 异步发送到服务器，同时客户端也会定期向服务器请求新的消息，实现实时聊天的效果。

2. 文件

- 在 JavaScript 里，文件对象主要在浏览器环境中使用，它通常与用户选择文件的操作相关，代表着用户从本地系统选择的文件。
  - 获取文件对象：在浏览器中，一般通过 HTML 的 <input type="file"> 元素让用户选择文件，然后使用 JavaScript 访问所选文件对应的文件对象。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Get File Object</title>
</head>

<body>
    <input type="file" id="fileInput">
    <script>
        const fileInput = document.getElementById('fileInput');
        fileInput.addEventListener('change', function (event) {
            const files = event.target.files;
            if (files.length > 0) {
                const firstFile = files[0];
                console.log('Selected file:', firstFile);
            }
        });
    </script>
</body>

</html>
```

2. 文件

- 文件对象的属性
  - name：该属性返回文件的名称，包含文件扩展名。
  - size：返回文件的大小，单位是字节。
  - type：返回文件的 MIME 类型，例如 image/jpeg、text/plain 等。
  - lastModified：返回文件最后修改的时间戳（从 1970 年 1 月 1 日 00:00:00 UTC 到文件最后修改时间的毫秒数）。

```
const file = files[0];
console.log('File name:', file.name);
console.log('File size:', file.size, 'bytes');
console.log('File type:', file.type);
console.log('Last modified:', lastModifiedDate.toLocaleString());
```

2. 文件

- FileReader 是 JavaScript 中用于在浏览器环境下异步读取用户本地文件内容的工具
  - 由于浏览器的安全限制，不能直接访问本地文件系统，但可以借助 FileReader 配合 <input type="file"> 元素让用户选择文件后读取其内容。
  - 创建 FileReader 对象：创建 FileReader 对象非常简单，只需使用 new 关键字调用构造函数即可。

```
const reader = new FileReader();
```

2. 文件

- FileReader 的方法
  - readAsText(file[, encoding])：用于以文本形式读取文件内容。
    - file：必需，要读取的 File 或 Blob 对象。
    - encoding：可选，指定文件的字符编码，默认值为 UTF-8。
  - readAsDataURL(file)：将文件读取为 Data URL 格式，常用于预览图片等场景。Data URL 是一种将文件内容以 Base64 编码嵌入到 URL 中的方式。
    - file：必需，要读取的 File 或 Blob 对象。
  - readAsArrayBuffer(file)：将文件读取为 ArrayBuffer 对象，ArrayBuffer 是一种用于表示通用的、固定长度的二进制数据缓冲区。
    - file：必需，要读取的 File 或 Blob 对象。
  - abort()：中止文件读取操作。

2. 文件

- FileReader 的事件
  - onloadstart：当读取操作开始时触发。
  - onprogress：在读取过程中周期性触发，可用于显示读取进度。
  - onload：当读取操作成功完成时触发，此时可以通过 event.target.result 获取文件内容。
  - onerror：当读取操作发生错误时触发，可通过 event.target.error 获取错误信息。
  - onloadend：无论读取操作成功还是失败，在读取结束时触发。

2. 文件

- Blob
  - 在 JavaScript 里，Blob（Binary Large Object，二进制大对象）是一种用于表示不可变的、原始数据的类文件对象。它可以包含任意类型的二进制数据，如文本、图像、音频等。
  - 创建 Blob 对象：使用 Blob 构造函数来创建 Blob 对象，其语法如下。

```
new Blob(array, options);
```

    - array：这是一个由 ArrayBuffer、ArrayBufferView、Blob、DOMString 等对象构成的数组，数组中的元素会按顺序拼接起来组成 Blob 的数据内容。
    - options：这是一个可选的对象，可用于指定 Blob 的一些属性，常用的属性是 type，用于指定 Blob 的 MIME 类型，默认值为空字符串。

2. 文件

- Blob
  - Blob 对象的属性
    - size：此属性返回 Blob 对象所包含数据的大小，单位是字节。
    - type：此属性返回 Blob 对象的 MIME 类型，如果 MIME 类型未知，则返回空字符串。
  - Blob 对象的方法
    - slice(start, end, contentType)：该方法用于创建一个新的 Blob 对象，新对象包含原始 Blob 从 start 到 end（不包含 end）位置的数据，并且可以指定新 Blob 的 MIME 类型。
      - start：可选参数，指定开始切片的位置，默认为 0。
      - end：可选参数，指定结束切片的位置，默认为 Blob 的大小。
      - contentType：可选参数，指定新 Blob 的 MIME 类型，默认为原始 Blob 的 MIME 类型。

```
const text = 'Hello, Blob!';
const blob = new Blob([text], { type: 'text/plain' });
const slicedBlob = blob.slice(0, 5, 'text/plain');

const reader = new FileReader();
reader.onload = function (e) {
    const content = e.target.result;
    console.log('Sliced Blob content:', content);
};
reader.readAsText(slicedBlob);
```

2. 文件

- 文件上传
  - 普通文件上传
    - 在 HTML 中创建一个文件输入框
    - 在 JavaScript 中获取文件输入框的引用，并在其上设置事件监听器

```
// 创建文件输入框
<input type="file" id="fileInput">

// 设置事件监听器
var fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', function () {
  // 使用XMLHttpRequest上传
  var file = fileInput.files[0];
  var formData = new FormData();
  formData.append('file', file);

  var xhr = new XMLHttpRequest();
  xhr.open('POST', 'url', true);
  xhr.onload = function () {
  if (xhr.status === 200) {
    console.log('upload success');
  }
  };
  xhr.send(formData);
});
```

```
// 创建文件输入框
<input type="file" id="fileInput">

// 设置事件监听器
var fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', function () {
  // 使用fetch上传
  var file = fileInput.files[0];
  var formData = new FormData();
  formData.append('file', file);

  fetch('url', {
    method: 'POST',
    body: formData
  }).then(response => {
    if (response.ok) {
      console.log('upload success');
    }
  });
});
```

2. 文件

- 大文件上传：在上传大文件时，通常采用分块上传的方式。将大文件分成若干个块，每块一个 HTTP 请求上传。
  - 用户选择文件。
  - 将文件分成若干块。
  - 对于每一块，向服务器发送 HTTP 请求上传。
  - 服务器接收到文件块后，将其存储在服务器上。
  - 在所有块上传完成后，服务器将所有块合并成一个完整的文件。
- JavaScript 可以使用 File API（File 和 Blob 对象）来实现文件的读取和上传。

```
// 创建文件输入框
<input type="file" id="file-input">

// 上传文件块
function uploadChunk(file, start, end, chunk) {
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '/upload', true);
    xhr.setRequestHeader('Content-Type', 'application/octet-stream');
    xhr.setRequestHeader('Content-Range', 'bytes ' + start + '-' + end + '/' +
file.size);
    xhr.send(chunk);
}

// 上传文件
function uploadFile(file) {
    var chunkSize = 1 * 1024 * 1024; // 分块大小为1MB
    var chunks = Math.ceil(file.size / chunkSize); // 计算分块数
    var currentChunk = 0; // 当前分块
    var start, end;
    while (currentChunk < chunks) {
        start = currentChunk * chunkSize;
        end = start + chunkSize >= file.size ? file.size : start + chunkSize;
        var chunk = file.slice(start, end);
        uploadChunk(file, start, end, chunk);
        currentChunk++;
    }
}

// 监听文件选择事件
document.getElementById('file-input').addEventListener('change', function(e) {
    var file = e.target.files[0];
    if (file) {
        uploadFile(file);

});
```

2. 文件

- 在实际应用中还需要考虑以下几点：
  - 如果服务器端支持断点续传，可以在服务器端记录已经上传的文件块，避免重复上传。
  - 需要考虑如何处理上传失败的文件块，是否需要重试。
  - 在上传过程中需要提供进度条，让用户了解上传进度。
  - 在上传完成后需要有反馈，告知用户上传是否成功。
  - 服务器端如何处理上传的文件块，将其合并成一个完整的文件。
  - 服务器端存储空间的问题。可以使用分布式文件系统（如 HDFS）或云存储（如 Amazon S3）来存储上传的文件。
  - 文件块上传顺序、文件块校验、断点续传等问题。
  - 通过分块上传的方式，可以将大文件分成若干块上传，避免一次性上传大文件造成的超时或者内存不足的问题，同时也方便实现断点续传和上传进度的显示。

```java
import java.io.IOException;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItemIterator;
import org.apache.commons.fileupload.FileItemStream;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.IOUtils;

public class FileUploadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // 保存所有分块数据，使用ConcurrentHashMap保证线程安全
    private Map<String, byte[]> chunks = new ConcurrentHashMap<>();

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        if (!ServletFileUpload.isMultipartContent(request)) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Not a
multipart request");
            return;
        }

        ServletFileUpload upload = new ServletFileUpload();
        try {
            FileItemIterator iter = upload.getItemIterator(request);
            while (iter.hasNext()) {
                FileItemStream item = iter.next();
                if (!item.isFormField()) {
                    // 处理文件分块
                    processFilePart(item);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2. 文件

- 文件下载
  - 使用 Blob 对象和 URL.createObjectURL：当需要动态生成文件内容并下载时，可以使用 Blob 对象来创建文件，再结合 URL.createObjectURL 方法生成一个临时的 URL，最后通过创建 <a> 标签并模拟点击来触发下载。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic File Download with Blob</title>
</head>

<body>
    <button id="downloadButton">Download File</button>
    <script>
        const downloadButton = document.getElementById('downloadButton');
        downloadButton.addEventListener('click', () => {
            const content = 'This is a sample text for the downloaded file.';
            const blob = new Blob([content], { type: 'text/plain' });
            const url = URL.createObjectURL(blob);

            const a = document.createElement('a');
            a.href = url;
            a.download = 'sample.txt';
            a.click();

            URL.revokeObjectURL(url);
        });
    </script>
</body>

</html>
```

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Download Image File</title>
</head>

<body>
    <button id="downloadImageButton">Download Image</button>
    <script>
        const downloadImageButton =
document.getElementById('downloadImageButton');
        downloadImageButton.addEventListener('click', async () => {
            try {
                const response = await fetch('https://example.com/image.jpg');
                const blob = await response.blob();
                const url = URL.createObjectURL(blob);

                const a = document.createElement('a');
                a.href = url;
                a.download = 'downloaded-image.jpg';
                a.click();

                URL.revokeObjectURL(url);
            } catch (error) {
                console.error('Error downloading image:', error);
            }
        });
    </script>
</body>

</html>
```

示例6-3-1：文件上传示例

3. 绘图 3.1 属性与方法

- Canvas元素在页面上提供一块像画布一样无色透明的区域，可通过Javascript脚本绘制图形。
- 在HTML页面上定义Canvas元素除了可以指定id、style、class、hidden等通用属性之外，还可以指定以下2个属性。
  - height：设置画布组件的高度。
  - width：设置画布组件的宽度。
- 在画布上绘制图形必须经过以下三个步骤。
  - 获取Canvas对应的DOM对象，得到一个Canvas对象。
  - 调用Canvas对象的getContext()方法，得到CanvasRenderingContext2D对象（可绘制图形）。
  - 调用CanvasRenderingContext2D对象方法绘图。绘图方法有很多种，如常用的填充矩形区域的方法sfillRect()、绘制矩形边框的strokeRect()等。

3. 绘图 3.1 属性与方法

- Canvas中，绘图API主要通过2D绘图上下文（CanvasRenderingContext2D）提供，其方法与属性。

方法	简要说明	方法	简要说明
void arc(float x, float y, float radius, float startAngel, endAngle, boolen counterclockwise)	向Canvas的当前路径上添加一段弧	void fillRect(float x, float y, float width, float height)	填充一个矩形区域
void arcTo(float x1, float y1, float x2, float y2, float radius)	向Canvas的当前路径上添加一段弧，与前一个方法相比，只是定义弧的方式不同	void fillText(String text, float x, float y [, float maxWidth])	填充字符串
void beginPath()	开始定义路径	void lineTo(float x, float y)	把Canvas的当前路径从当前结束点连接到x、y的对应点
void closePath()	关闭前面定义的路径	void moveTo(float x, float y)	把Canvas的当前路径结束点移动到x、y对应的点
void bezierCurveTo(float cpX1, float cpY1, float cpX2, float cpY2, float x, float y)	向Canvas的当前路径上添加一段贝塞尔曲线	void quadraticCurveTo(float cpX, float cpY, float x, float y)	向Canvas当前路径上添加一段二次曲线
void clearRect(float x, float y, float width, float height)	擦除制定区域上绘制的图形	void rect(float x, float y, float width, float height)	向Canvas当前路径上添加一个矩形
void clip()	从画布上裁切一块出来	void stroke()	沿着Canvas当前路径绘制边框
Canvas Gradient createLinearGradient(float xStart, float yStart, float xEnd, float yEnd)	创建一个线性渐变	void strokeRect(float x, float y, float width, float height)	绘制一个矩形边框
CanvasPatterm careatePattern(image image, string style)	创建一个图形平铺	void strokeText(string text, float x, float y, float width [,float maxWidth])	绘制字符串边框
Canvas Gradient createLinearGradient(float xStart, float yStart, float radiusStart, float xEnd, float yEnd, float radiusEnd)	创建一个圆形渐变	void save()	保存当前绘图状态
void drawImage(Image image, float x, float y)	绘制位图	void restore()	恢复之前保存的绘图状态
void drawImage(Image image, float x, float y, float width, float height)		void rotate(float angle)	旋转坐标系统
void drawImage(Image image, integer sx, integer xy, integer sw, integer sh, float dx, float dy, float dw, float dh)		void scale(float sx, float sy)	缩放坐标系统
void fill()	填充Canvas的当前路径	void translate(float dy, float dy)	平移坐标系统

3. 绘图 3.1 属性与方法

- Canvas中，绘图API主要通过2D绘图上下文（CanvasRenderingContext2D）提供，其方法与属性。

属性名	简要说明	属性名	简要说明
fillStyle	设置填充路径时所用的填充风格，该属性支持三种类型的值：	lineJoin	设置线条连接点的风格。该属性支持如下3个值：
	符合颜色格式的字符串值，表明使用纯色填充		meter，默认属性值，线条连接点形如箭头
	CanvasGeadient，表明使用渐变填充		round，线条连接点形如圆角
	CanvasPattern，表明填充绘图的模式		bevel，线条连接点形如平角
strokeStyle	设置绘制路径时所用的填充风格，该属性支持三种类型的值：	miterLimit	把lineJoin树形设置为meter风格时，该属性控制锐角箭头的长度
	符合颜色格式的字符串值，表明使用纯色填充	linewidth	设置笔触线条宽度
	CanvasGeadient，表明使用渐变填充	shadowBlur	设置阴影的模糊程度
	CanvasPattern，表明填充路径的模式	shadowColor	设置阴影的颜色
Font	设置绘制字符串时所用的字体	shadowOffsetX	设置阴影在X方向上的偏移
globalAlpha	设置全局透明度	shadowOffsetY	设置阴影在Y方向上的偏移
globalCompositeOperation	设置全局叠加效果	textAlign	设置绘制字符串的水平对齐方式，该属性支持start、end、left、right、center等属性值
lineCap	设置线段端点的绘图形状。该属性支持如下三个值：	textBaseAlign	设置绘制字符串的垂直对齐方式，该属性支持top、hanging、middle、alphabetic、idecgraphic、bottom等属性值
	“butt”，默认的属性值，该属性值指定不绘制端点，线条结尾处直接结束		
	“round”，该属性值指定绘制圆形端点。线条结尾处绘制一个直径为线条宽度的半圆		
	“square”，该属性值制定绘制正反形端点。线条结尾处绘制半个边长为线条宽度的正方形。这种形状的端点“butt”形状端点相似，但线条略长		

3. 绘图 3.2 矩形

- 矩形
  - 绘制填充矩形：使用 fillRect() 方法可以直接在画布上绘制一个填充的矩形。

```
ctx.fillRect(x, y, width, height);
```

  - x：矩形左上角的 x 坐标。
  - y：矩形左上角的 y 坐标。
  - width：矩形的宽度。
  - height：矩形的高度

3. 绘图 3.2 矩形

- 矩形
  - 绘制描边矩形：使用 strokeRect() 方法可以绘制一个只有边框的矩形。

```
ctx.strokeRect(x, y, width, height);
```

  - x：矩形左上角的 x 坐标。
  - y：矩形左上角的 y 坐标。
  - width：矩形的宽度。
  - height：矩形的高度。

3. 绘图 3.3 路径

- 路径：由一系列的点和连接这些点的线段或曲线组成的。通过路径可以创建复杂的图形，如多边形、不规则图形等。
  - 定义路径，调用CanvasRenderingContext2D对象的beginPath()方法；
  - 定义子路径，可以使用的方法有arc()、arcTo()、bezierCurveTo()、lineTo()、moveTo()、quadraticCurveTo()、rect()；
  - 关闭路径，调用CanvasRenderingContext2D对象的closePath()方法；
  - 填充路径或绘制路径，调用CanvasRenderingContext2D对象的fill()方法或stroke()方法。

3. 绘图 3.3 路径

- beginPath()：开始一个新的路径。在绘制多个独立图形时，使用该方法可以避免不同图形的路径相互干扰。每次调用 beginPath() 后，后续的绘图命令会创建一个新的独立路径。

```
ctx.beginPath();
```

- moveTo(x, y)：将绘图的起始点移动到指定的坐标 (x, y)，不会绘制任何线条，常用于开始绘制新的线段或图形时定位起始位置。

```
ctx.moveTo(x, y);
```

3. 绘图 3.3 路径

- lineTo(x, y)：从当前画笔位置绘制一条直线到指定的坐标 (x, y)。

```
ctx.lineTo(x, y);
```

- arc(x, y, radius, startAngle, endAngle, anticlockwise)：绘制一个圆弧路径。圆心坐标为 (x, y)，半径为 radius，startAngle 和 endAngle 分别表示圆弧的起始和结束角度（以弧度为单位），anticlockwise 是一个布尔值，true 表示逆时针绘制，false 表示顺时针绘制（默认值为 false）。

```
ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);
```

3. 绘图 3.3 路径

- arcTo(x1, y1, x2, y2, radius)：根据给定的控制点和半径绘制一段圆弧，该圆弧与当前点到第一个控制点 (x1, y1) 的直线以及第一个控制点到第二个控制点 (x2, y2) 的直线相切。

```
ctx.arcTo(x1, y1, x2, y2, radius);
```

- rect(x, y, width, height)：绘制一个矩形路径，矩形的左上角坐标为 (x, y)，宽度为 width，高度为 height。

```
ctx.rect(x, y, width, height);
```

3. 绘图 3.3 路径

- quadraticCurveTo(cpx, cpy, x, y)：绘制一条二次贝塞尔曲线。(cpx, cpy) 是一个控制点，用于确定曲线的形状，(x, y) 是曲线的终点。

```
ctx.quadraticCurveTo(cpx, cpy, x, y);
```

- bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)：绘制一条三次贝塞尔曲线。(cp1x, cp1y) 和 (cp2x, cp2y) 是两个控制点，用于确定曲线的形状，(x, y) 是曲线的终点。

```
ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);
```

3. 绘图 3.3 路径

- closePath()：closePath() 方法用于闭合当前正在绘制的路径。它会从当前点绘制一条直线连接到路径的起始点，从而形成一个封闭的图形。不过，该方法只是在逻辑上闭合路径，并不会实际绘制或填充图形，后续仍需使用 stroke() 或 fill() 来完成绘制或填充操作。

```
ctx.closePath();
```

- fill()：fill() 方法用于填充当前路径所围成的区域。填充的颜色由 fillStyle 属性决定，该属性可以设置为 CSS 颜色值、渐变对象或图案对象。

```
ctx.fill();
```

3. 绘图 3.3 路径

- stroke()：stroke() 方法用于绘制当前路径的边框。边框的颜色由 strokeStyle 属性决定，边框的宽度由 lineWidth 属性决定。

```
ctx.stroke();
```

3. 绘图 3.4 文字

- Canvas 中，可以使用 JavaScript 在画布上绘制文字。
  - fillText(text, x, y, maxWidth)：用于在指定位置填充文本，text 是要绘制的文本内容，(x, y) 是文本起始点的坐标，maxWidth 是可选参数，用于指定文本的最大宽度，如果文本超过该宽度，会自动进行缩放。

```
ctx.fillText(text, x, y [, maxWidth]);
```

  - strokeText()：用于在指定位置绘制文本的边框，参数含义与 fillText 相同。

```
ctx.strokeText(text, x, y [, maxWidth]);
```

3. 绘图 3.4 文字

- Canvas 中，可以将图片绘制到画布上，还能对图片进行缩放、裁剪、旋转等操作。
  - 加载图片：在使用 Canvas 绘制图片之前，需要先创建一个 Image 对象，并加载图片资源。可以通过监听 Image 对象的 onload 事件来确保图片加载完成后再进行绘制操作。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas Image Loading</title>
</head>

<body>
    <canvas id="myCanvas" width="400" height="300"></canvas>
    <script>
        const canvas = document.getElementById('myCanvas');
        const ctx = canvas.getContext('2d');

        const img = new Image();
        img.src = 'example.jpg'; // 替换为实际的图片路径

        img.onload = function () {
            // 图片加载完成后进行绘制操作
            ctx.drawImage(img, 0, 0);
        };
    </script>
</body>

</html>
```

3. 绘图 3.5 图片

- 绘制图片：使用 drawImage() 方法将图片绘制到画布上，该方法有三种不同的重载形式。
  - 基本绘制：ctx.drawImage(image, dx, dy);

```
ctx.drawImage(image, dx, dy);
```

- image：必需，要绘制的 Image 对象。
- dx：图片在画布上的左上角 x 坐标。
- dy：图片在画布上的左上角 y 坐标。

3. 绘图 3.5 图片

- 绘制图片：使用 drawImage() 方法将图片绘制到画布上，该方法有三种不同的重载形式。
  - 绘制并缩放图片：ctx.drawImage(image, dx, dy);

```
ctx.drawImage(image, dx, dy, dWidth, dHeight);
```

- dWidth：绘制到画布上的图片宽度。
- dHeight：绘制到画布上的图片高度。

3. 绘图 3.5 图片

- 绘制图片：使用 drawImage() 方法将图片绘制到画布上，该方法有三种不同的重载形式。
  - 裁剪并绘制图片：ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);

```
ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);
```

- sx：裁剪区域的左上角 x 坐标。
- sy：裁剪区域的左上角 y 坐标。
- sWidth：裁剪区域的宽度。
- sHeight：裁剪区域的高度。

3. 绘图 3.5 图片

- 图片的旋转和变形：可结合 Canvas 的变换方法（如 translate()、rotate() 等）对图片进行旋转和变形操作。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas Image Rotation</title>
</head>

<body>
    <canvas id="myCanvas" width="400" height="300"></canvas>
    <script>
        const canvas = document.getElementById('myCanvas');
        const ctx = canvas.getContext('2d');

        const img = new Image();
        img.src = 'example.jpg';

        img.onload = function () {
            ctx.save(); // 保存当前画布状态
            ctx.translate(200, 150); // 将原点移动到画布中心
            ctx.rotate(Math.PI / 4); // 旋转 45 度
            ctx.drawImage(img, -img.width / 2, -img.height / 2); // 绘制图片
            ctx.restore(); // 恢复画布状态
        };
    </script>
</body>

</html>
```

3. 绘图 3.6 坐标变换

- 保存和恢复画布状态
  - save()：保存当前画布的状态到一个栈中。
  - restore()：从栈中弹出上一次保存的画布状态并恢复。

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');

ctx.save(); // 保存当前状态
// 进行一系列操作
ctx.restore(); // 恢复到之前保存的状态
```

3. 绘图 3.6 坐标变换

- 平移（translate()）：translate() 方法将画布的原点移动到指定的坐标 (x, y)。后续的绘图操作将基于新的原点进行。

```
ctx.translate(x, y);
```

- 旋转（rotate()）：rotate() 方法将画布绕当前原点旋转指定的角度（以弧度为单位）。旋转操作会影响后续所有的绘图操作。

```
ctx.rotate(angle);
```

3. 绘图 3.6 坐标变换

- 缩放（scale()）：scale() 方法对画布的坐标系进行缩放。x 和 y 分别是水平和垂直方向的缩放比例。

```
ctx.scale(x, y);
```

3. 绘图 3.7 常用属性

- 颜色与样式相关属性
  - fillStyle：设置图形填充的颜色、渐变或图案。可以使用 CSS 颜色值（如颜色名称、十六进制值、RGB 值等），也可以使用 CanvasGradient 或 CanvasPattern 对象。
  - strokeStyle：设置图形描边的颜色、渐变或图案，用法与 fillStyle 类似。
  - lineWidth：设置线条的宽度，单位为像素。
  - lineCap：设置线条端点的样式，可选值有 'butt'（默认值，线条端点为方形，不超出线条本身）、'round'（线条端点为圆形）和 'square'（线条端点为方形，超出线条本身半个线宽）。
  - lineJoin：设置两条线条相交处的样式，可选值有 'miter'（默认值，尖角连接）、'round'（圆角连接）和 'bevel'（斜角连接）。

- 文本相关属性
  - font：设置绘制文本时使用的字体样式，语法和 CSS 的 font 属性类似。
  - textAlign：设置文本的水平对齐方式，可选值有 'start'（默认值，与文本方向开始处对齐）、'end'、'left'、'right' 和 'center'。
  - textBaseline：设置文本的垂直对齐方式，可选值有 'top'、'hanging'、'middle'、'alphabetic'（默认值）、'ideographic' 和 'bottom'。

3. 绘图 3.7 常用属性

- 阴影相关属性
  - shadowColor：设置阴影的颜色。
  - shadowBlur：设置阴影的模糊程度，值越大，阴影越模糊。
  - shadowOffsetX 和 shadowOffsetY：分别设置阴影在水平和垂直方向上的偏移量。

- 全局合成相关属性
  - globalAlpha：设置绘制图形的全局透明度，取值范围是 0.0（完全透明）到 1.0（完全不透明）。
  - globalCompositeOperation：设置新绘制的图形与画布上已有图形的合成方式，有多种可选值，如 'source-over'（默认值，新图形覆盖在已有图形之上）、'destination-over'（已有图形覆盖在新图形之上）等。

示例6-3-1：Canvas简单示例
示例6-3-2：某网站用户访问来源
示例6-3-3：头像裁剪

4. 本地存储 4.1 Cookie

- 本地存储是一种在浏览器端存储数据的技术，允许网页在用户的浏览器中保存数据，以便在后续的页面访问中使用。常见的本地存储方式有Cookie、Web Storage（包含localStorage和sessionStorage）和IndexedDB。

- Cookie是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器下次向同一服务器再发起请求时被携带上并发送到服务器上。
- 特点
  - 数据会随 HTTP 请求一起发送到服务器端，会增加请求的数据量。
  - 存储容量有限，一般为 4KB 左右。
  - 有过期时间，可以设置为会话期（浏览器关闭即失效）或指定过期时间。

示例6-4-1：Cookie

4. 本地存储 4.2 Web Storage

- Web Storage 包含localStorage和sessionStorage，它们的使用方式基本相同，但有不同的生命周期。

- localStorage：localStorage用于长期保存数据，除非主动删除，否则数据不会过期。
- 特点
  - 存储容量一般为 5MB 左右。
  - 数据仅存储在客户端，不会随 HTTP 请求发送到服务器端。
  - 同一浏览器的同源页面（协议、域名、端口都相同）可以共享数据。

示例6-4-2：localStorage

4. 本地存储 4.2 Web Storage

- Web Storage 包含localStorage和sessionStorage，它们的使用方式基本相同，但有不同的生命周期。

- sessionStorage：sessionStorage用于临时保存同一窗口（或标签页）的数据，在关闭窗口或标签页后数据会被清除。
- 特点
  - 存储容量一般为 5MB 左右。
  - 数据仅存储在客户端，不会随 HTTP 请求发送到服务器端。
  - 数据仅在当前会话期间有效，不同窗口或标签页之间不共享数据。

示例6-4-3：sessionStorage

4. 本地存储 4.3 IndexedDB

- IndexedDB是一种在浏览器中存储大量结构化数据的数据库系统，支持事务操作和索引，可用于离线应用开发。
- 特点
  - 存储容量大，通常没有严格的限制。
  - 支持异步操作，不会阻塞主线程。
  - 可以存储各种类型的数据，包括文件、二进制数据等。

4. 本地存储 4.3 IndexedDB

- IndexedDB是一种在浏览器中存储大量结构化数据的数据库系统，支持事务操作和索引，可用于离线应用开发。
- 特点
  - 存储容量大，通常没有严格的限制。
  - 支持异步操作，不会阻塞主线程。
  - 可以存储各种类型的数据，包括文件、二进制数据等。

示例6-4-4：IndexedDB

5. 地理定位

- 地理定位功能允许网页获取用户设备的地理位置信息。这一功能基于浏览器提供的Geolocation API实现。

- 在使用地理定位功能之前，浏览器会向用户请求获取地理位置的权限。只有当用户允许后，才能获取到相关信息。

- Geolocation API 主要方法
  - getCurrentPosition()：用于获取用户的当前地理位置信息，这是一个异步操作，它接受三个参数：成功回调函数、失败回调函数和可选的配置对象。
  - watchPosition()：用于持续跟踪用户的地理位置信息，只要用户的位置发生变化，就会触发成功回调函数。它的参数和getCurrentPosition()类似。
  - clearWatch()：用于停止watchPosition()方法的跟踪，需要传入watchPosition()返回的唯一标识符。

5. 地理定位

- 获取当前地理位置
  - 首先检查浏览器是否支持Geolocation API。
  - 如果支持，调用getCurrentPosition()方法，传入成功回调函数和失败回调函数。
  - 成功回调函数中，从position对象获取纬度和经度信息，并显示在页面上。
  - 失败回调函数中，根据不同的错误代码显示相应的错误信息。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Geolocation Example</title>
</head>

<body>
    <button id="getLocationButton">获取当前位置</button>
    <p id="locationInfo"></p>

    <script>
        const getLocationButton = document.getElementById('getLocationButton');
        const locationInfo = document.getElementById('locationInfo');

        getLocationButton.addEventListener('click', () => {
            if (navigator.geolocation) {
                // 调用 getCurrentPosition 方法获取当前位置
                navigator.geolocation.getCurrentPosition(
                    // 成功回调函数
                    (position) => {
                        const latitude = position.coords.latitude;
                        const longitude = position.coords.longitude;
                        locationInfo.textContent = `纬度: ${latitude}, 经度: ${longitude}`;
                    },
                    // 失败回调函数
                    (error) => {
                        switch (error.code) {
                            case error.PERMISSION_DENIED:
                                locationInfo.textContent = "用户拒绝了地理位置请求";
                                break;
                            case error.POSITION_UNAVAILABLE:
                                locationInfo.textContent = "位置信息不可用";
                                break;
                            case error.TIMEOUT:
                                locationInfo.textContent = "请求地理位置超时";
                                break;
                            case error.UNKNOWN_ERROR:
                                locationInfo.textContent = "发生未知错误";
                                break;
                        }
                    }
                );
            } else {
                locationInfo.textContent = "你的浏览器不支持地理定位功能";
            }
        });
    </script>
</body>

</html>
```

5. 地理定位

- 持续跟踪地理位置
  - 点击 “开始跟踪位置” 按钮时，调用watchPosition()方法开始持续跟踪用户的地理位置。
  - 每次位置发生变化时，更新页面上的位置信息。
  - 点击 “停止跟踪位置” 按钮时，调用clearWatch()方法停止跟踪。

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Watch Geolocation Example</title>
</head>

<body>
    <button id="startTrackingButton">开始跟踪位置</button>
    <button id="stopTrackingButton">停止跟踪位置</button>
    <p id="trackingInfo"></p>

    <script>
        const startTrackingButton = document.getElementById('startTrackingButton');
        const stopTrackingButton = document.getElementById('stopTrackingButton');
        const trackingInfo = document.getElementById('trackingInfo');
        let watchId;

        startTrackingButton.addEventListener('click', () => {
            if (navigator.geolocation) {
                // 调用 watchPosition 方法持续跟踪位置
                watchId = navigator.geolocation.watchPosition(
                    (position) => {
                        const latitude = position.coords.latitude;
                        const longitude = position.coords.longitude;
                        trackingInfo.textContent = `当前纬度: ${latitude}, 当前经度: ${longitude}`;
                    },
                    (error) => {
                        switch (error.code) {
                            case error.PERMISSION_DENIED:
                                trackingInfo.textContent = "用户拒绝了地理位置请求";
                                break;
                            case error.POSITION_UNAVAILABLE:
                                trackingInfo.textContent = "位置信息不可用";
                                break;
                            case error.TIMEOUT:
                                trackingInfo.textContent = "请求地理位置超时";
                                break;
                            case error.UNKNOWN_ERROR:
                                trackingInfo.textContent = "发生未知错误";
                                break;
                        }
                    }
                );
            } else {
                trackingInfo.textContent = "你的浏览器不支持地理定位功能";
            }
        });

        stopTrackingButton.addEventListener('click', () => {
            if (watchId) {
                // 调用 clearWatch 方法停止跟踪
                navigator.geolocation.clearWatch(watchId);
                trackingInfo.textContent = "位置跟踪已停止";
            }
        });
    </script>
</body>

</html>
```

5. 地理定位

- 注意事项
  - 隐私问题：地理定位涉及用户的隐私信息，因此在使用时需要遵循相关的隐私政策，并在获取用户位置信息前获得用户的明确授权。
  - 兼容性：虽然大多数现代浏览器都支持Geolocation API，但在一些旧版本的浏览器中可能不支持，使用时需要进行兼容性检查。
  - 准确性：地理位置信息的准确性可能受到多种因素的影响，如设备的 GPS 功能、网络环境等。

示例6-5-1：地理位置

扩展1：跨域问题

- 跨域的概念
  - 跨域是指浏览器从一个域名的网页去请求另一个域名的资源时，由于浏览器的同源策略，会导致请求被限制。同源策略是一种重要的安全机制，它限制了一个源（协议、域名、端口三者相同）的网页与另一个源的资源进行交互。
  - 例如，以下几种情况属于跨域请求：
    - 协议不同：http://example.com 向 https://example.com 发送请求。
    - 域名不同：http://domain1.com 向 http://domain2.com 发送请求。
    - 端口不同：http://example.com:8080 向 http://example.com:3000 发送请求。

- 产生原因
  - 跨域问题的产生主要是由于浏览器的同源策略。同源策略的目的是为了防止不同源的脚本访问和操作其他源的敏感数据，保护用户信息安全。当浏览器检测到请求的源与当前页面的源不一致时，会对请求进行限制，导致请求无法正常完成。

扩展1：跨域问题

- JSONP（JSON with Padding）
  - JSONP 是一种跨域解决方案，它利用了 <script> 标签的 src 属性不受同源策略限制的特点。通过在页面中动态创建 <script> 标签，向服务器请求一个 JSON 数据，并在请求的 URL 中添加一个回调函数名作为参数。服务器收到请求后，会将 JSON 数据包装在回调函数中返回给客户端。客户端的 <script> 标签会执行这个回调函数，从而获取到服务器返回的数据。

- CORS（Cross-Origin Resource Sharing）
  - CORS 是一种现代的跨域解决方案，它是 W3C 标准，允许浏览器和服务器进行跨域通信。服务器通过设置响应头来告诉浏览器哪些源可以访问该资源。当浏览器发送跨域请求时，会自动在请求头中添加 Origin 字段，服务器根据这个字段来判断是否允许该请求。如果允许，服务器会在响应头中添加 Access-Control-Allow-Origin 等相关字段，浏览器收到响应后会根据这些字段来决定是否允许访问该资源。

- 代理服务器
  - 在同源的服务器上设置一个代理，前端将请求发送到同源的代理服务器，代理服务器再将请求转发到目标服务器，并将目标服务器的响应返回给前端。由于前端与代理服务器是同源的，所以不会受到同源策略的限制。

扩展2：ECharts

- ECharts 是一个由百度开源的数据可视化工具，它提供了直观、交互丰富、可高度个性化定制的数据可视化图表。
- 特点
  - 丰富的图表类型：支持折线图、柱状图、饼图、散点图、雷达图、地图、热力图等几十种常见和复杂的图表类型，满足不同场景下的数据可视化需求。
  - 高度可定制：你可以对图表的颜色、样式、字体、标签等各个细节进行定制，以实现独特的视觉效果，使图表与项目整体风格相匹配。
  - 强大的交互功能：支持鼠标悬停、点击、缩放、拖拽等交互操作，方便用户深入探究数据。例如，通过鼠标悬停可以显示数据的详细信息，通过缩放和拖拽可以查看数据的不同范围。
  - 跨平台兼容性：可以在 PC 端和移动端等多种平台上流畅运行，并且能自适应不同的屏幕尺寸，保证在各种设备上都能呈现出良好的视觉效果。
  - 数据驱动：采用数据驱动的设计理念，只需要将数据传入相应的配置项中，ECharts 就能自动生成对应的图表，方便数据的动态更新和展示。

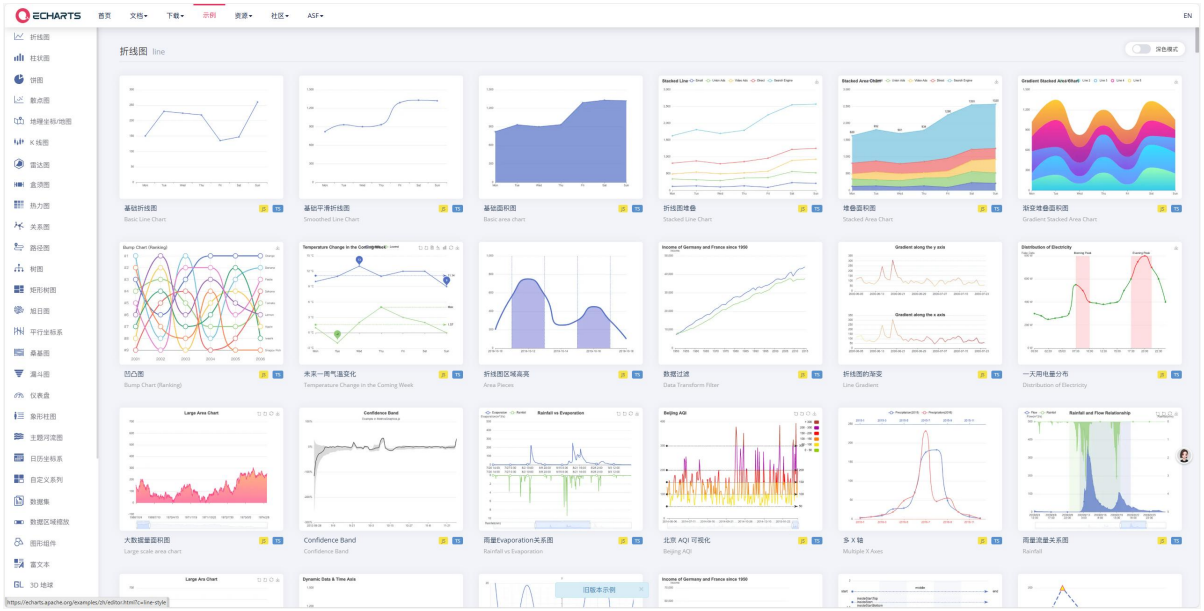
扩展2：ECharts

- 可以从 ECharts 的官方网站（https://echarts.apache.org/zh/download.html ）下载 ECharts 的 JavaScript 文件，然后在 HTML 文件中通过 <script> 标签引入。

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ECharts Example</title>
    <!-- 引入 ECharts 文件 -->
    <script src="echarts.min.js"></script>
</head>
<body>
    <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
    <div id="main" style="width: 600px;height:400px;"></div>
    <script>
        // 基于准备好的dom，初始化echarts实例
        var myChart = echarts.init(document.getElementById('main'));
        // 指定图表的配置项和数据
        var option = {
            xAxis: {
                type: 'category',
                data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
            },
            yAxis: {
                type: 'value'
            },
            series: [{
                data: [820, 932, 901, 934, 1290, 1330, 1320],
                type: 'line'
            }]
        };
        // 使用刚指定的配置项和数据显示图表。
        myChart.setOption(option);
    </script>
</body>
</html>
```

扩展2：ECharts

- 基本使用步骤
  - 准备 DOM 容器：在 HTML 文件中创建一个具有明确宽度和高度的 元素，用于显示 ECharts 图表。
  - 初始化 ECharts 实例：使用 echarts.init() 方法初始化一个 ECharts 实例，并将其绑定到之前创建的 DOM 容器上。
  - 配置图表选项：定义一个包含图表配置信息的 JavaScript 对象，包括图表类型、数据、坐标轴、标题等。
  - 显示图表：使用 setOption() 方法将配置选项应用到 ECharts 实例上，从而显示出图表。



https://echarts.apache.org/zh/index.html

重点回顾

- AJAX
  - 概念与原理：AJAX 是异步的 JavaScript 和 XML，能在不重新加载整个网页的情况下与服务器异步通信并更新部分网页，核心是利用 XMLHttpRequest 对象（或 fetch API）发送 HTTP 请求，进行数据交换。
  - 实现方式：使用 XMLHttpRequest 对象需创建对象、打开连接、监听状态变化、处理响应和发送请求；fetch API 更简洁，通过链式调用处理请求和响应。
  - 优缺点：优点有无刷新更新页面、异步通信、减少数据传输量；缺点有浏览器兼容性问题、安全风险、影响 SEO。
  - 应用案例：用于实时搜索建议、动态加载内容、表单异步提交、实时数据更新、在线聊天应用等场景。
- 文件操作
  - 文件对象获取与属性：通过 HTML 的<input type="file">元素获取文件对象，文件对象有 name、size、type、lastModified 等属性。
  - FileReader 工具：用于异步读取本地文件内容，有 readAsText、readAsDataURL、readAsArrayBuffer 等方法，还有 onloadstart、onprogress 等事件。
  - Blob 对象：表示不可变的原始数据，可创建、获取属性和进行切片操作。
  - 文件上传与下载：普通文件上传可使用 XMLHttpRequest 或 fetch；大文件上传采用分块上传；文件下载可通过 Blob 对象和 URL.createObjectURL 实现。

重点回顾

- 绘图（Canvas）
  - 基本概念与步骤：Canvas 提供绘图区域，需获取 DOM 对象、调用 getContext () 方法得到绘图上下文对象，然后调用其方法绘图。
  - 绘图方法与属性：包括绘制矩形、路径、文字、图片等方法，以及颜色、样式、文本、阴影、全局合成等相关属性。
  - 坐标变换：可进行保存和恢复画布状态，以及平移、旋转、缩放等坐标变换操作。
- 本地存储
  - Cookie：服务器发送到浏览器保存的数据，会随 HTTP 请求发送，存储容量有限，有过期时间。
  - Web Storage：包含 localStorage（长期保存数据）和 sessionStorage（临时保存同一窗口数据），存储容量约 5MB，仅存储在客户端。
  - IndexedDB：用于存储大量结构化数据，支持事务操作和索引，存储容量大，支持异步操作。
- 地理定位
  - API 方法：基于 Geolocation API，主要方法有 getCurrentPosition () 获取当前位置、watchPosition () 持续跟踪位置、clearWatch () 停止跟踪。
  - 使用流程与注意事项：获取当前位置需检查浏览器支持、调用方法并处理回调；持续跟踪位置需操作按钮控制；使用时要注意隐私、兼容性和准确性问题。

重点回顾

- 扩展知识
  - 跨域问题：跨域因浏览器同源策略限制产生，可通过 JSONP（利用<script>标签）、CORS（服务器设置响应头）、代理服务器（同源服务器转发请求）解决。
  - ECharts：百度开源的数据可视化工具，有丰富图表类型、高度可定制、强大交互功能、跨平台兼容、数据驱动等特点，使用步骤包括准备 DOM 容器、初始化实例、配置选项、显示图表。

信创智能医疗系统研发课程体系

河南中医药大学信息技术学院（智能医疗行业学院）

河南中医药大学信息技术学院（智能医疗行业学院）智能医疗教研室

河南中医药大学医疗健康信息工程技术研究所