JS写题常见API

基本数学运算方法

- `Math.abs()`
  - 功能：返回一个数的绝对值。
  - 示例：`Math.abs(-5)` 会返回 `5`，`Math.abs(5)` 也返回 `5`。它用于确保得到的结果是非负的，在处理数值比较或者计算距离等场景中很有用。
- `Math.ceil()`、`Math.floor()` 和 `Math.round()`
  - `Math.ceil()`：功能是向上取整。例如，`Math.ceil(4.1)` 返回 `5`，它总是将小数部分向上舍入到最近的整数。
  - `Math.floor()`：与 `Math.ceil()` 相反，是向下取整。例如，`Math.floor(4.9)` 返回 `4`，它会将小数部分直接舍去，得到不大于原数的最大整数。
  - `Math.round()`：是四舍五入取整。例如，`Math.round(4.4)` 返回 `4`，`Math.round(4.5)` 返回 `5`。
- `Math.pow()` 和 `Math.sqrt()`
  - `Math.pow()`：用于计算一个数的幂次方。例如，`Math.pow(2, 3)` 表示计算 `2` 的 `3` 次方，结果是 `8`。它接受两个参数，第一个是底数，第二个是指数。
  - `Math.sqrt()`：用于计算一个数的平方根。例如，`Math.sqrt(9)` 返回 `3`，用于获取一个非负数的平方根。

1. 三角函数方法
   - `Math.sin()`、`Math.cos()` 和 `Math.tan()`
     - 这些方法用于计算三角函数值。`Math.sin()` 计算正弦值，`Math.cos()` 计算余弦值，`Math.tan()` 计算正切值。角度是以弧度为单位的。例如，`Math.sin(Math.PI / 2)`（因为 `PI / 2` 弧度对应的角度是 `90` 度）返回 `1`。
2. 随机数相关方法
   - `Math.random()`
     - 功能：产生一个介于 `0`（包含）和 `1`（不包含）之间的随机数。例如，每次调用 `Math.random()` 可能会得到一个不同的小数，如 `0.123`、`0.789` 等。这个方法在模拟随机事件、生成随机验证码等场景中经常使用。
     - 可以通过一些技巧来生成指定范围内的随机数。例如，要生成一个介于 `1` 和 `10` 之间（包含 `1` 和 `10`）的整数随机数，可以使用 `Math.floor(Math.random() * 10 + 1)`。

3. 其他实用方法
    - `Math.max()` 和 `Math.min()`
        - `Math.max()`：用于返回一组数中的最大值。例如，`Math.max(1, 3, -1)` 返回 `3`。它可以接受多个参数，比较这些参数并返回最大值。
        - `Math.min()`：与 `Math.max()` 相反，用于返回一组数中的最小值。例如，`Math.min(1, 3, -1)` 返回 `-1`。
    - `Math.exp()` 和 `Math.log()`
        - `Math.exp()`：用于计算自然常数 `e` 的指定幂次方。例如，`Math.exp(1)` 约等于 `2.71828`（因为 `e` 约等于 `2.71828`）。
        - `Math.log()`：用于计算自然对数。例如，`Math.log(Math.exp(1))` 返回 `1`。如果参数小于等于 `0`，则返回 `NaN`。
    - **toFixed 四舍五入**

数组（Array）相关 API

1. 数组的创建与转换
    - `Array()` constructor（构造函数）
        - 功能：用于创建一个新的数组。可以传入不同类型的参数来初始化数组。例如，`let arr1 = new Array();` 创建一个空数组，`let arr2 = new Array(3);` 创建一个长度为 3 的数组（元素初始值为 `undefined`），`let arr3 = new Array(1, 2, 3);` 创建一个包含 1、2、3 这三个元素的数组。
        - 场景：在需要动态创建数组，特别是根据某些条件确定数组长度或初始元素时使用。
    - `from()`
        - 功能：将类数组对象或可迭代对象转换为真正的数组。例如，`let arrayLike = {0: 'a', 1: 'b', length: 2}; let arr = Array.from(arrayLike);` 会将 `arrayLike` 转换为 `["a", "b"]`。它还可以接受第二个参数，用于对每个元素进行映射操作，如 `let str = "abc"; let arr = Array.from(str, (x) => x.toUpperCase());` 会得到 `["A", "B", "C"]`。
        - 场景：处理 DOM 节点列表（类数组对象），或者将其他可迭代的数据结构（如 `Set`、`Map`）转换为数组时很有用。
    - `of()`
        - 功能：用于创建一个包含任意数量参数的数组。例如，`let arr = Array.of(1, 2, 3);` 创建一个包含 1、2、3 这三个元素的数组。与 `Array()` 构造函数不同的是，`Array.of()` 不会因为传入一个数字参数而创建一个指定长度的空数组。

- 场景：在需要明确创建一个包含特定元素的数组时使用，避免了 `Array()` 构造函数可能出现的歧义。

2. 数组元素的访问与遍历
   - `length`
     - 功能：返回数组中元素的数量。例如，`let arr = [1, 2, 3]; console.log(arr.length);` 返回 3。它也可以用于设置数组的长度，如 `arr.length = 4;` 会在数组末尾添加一个 `undefined` 元素（如果新长度大于原长度），或者截断数组（如果新长度小于原长度）。
     - 场景：用于循环遍历数组、判断数组是否为空、调整数组大小等多种场景。
   - `[]`（方括号）访问
     - 功能：用于访问数组中指定索引位置的元素。索引从 0 开始计数。例如，`let arr = [1, 2, 3]; console.log(arr[1]);` 返回 2。也可以通过 `[]` 来修改数组元素的值，如 `arr[0] = 4;` 会将数组的第一个元素修改为 4。
     - 场景：在需要按位置获取或修改数组元素时使用，是数组操作中最基本的方法之一。
   - `forEach()`
     - 功能：对数组中的每个元素执行一次提供的函数。该函数接受三个参数：当前元素、当前元素的索引和整个数组。例如，`let arr = [1, 2, 3]; arr.forEach((element, index, array) => { console.log(element, index, array); });` 会依次打印出每个元素及其索引和整个数组。
     - 场景：在需要对数组的每个元素进行相同的操作（如打印、修改、调用方法等）时使用，是一种简单直接的遍历方式。
   - `entries()`、`keys()` 和 `values()`
     - `entries()`：返回一个新的迭代器对象，该对象包含数组的索引 - 元素对。例如，`let arr = [1, 2, 3]; for (let [index, element] of arr.entries()) { console.log(index, element); }` 会依次打印出每个元素的索引和元素本身。
     - `keys()`：返回一个新的迭代器对象，包含数组的索引。例如，`for (let key of arr.keys()) { console.log(key); }` 会打印出数组的所有索引。
     - `values()`：返回一个新的迭代器对象，包含数组的元素。例如，`for (let value of arr.values()) { console.log(value); }` 会打印出数组的所有元素。
     - 场景：在需要同时获取数组的索引和元素，或者分别获取索引或元素时使用，特别是在与 `for...of` 循环结合时，可以更灵活地遍历数组。

3. 数组元素的添加与删除
   - `push()` 和 `pop()`

- `push()`：用于在数组的末尾添加一个或多个元素，并返回新的数组长度。例如，`let arr = [1, 2]; let newLength = arr.push(3);` 此时 `arr` 变为 `[1, 2, 3]`，`newLength` 为 3。
- `pop()`：用于删除数组的最后一个元素，并返回被删除的元素。例如，`let arr = [1, 2, 3]; let lastElement = arr.pop();` 此时 `arr` 变为 `[1, 2]`，`lastElement` 为 3。
- 场景：`push()` 常用于向数组中动态添加元素，`pop()` 常用于模拟栈数据结构（后进先出），如在函数调用栈的实现或者撤销操作的模拟中使用。

- `unshift()` 和 `shift()`
  - `unshift()`：用于在数组的开头添加一个或多个元素，并返回新的数组长度。例如，`let arr = [2, 3]; let newLength = arr.unshift(1);` 此时 `arr` 变为 `[1, 2, 3]`，`newLength` 为 3。
  - `shift()`：用于删除数组的第一个元素，并返回被删除的元素。例如，`let arr = [1, 2, 3]; let firstElement = arr.shift();` 此时 `arr` 变为 `[2, 3]`，`firstElement` 为 1。
  - 场景：`unshift()` 和 `shift()` 可以用于在数组头部进行元素操作，`shift()` 也常用于模拟队列数据结构（先进先出）的出队操作。

- `splice()`
  - 功能：用于删除、插入和替换数组元素。它接受至少一个参数，第一个参数是起始索引。例如，`let arr = [1, 2, 3, 4, 5]; arr.splice(1, 2);` 会从索引 1 开始删除 2 个元素，此时 `arr` 变为 `[1, 4, 5]`。如果提供更多的参数，可以在删除的位置插入新的元素，如 `let arr = [1, 2, 3, 4, 5]; arr.splice(1, 0, 'a', 'b');` 会在索引 1 处插入 `'a'` 和 `'b'`，此时 `arr` 变为 `[1, 'a', 'b', 2, 3, 4, 5]`。
  - 场景：在需要对数组中间部分进行复杂的元素操作（如删除一段元素后插入新元素）时使用，是一个功能强大的数组修改方法。

4. 数组元素的筛选与查找
   - `filter()`
     - 功能：用于过滤数组中的元素，返回一个新的数组，其中包含满足条件的元素。例如，`let arr = [1, 2, 3, 4, 5]; let newArr = arr.filter((x) => x > 3);` 此时 `newArr` 为 `[4, 5]`。
     - 场景：在数据筛选、过滤不符合条件的元素等场景中使用，如从一组用户数据中筛选出年龄大于某个值的用户。
   - `find()` 和 `findIndex()`

- `find()`：返回数组中满足条件的第一个元素。如果没有满足条件的元素，则返回`undefined`。例如，`let arr = [1, 2, 3, 4, 5]; let foundElement = arr.find((x) => x > 3);`此时`foundElement`为4。
- `findIndex()`：返回数组中满足条件的第一个元素的索引。如果没有满足条件的元素，则返回-1。例如，`let arr = [1, 2, 3, 4, 5]; let foundIndex = arr.findIndex((x) => x > 3);`此时`foundIndex`为3。
- 场景：在查找满足特定条件的单个元素或其索引时使用，比如在一个菜单数组中查找名称为某个值的菜单项及其位置。

◦ `includes()`

- 功能：用于判断数组是否包含指定的元素，返回一个布尔值。例如，`let arr = [1, 2, 3]; console.log(arr.includes(2));`返回`true`。
- 场景：在检查元素是否存在于数组中时使用，比如在权限检查中判断用户是否具有某个权限（以权限码数组来表示用户权限）。

5. 数组元素的排序与转换

◦ `sort()`

- 功能：用于对数组元素进行排序。默认情况下，它会将数组元素转换为字符串，然后按照字典序进行排序。例如，`let arr = [3, 1, 2]; arr.sort();`此时`arr`变为`[1, 2, 3]`。也可以传入一个比较函数来指定排序规则，如`let arr = [3, 1, 2]; arr.sort((a, b) => a - b);`按照数字大小升序排序。
- 场景：在需要对数组元素进行排序的场景中使用，如对成绩数组进行排序、对文件列表按名称排序等。

◦ `reverse()`

- 功能：用于反转数组中元素的顺序。例如，`let arr = [1, 2, 3]; arr.reverse();`此时`arr`变为`[3, 2, 1]`。
- 场景：在需要将数组元素顺序颠倒的场景中使用，比如将一个栈中的元素顺序反转，或者将一个列表的显示顺序反转。

◦ `map()`

- 功能：对数组中的每个元素进行操作，并返回一个新的数组，新数组的元素是原数组元素经过操作后的结果。例如，`let arr = [1, 2, 3]; let newArr = arr.map((x) => x * 2);`此时`newArr`为`[2, 4, 6]`。
- 场景：在需要对数组元素进行批量转换或计算时使用，如将一个价格数组中的所有价格都乘以一个折扣系数。

◦ `reduce()`和`reduceRight()`

- `reduce()`：用于将数组中的元素累积成一个值。它接受一个回调函数和一个初始值（可选）。例如，`let arr = [1, 2, 3]; let sum = arr.reduce((acc, cur) => acc + cur, 0);` 这里是计算数组元素的总和，`sum` 为 6。
- `reduceRight()`：与 `reduce()` 类似，但从数组的末尾开始累积。例如，在某些需要从右向左计算的场景下使用，如计算后缀表达式的值。
- 场景：在计算数组元素的总和、乘积、拼接字符串等累积操作，或者在将数组转换为一个单一的值（如对象）的场景中使用。

6. 数组的合并与复制
   - `concat()`
     - 功能：用于合并两个或多个数组，并返回一个新的数组。例如，`let arr1 = [1, 2]; let arr2 = [3, 4]; let newArr = arr1.concat(arr2);` 此时 `newArr` 为 `[1, 2, 3, 4]`。
     - 场景：在需要将多个数组组合在一起的场景中使用，如合并两个数据集。
   - `slice()`
     - 功能：用于从数组中提取一部分元素，返回一个新的数组。它接受两个参数，第一个参数是起始索引（包含），第二个参数是结束索引（不包含）。例如，`let arr = [1, 2, 3, 4, 5]; let newArr = arr.slice(1, 3);` 此时 `newArr` 为 `[2, 3]`。如果省略第二个参数，则提取从起始索引到数组末尾的所有元素。
     - 场景：在需要复制数组的一部分元素，或者提取子数组的场景中使用，如获取列表中的某一页数据（假设数组存储列表数据）。
7. 数组的其他方法
   - `some()` 和 `every()`
     - `some()`：用于检查数组中是否至少有一个元素满足给定的条件，返回一个布尔值。例如，`let arr = [1, 2, 3]; console.log(arr.some((x) => x > 2));` 返回 `true`。
     - `every()`：用于检查数组中所有元素是否都满足给定的条件，返回一个布尔值。例如，`let arr = [1, 2, 3]; console.log(arr.every((x) => x > 0));` 返回 `true`。
     - 场景：在验证数组中的元素是否符合某种条件（如是否全部有效、是否至少有一个符合要求）的场景中使用，比如检查一组表单输入是否全部合法，或者是否至少有一个输入符合搜索条件。
   - `flat()` 和 `flatMap()`

- `flat()`：用于将嵌套的数组“扁平化”，即将多层嵌套的数组转换为一层数组。例如，`let arr = [1, [2, 3], [4, [5]]]; let flattenedArr = arr.flat();` 此时 `flattenedArr` 为 `[1, 2, 3, 4, [5]]`，可以传入一个深度参数来指定扁平化的深度，如 `arr.flat(2)` 会将数组完全扁平化。
- `flatMap()`：先对数组中的每个元素执行一个映射函数，然后将结果扁平化。例如，`let arr = [1, 2, 3]; let newArr = arr.flatMap((x) => [x, x * 2]);` 此时 `newArr` 为 `[1, 2, 2, 4, 3, 6]`。
- 场景：在处理嵌套数组结构，如树形数据结构扁平化、将一组数据转换并合并的场景中使用。

字符串（String）相关 API

1. 字符访问与遍历
   - `charAt()`
     - 功能：返回指定位置的字符。例如，`var str = "hello"; str.charAt(1);` 返回 `"e"`。位置从 0 开始计数。
     - 场景：用于按位置获取字符，如在遍历字符串查找特定字符时。
   - `charCodeAt()` 和 `codePointAt()`
     - `charCodeAt()`：返回指定位置字符的 UTF - 16 代码单元值。例如，`"A".charCodeAt(0)` 返回 65。
     - `codePointAt()`：返回指定位置字符（Unicode 代码点）的值，对于处理包含两个代码单元的字符（如表情符号）很有用。
     - 场景：在字符编码转换、处理特殊字符等场景下使用。
   - `[]`（方括号）访问
     - 功能：和 `charAt()` 类似，也可以访问指定位置的字符。例如，`var str = "world"; var char = str[2];` 返回 `"r"`。
     - 场景：使用起来更简洁，特别是在需要动态指定位置时更方便。
2. 字符串长度与位置
   - `length`
     - 功能：返回字符串的长度。例如，`var str = "javascript"; console.log(str.length);` 返回 10。
     - 场景：用于循环遍历字符串、判断字符串是否为空等多种场景。
   - `indexOf()` 和 `lastIndexOf()`
     - `indexOf()`：返回指定子串在字符串中第一次出现的位置，从 0 开始计数。若未找到则返回 - 1。例如，`"hello world".indexOf("world")` 返回 6。

- `lastIndexOf()`：返回指定子串在字符串中最后一次出现的位置。例如，`"abab".lastIndexOf("ab")` 返回 2。
- 场景：用于查找子串位置，如在文本搜索、替换操作前确定位置。

3. 字符串提取与分割
   - `substring()`、`substr()` 和 `slice()`
     - `substring()`：提取字符串中介于两个指定索引之间的子串（左闭右开区间）。例如，`"abcdef".substring(1, 4)` 返回 `"bcd"`。
     - `substr()`：提取从指定位置开始的指定长度的子串。例如，`"abcdef".substr(1, 3)` 返回 `"bcd"`。
     - `slice()`：和 `substring()` 类似，但处理负数索引更灵活。例如，`"abcdef".slice(1, -1)` 返回 `"bcde"`。
     - 场景：用于提取部分字符串，如在数据格式化、文本裁剪等场景。
   - `split()`
     - 功能：将字符串分割为子串数组，以指定的分隔符为界。例如，`"a,b,c".split(",")` 返回 `["a", "b", "c"]`。
     - 场景：用于解析以特定字符分隔的数据，如 CSV 文件内容的初步处理。
4. 字符串拼接与组合
   - `concat()`
     - 功能：用于连接两个或多个字符串。例如，`"hello".concat(" world")` 返回 `"hello world"`。
     - 场景：字符串拼接场景，但在实际中 `+` 运算符使用更普遍。
   - `join()`
     - 功能：将数组中的所有元素连接成一个字符串，元素之间用指定的分隔符。例如，`["a", "b", "c"].join("-")` 返回 `"a - b - c"`。
     - 场景：将数组元素组合成特定格式的字符串，如生成 URL 参数。
5. 字符串大小写转换
   - `toLowerCase()` 和 `toUpperCase()`
     - 功能：分别将字符串中的所有字符转换为小写或大写。例如，`"Hello".toLowerCase()` 返回 `"hello"`。
     - 场景：在不区分大小写的文本比较、数据规范化等场景中使用。
   - `toLocaleLowerCase()` 和 `toLocaleUpperCase()`
     - 功能：类似于前面两种方法，但会根据特定的本地环境（locale）进行大小写转换。

- 场景：在处理多语言环境下的文本，遵循当地语言规则进行大小写转换。

6. 字符串比较与匹配
   - `localeCompare()`
     - 功能：比较两个字符串，返回一个数字表示比较结果。例如，`"apple".localeCompare("banana")` 返回小于 0 的数。
     - 场景：用于排序字符串数组，按照特定语言的字典序排列。
   - `match()`、`search()` 和 `replace()`
     - `match()`：用于在字符串中查找匹配指定正则表达式或子串的内容，返回匹配结果数组。例如，`"abc123".match(/\d/)` 返回 `["1"]`。

```
var str = "hello world";
var result = str.match("world");
console.log(result);
// 输出 ["world"]，因为在 "hello world" 中找到了 "world" 这个子串，以数组形式返回找到的内容
```

     - `search()`：在字符串中查找指定子串或正则表达式第一次出现的位置，返回位置索引。例如，`"abc123".search(/\d/)` 返回 3。
     - `replace()`：用于替换字符串中的指定子串或匹配正则表达式的内容。例如，`"abc123".replace(/\d/, "x")` 返回 `"abcx23"`。
     - 场景：文本搜索、替换、数据清洗等场景，特别是涉及正则表达式的复杂文本处理。
7. 字符串填充与重复
   - `padStart()` 和 `padEnd()`
     - 功能：在字符串的开头或结尾填充指定的字符，直到达到指定的长度。例如，`"123".padStart(5, "0")` 返回 `"00123"`。
     - 场景：格式化字符串长度，如在生成固定格式的编号、代码等场景。
   - `repeat()`
     - 功能：返回一个新字符串，表示将原字符串重复指定次数。例如，`"abc".repeat(2)` 返回 `"abcabc"`。
     - 场景：用于生成有规律的重复字符串，如在图案绘制、数据模拟等场景。
8. 模板字符串（ES6 及以上）
   - `（反引号）和${}`

- 功能：使用反引号定义模板字符串，可以在其中嵌入表达式（用 `${}` 包裹）。例如，`let name = "John"; console.log(` Hello, ${name}! `);` 返回 `Hello, John!`。
- 场景：用于构建动态字符串，如在拼接 SQL 语句、生成 HTML 片段等场景，使字符串拼接更简洁直观。

- `String.raw()`
  - 功能：获取一个原始字符串模板，忽略字符串中的转义字符。例如，`console.log(String.raw` ${'\n'} `);` 输出 `\n`。
  - 场景：在处理模板字符串中的特殊字符，如反斜杠等，需要原始文本时使用。

日期（Date）相关 API

- `getFullYear()`、`getMonth()`、`getDate()` 等
  - `getFullYear()`：用于获取日期对象中的年份。例如：`let date = new Date(); let year = date.getFullYear();`，返回当前的年份。
  - `getMonth()`：用于获取日期对象中的月份，月份是从 `0` 开始计数的，即 `0` 表示一月，`1` 表示二月，以此类推。例如：`let date = new Date(); let month = date.getMonth();`，返回当前月份减 `1` 的值。
  - `getDate()`：用于获取日期对象中的日期，即一个月中的第几天。例如：`let date = new Date(); let day = date.getDate();`，返回当前日期。
- `setFullYear()`、`setMonth()`、`setDate()` 等
  - 这些方法用于设置日期对象中的年份、月份、日期等信息。例如：`let date = new Date(); date.setFullYear(2025);` 会将日期对象中的年份设置为 `2025`。

文档对象模型（DOM）相关 API（在浏览器环境中）

- `getElementById()`、`getElementsByClassName()` 和 `ge[illegible]e()`
  - `getElementById()`：用于通过元素的 `ID` 获取单个元素。例如：`let element = document.getElementById("my - element");`，如果页面中有一个 `ID` 为 `my - element` 的元素，就可以通过这个方法获取到它。
  - `getElementsByClassName()`：用于通过元素的类名获取元素集合。例如：`let elements = document.getElementsByClassName("my - class");`，会返回一个包含所有类名为 `my - class` 的元素的集合。
  - `getElementsByTagName()`：用于通过元素的标签名获取元素集合。例如：`let elements = document.getElementsByTagName("div");`，会返回一个包含所有 `div` 标签的元素集合。
- `addEventListener()`

- 用于为元素添加事件监听器。它接受三个参数，第一个参数是事件类型（如`"click"`、`"mouseover"`等），第二个参数是事件处理函数，第三个参数是一个布尔值（可选），用于指定事件是在捕获阶段还是冒泡阶段处理。例如：`let button = document.getElementById("my - button"); button.addEventListener("click", function() { console.log("Button clicked"); });`，当按钮被点击时，会在控制台输出`Button clicked`。