js对于变量作用域的规则

在 JavaScript 中，变量作用域决定了变量的可访问范围，主要有以下几种规则：

1. 全局作用域 (Global Scope)

在 JavaScript 中，如果变量在任何函数或块外定义，则它是全局变量。全局变量可以在代码的任何地方访问。

```javascript
var globalVar = "I'm a global variable"; // 全局作用域

function example() {
    console.log(globalVar); // 可以访问全局变量
}
```

2. 函数作用域 (Function Scope)

使用 `var` 关键字声明的变量具有函数作用域。它们只能在声明它们的函数内部访问，无法在函数外部访问。

```javascript
function example() {
    var localVar = "I'm a local variable"; // 函数作用域
    console.log(localVar); // 可以访问
}
console.log(localVar); // 错误，localVar 在函数外部不可访问
```

3. 块作用域 (Block Scope)

使用 `let` 和 `const` 声明的变量具有块作用域。块作用域是指在 `{}` 大括号内部（如 `if` 或 `for` 语句块）声明的变量只能在这个块内访问。

```javascript
if (true) {
    let blockVar = "I'm a block-scoped variable";
    console.log(blockVar); // 可以访问
}
console.log(blockVar); // 错误，blockVar 在块外不可访问
```

4. 作用域链 (Scope Chain)

在 JavaScript 中，嵌套的作用域会形成一个作用域链。当访问变量时，JavaScript 会从当前作用域开始查找变量，如果没找到就逐层向上查找，直到找到全局作用域。

```javascript
var globalVar = "global";

function outerFunction() {
    var outerVar = "outer";

    function innerFunction() {
        var innerVar = "inner";
        console.log(globalVar); // 访问到全局变量
        console.log(outerVar);  // 访问到外部函数变量
        console.log(innerVar);  // 访问到内部函数变量
    }

    innerFunction();
}

outerFunction();
```

5. `let` 和 `const` 的暂时性死区 (Temporal Dead Zone, TDZ)

`let` 和 `const` 声明的变量在块作用域的顶部就已经存在了，但在实际声明之前访问会导致错误，这段区域称为"暂时性死区"。在进入作用域后，必须等到变量声明之后才能使用它。

```javascript
{
    console.log(myVar); // ReferenceError: myVar is not defined
    let myVar = 10;     // 变量在此行之前处于TDZ
}
```

6. 闭包 (Closure) 与作用域

闭包是 JavaScript 中常见的概念，它允许函数“记住”并访问它的词法作用域，即使函数在当前作用域之外执行。

```javascript
function outer() {
    let outerVar = "I'm from outer function";

    function inner() {
        console.log(outerVar); // 可以访问外部函数作用域中的变量
    }

    return inner;
}

const closureFunction = outer();
closureFunction(); // 仍然可以访问 outerVar
```

7. 顶层作用域的声明提升 (Hoisting)

使用 `var` 声明的变量会“提升”到当前作用域的顶部，声明提升仅对 `var` 有效，而 `let` 和 `const` 不会提升。

```javascript
console.log(a); // undefined（变量提升）
var a = 5;

console.log(b); // ReferenceError（不会提升）
let b = 10;
```

总结来说：

- `var` 声明的变量有函数作用域，会提升；
- `let` 和 `const` 声明的变量有块作用域，不会提升