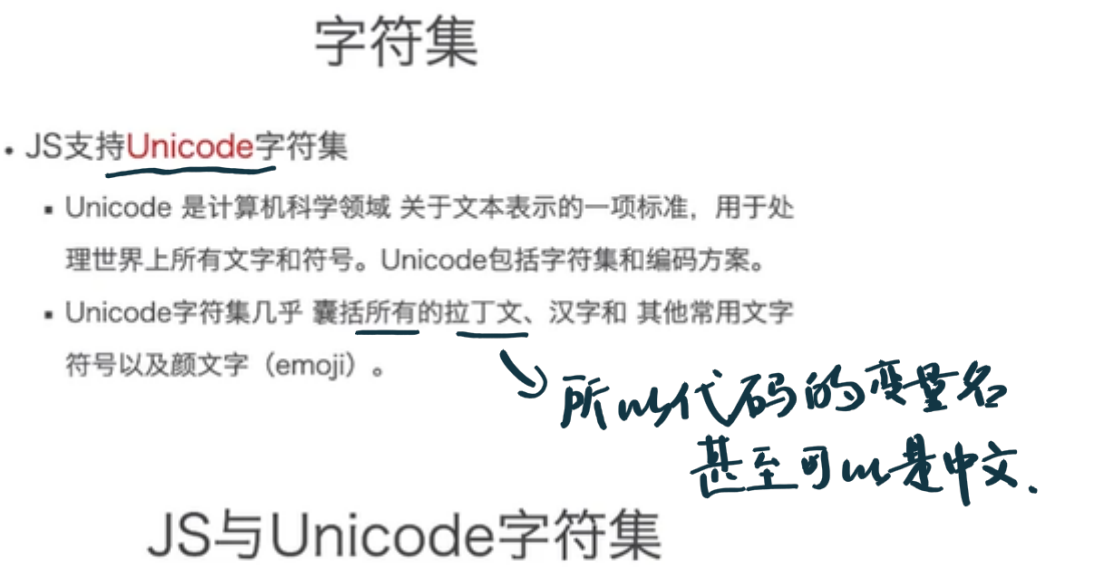
第五章：JS基础（上）

https://live.bytedance.com/9715/8787316

1. JS语言结构

2. 字符集




```
const CARD_POINTS = ['A', '2', '3', '4', '5', '6', '7',
  '8', '9', '10', 'J', 'K', 'Q'];

const CARD_SUITS = ['♥', '♠', '♣', '♦'];

function getRandomItem(list) {
  return list[Math.floor(Math.random(list.length))];
}

function getRandomCard() {
  const point = getRandomItem(CARD_POINTS);
  const suit = getRandomItem(CARD_SUITS);
}
```

扑克牌颜文字.
可以直接作为字符串

3. 符号

符号

1. 标识符
   - 由Unicode组成的符号，它可以是变量名、函数名 以及保留字
2. 字面量
   - 直接表示程序中的某些数据的符号，包括Null、Boolean、Number、String以及正则表达式（RegularExpression）等
3. 标点符
   - 表达式中连接标识名与字面量的运算符以及表示结构的花括号、小括号、中括号、点、分号、逗号、冒号等。
4. 模板
   - 一种JavaScript支持的特殊的字符串语法

符号

标识符 / 标点符 字面量.

右侧的代码中，`function, greeting, message, return` 等都是标识符，`(, ), {, }, +, = ` 等都是标点符，`'Hello ', 'world', 'everyone'` 等都是字面量。

```
function greeting(message = 'world') {
  return 'Hello ' + message;
}
const message = greeting('everyone');
console.log(message);
```

语句和语句块

- 语句由符号组成
- 分号代表语句的终止
- 一个或多个语句可以组成语句块
- 语句以花括号标记起始和结束

```
function sum(n) {
  let ret = 0;
  let i = 1;
  while(i <= n) {
    ret += i;
    i++;
  }
  return ret;
}
console.log(sum(10));
```

4. 空白符

空白符

- 空白符指Token之间可以插入的所有字符
- 空白符包括空格、换行和制表符

换行

通常符号与符号之间 也能插入一个或多个换行符，
但有一些特殊情况不允许换行：

- ❌ return和返回值之间
- ❌ break/continue和label名之间
- ❌ 变量和`++`、`--`后缀运算符之间
- ❌ throw 和异常对象之间
- ❌ 箭头函数的参数列表和箭头`=>`之间
- ❌ yield和迭代值之间
- ❌ async和异步函数声明、函数表达式、方法名之间

缩进

缩进让代码看起来更加整齐，便于阅读和理解
缩进一般用Tab或若干个空格
在一个项目中，缩进规则应当统一

↳指要么都Tab
要么都空格

缩进 → `let ret = 0;`
缩进 → `while(i <= n) {`

```
function sum(n) {
  let ret = 0;
  let i = 1;
  while(i <= n) {
    ret += i;
    i++;
  }
  return ret;
}
console.log(sum(10));
```

注释

JavaScript采用与C和Java语言一致的注释格式，分别用`//`表示单行注释，用`/*...*/`表示多行注释。

```
/**
 * 求1到n的和
 */
function sum(n) {
  let ret = 0;
  let i = 1;
  while(i <= n) {
    ret += i;
    i++;
  }
  return ret;
}

console.log(sum(10)); // 计算 1+2 ... +10
```

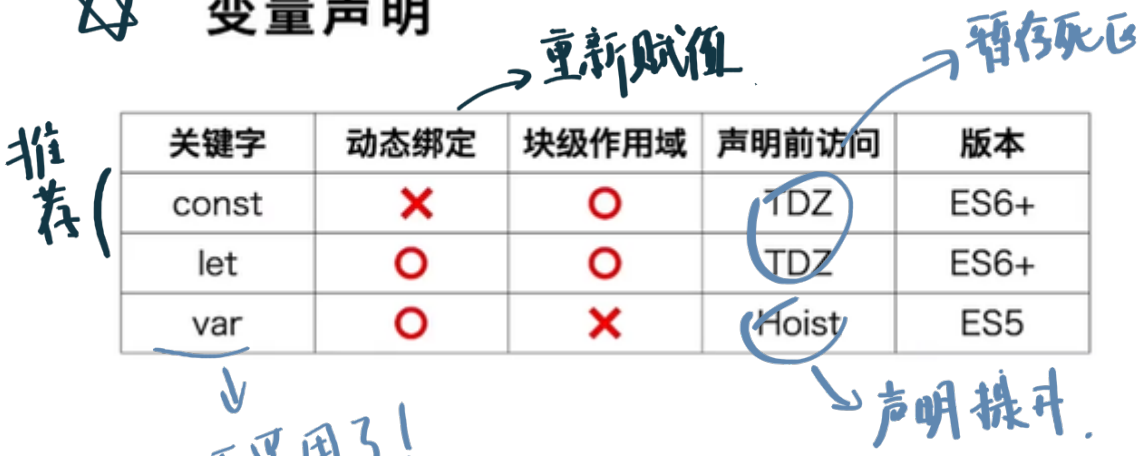
5. 作用域

作用域

- 语句块：块级作用域
- 函数：函数作用域
- 模块：模块作用域（ES Modules）

```
// 块级作用域
let i = 1;
{
  let i = 2;
  console.log(i); // 2
}
console.log(i); // 1

// 函数作用域
const bar = 'bar';
function foo() {
  const bar = 'foobar';
  console.log(bar);
}

foo();  // foobar
console.log(bar); // bar
```

变量声明

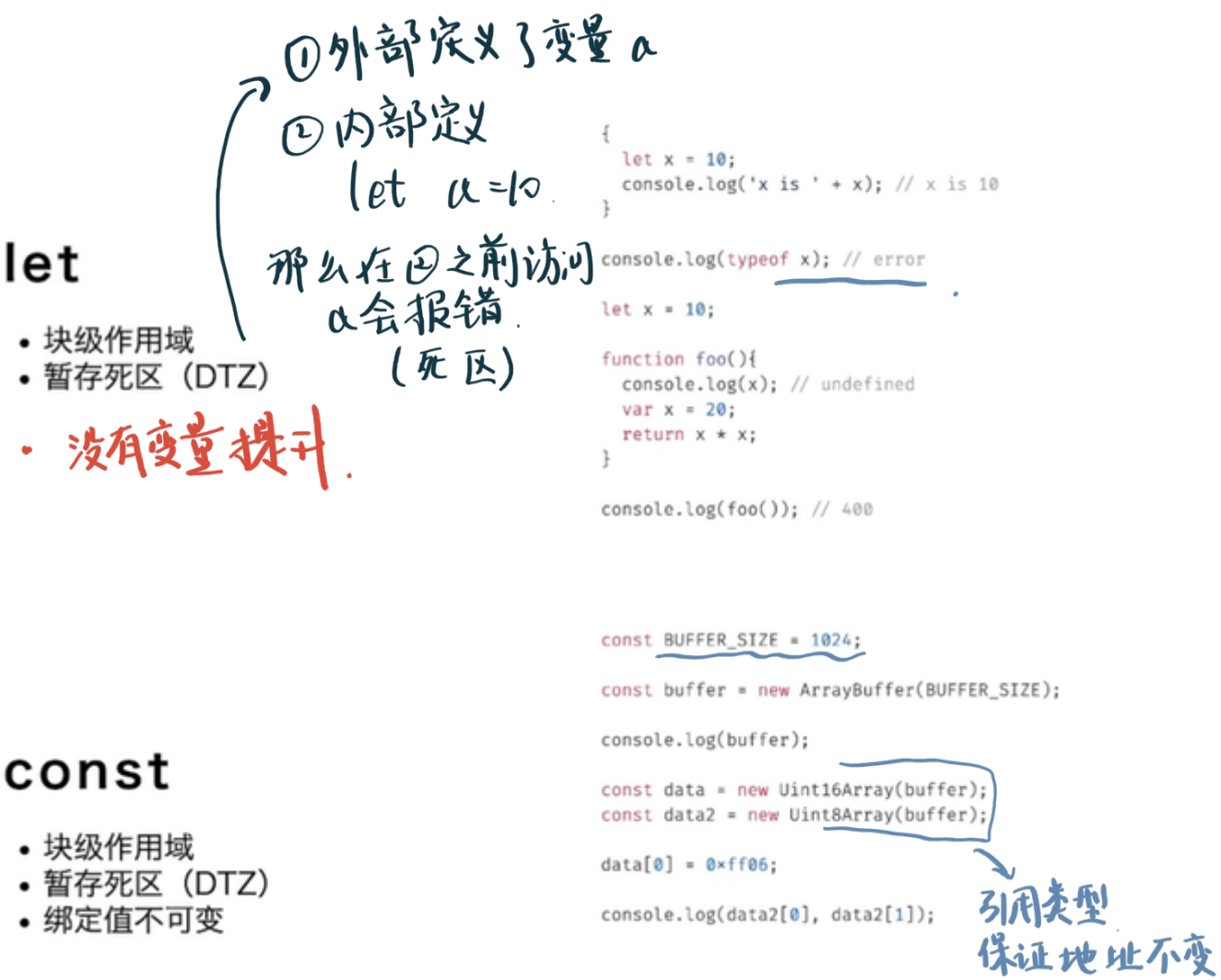


关键字	动态绑定	块级作用域	声明前访问	版本
const	✗	○	TDZ	ES6+
let	○	○	TDZ	ES6+
var	○	✗	Hoist	ES5

var

- ES5及之前的早期版本
- 无块级作用域
- 声明提升（hoist）

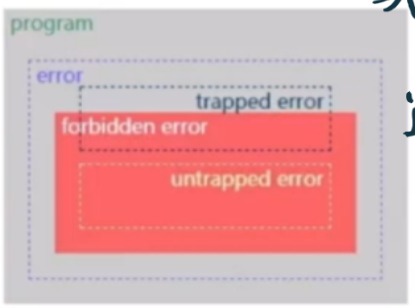
```
console.log(a === undefined); // true
var a = 10;

function foo(){
  console.log(a, i);  // undefined, undefined
  var a = 20;
  for(var i = 0; i < a; i++){
    //do sth.
  }
  console.log(a, i); // 20, 20
}

foo();
```

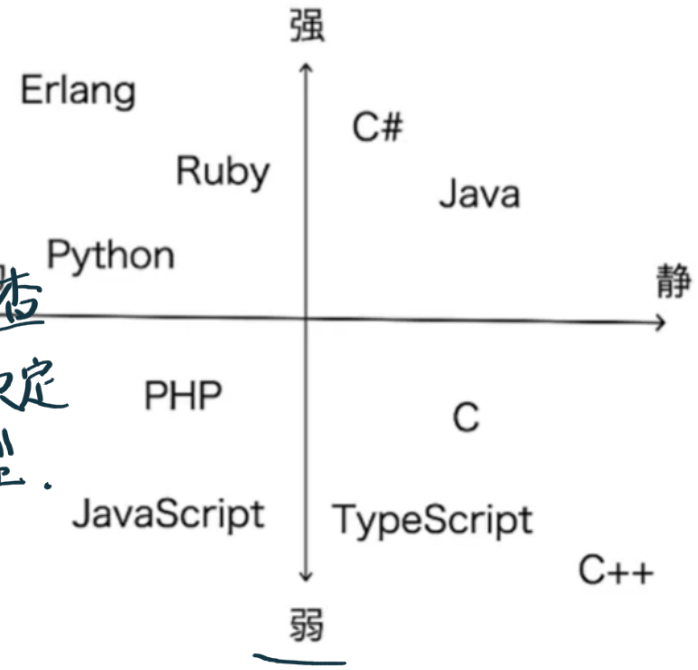


- 扩展阅读
  - https://developer.mozilla.org/zh-CN/docs/Learn/Getting_started_with_the_web/JavaScript_basics
  - https://juejin.cn/post/6844903971832758280
  - https://juejin.cn/post/6924234619621474318

2. JS类型系统

编程语言的类型系统



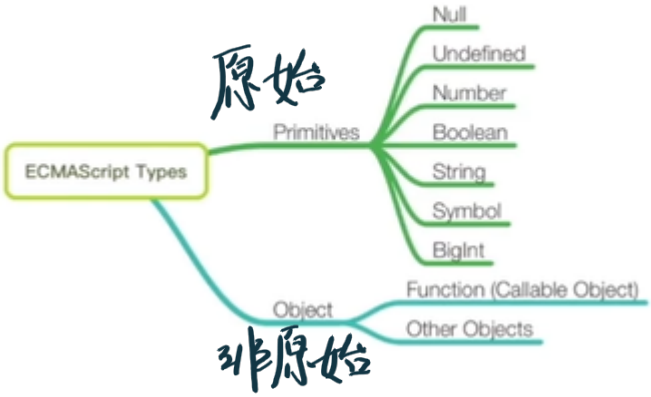


无类型检查

运行时决定变量类型.

JS数据类型

- 原始类型
- 非原始类型



原始

非原始

typeof

- 原始类型
- 非原始类型

可判断.

```
console.log(
  typeof null,        // object
  typeof undefined,   // undefined
  typeof 123,         // number
  typeof 'abc',       // string
  typeof true,        // boolean
  typeof Symbol(),    // symbol
  typeof 2n,          // bigint
  typeof Object(),    // object
);

function add(x, y) {
  return x + y;
}

console.log(typeof add); // function
```

历史遗留问题

隐式类型转换

- 字符串与数值相加时，数值被转转换为字符串
- 字符串参与非加法数学运算时，字符串被转换为数值
- 布尔值与数值进行运算时，true视为1，false视为0
- 布尔值与字符串进行相加时，布尔值被视为字符串
- 更多规则参考 ECMA-262

```
const a = 10, b = 'abc', c = 1;
console.log(a + b + c); // 10abc1

const a = 123, b = '456', c = 1
console.log(a + b - c); //123455

const a = 123, b = 'abc', c = 1;
console.log(a + b - c); // NaN

const a = true, b = false;
console.log(a + 1, b * 3); // 2 0

const a = true, b = false;
console.log(a + '', b + 'foobar');
// 'true', falsefoobar
```

隐式转换

```
console.log(100 == '1e2'); // true
console.log(true == '1', false == 0); // true true

let foo, bar = null;
console.log(foo == null,
  bar == null,
  foo == undefined,
  bar == undefined); // true true true true

console.log(true == 1, true === 1); // true false
```

用三等号时全为false

==与===

- 值用==操作符比较时，会触发隐式类型转换
- 值用===操作符比较时，不会触发隐式类型转换
- 一般原则是除了与null比较外，尽量用===
- 具体比较规则参考 ECMA-262

推荐

显式类型转换

通过调用方法`Number`、`String`、`Boolean`等
可以将值显式转换类型。

```
console.log(Number('123') == 123); // true

console.log(String(0xf) == '15'); // true

console.log(Boolean(null) == false); // true
```

- 调用原始类型的构造函数来进行显式类型转换

值类型和引用类型

- 原始类型默认是值类型
- 非原始类型默认是引用类型

（对象/数组

```
let x = 20, y = 30;

function foo(a, b){
  a++;
  b++;
  console.log([a, b]); // 21 31
}

foo(x, y);
console.log([x, y]); // 20 30

const obj = {x: 20, y: 30};
function foo2(obj){
  obj.x++;
  obj.y++;
  console.log(obj); // 21 31
}

foo2(obj);
console.log(obj); // {x: 21, y: 31}
```

- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Data_structures
- https://juejin.cn/post/6844903863430938637

3. JS原始类型(一)

Null和Undefined

Null和Undefined是JavaScript中的两种原始类型，它们分别只有一个值。

- Null的值是null
- Undefined的值是undefined
- 在非严格比较下，null == undefined

```
let foo; //变量标识符被声明而没有初始化
console.log(foo); // undefined

function bar(a, b){
    return [a, b]
}
// bar函数的第二个形参没有传入实参
console.log(bar(1)); // [1, undefined]

let sum = 0;
function addSum(num) {
  sum += num;
}
// addSum没有return值
console.log(addSum(10)); // undefined

// 访问p对象不存在的z属性
let p = {x:1, y:2};
console.log(p.z);  // undefined

let foo = null;
console.log(foo); // null
```

未初始化

无实参

无返回值

访问到不存在的属性

- `Undefined` 是初始化未赋值时的**默认值**，是**无返回值函数**的返回值
- `Null` 只有**手动指定**为 `Null` 时才会是 `Null`

0b 二
0o 八
0x 十六

Number $[-2^{53}+1, 2^{53}-1]$

Number类型表示整数和浮点数

- 是符合IEEE 754标准的64位浮点数
- 整数有二进制、八进制、十进制和十六进制表示法
- 可以用科学记数法表示
- 精确表示的整数范围从-2^53+1到2^53-1
- 常量Number.MAX_SAFE_INTEGER

```
0
7
-3
0b101  // 二进制表示的5，0b前缀表示二进制
0o777  // 八进制表示的511，0o前缀表示八进制
-0x7f   // 十六进制表示的-127，0x前缀表示十六进制
3e9    // 科学计数法表示3000000000

let n1 = 10,
  n2 = Number.MAX_SAFE_INTEGER,
  n3 = 1.2;

console.log(Number.isSafeInteger(n1),
  Number.isSafeInteger(n2),
  // n3是浮点数不是整数，所以結果也为false
  Number.isSafeInteger(n3),
  Number.isSafeInteger(n2 + 1));

//true, true, false, false
```

- `Number.isSafeInteger` 来判断某数字是否在可精确表示的**整数范围**内

浮点数

浮点数可以表示小数

- 规范规定浮点数的整数部分如果是0，0可以省略。
- 浮点数也可以使用科学计数法。
- 最大浮点数 Number.MAX_VALUE
- 最小浮点数 Number.MIN_VALUE
- 浮点数精度 Number.EPSILON
- 无穷大数 Infinity

```
.3 //相当于0.3
3.14159265
6.62e-34

Number.MAX_VALUE // 1.7976931348623157e+308
Number.MIN_VALUE // 5e-324
Number.EPSILON // 2.220446049250313e-16

let n1 = Number.MIN_VALUE,
    n2 = 1,
    n3 = n2 + n1,
    n4 = n2 + Number.EPSILON;

console.log(n1 > 0, n2 < n3, n2 < n4);
//true, false, true

console.log(Number.MAX_VALUE * 2,
  -Number.MAX_VALUE * 2,
  1 / 0,
  -1 / 0);

// Infinity, -Infinity, Infinity, -Infinity
```

↳最大值×2
除以0
Infinity

- js允许除以0，会出现 `infinity`

运算精度问题

浮点数运算存在精度问题

- 不可用相等比较浮点数
- 不可用相等比较浮点数
- 不可用相等比较浮点数

```
console.log(0.1 + 0.2); // 0.30000000000000004

console.log(0.1 + 0.2 == 0.3); // false

function floatEqual(a, b) {
  return Math.abs(a - b) < Number.EPSILON;
}

console.log(floatEqual(0.1 + 0.2, 0.3));
// true
```

要用 floatEqual(,)

- `0.1 + 0.2 -> 0.30000000000000004` 由于IEEE的规范导致的误差
- 所以需要判断相等时，需要判断两者之间的差距是否小于精度

NaN

符号NaN表示Not-a-Number。在计算的过程中，遇到无法表示为数值的情况，计算结果就会是`NaN`。

- 如果两个数值是NaN，它们的比较结果是不等的
- Number.isNaN 判断
- 用 Object.is 比较

```
const n1 = Math.sqrt(-1), // 负数开平方
    // 与一个不能转为数值的字符串进行运算
  n2 = 3 * "abc";

console.log(n1, n2); // NaN, NaN

console.log(n1 == n2); // false

console.log(Number.isNaN(n1), Number.isNaN(n2));
// true, true

console.log(Object.is(n1, n2)); // true
```

- 给负数开平方/与一个不能转换为数值的字符串进行运算 **时会出现NaN**
- `NaN == NaN -> false`
- `Number.isNaN` 来判断是否时 `NaN`

+0与-0

数值0有+0和-0两种形态，这两个值如果比较的话是相等的。但是如果它们作为除数进行运算，分别会得到+Infinity和-Infinity。

同样，如果一个有限的正数除以Infinity和-Infinity分别得到+0和-0。

```
const a = 0, b = -0;

console.log(a == b, 1 / a, 1 / b);
// true Infinity -Infinity

console.log(1 / Infinity, 1 / -Infinity); // 0 -0
```

- `+0 === -0 ->true`

- 拓展阅读
- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/undefined
- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/null
- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/number

- https://juejin.cn/post/6844903831071883277
- https://juejin.cn/post/6844903829293498375 1

4. JS原始类型(二)

Boolean

Boolean类型表示逻辑真和逻辑假，它只有两个可选的值，分别是字面量true和false。

- JS的比较操作符返回布尔类型的结果
- 做布尔判断时存在隐式类型转换
- +0、-0、NaN、空串、undefined、null 转为 false

```
const TRUE = true,
  FALSE = false;

console.log(typeof TRUE, typeof FALSE);
// boolean, boolean

if(true) {
  console.log('Something should print');
}

while(false) {
  console.log('Something should not print');
}

const result = (6 * 7 == 42);
console.log(result); // true
```

String

JS使用一对单引号 ' 或一对双引号 " 来表示字符串，单引号和双引号中间不能有换行符。

- 支持特殊转义符和Unicode转义符
- 由于HTML标签属性用双引号，所以JS字符串通常推荐用单引号

```
const text = 'This is a text.';
const html = '<p class="sth">a <em>paragraph</em></p>';

console.log(text);
console.log(html);

const text2 = '我所做的馅饼\n是全天下\n最好吃的';

console.log(text2);

const text3 = 'if(a){\n\tconsole.log(b);\n}';

console.log(text3);

const text4 = '\u5b57\u8282\u524d\u7aef';

console.log(text4);
```

↳HTML用双引号

```
"This is a text."
"<p class=\"sth\">a <em>paragraph</em></p>"
"我所做的馅饼
是全天下
最好吃的"
"if(a){
  console.log(b);
}"
"字节前端"
```

- 推荐使用单引号

处理字符

字符串可以使用spread操作符展开成字符数组。可以使用codePointAt方法来获得某位字符的Unicode码位。

- Unicode码位以多字节Unicode编码表示一个字符
- String.fromCodePoint方法可以将码位还原为字符串

```
const str1 = '東南';
const arr = [ ... str1];

console.log(arr);
console.log(str1.codePointAt(0));

const str2 = String.fromCodePoint(126978, 126979);
console.log(str1+str2);
```

```
["東", "南"]
126976
"東南西北"
```

- `[...str1]` 中的 `...str` 为 `spread` 操作

类型转换

字符串可以与其他类型数据相互操作。

- +操作符触发其他类型的隐式类型转换
- Number.parseInt与Number.parseFloat
- 显式类型转换
- 对象的toString方法

```
console.log([1+2, '1'+2, '1'-2]);
// [3, "12", -1]

console.log(Number.parseInt('100abc', 2)); // 4
console.log(Number('0b100')); // 4

console.log(Number.parseFloat('12.3e10xx'));
// 123000000000

var foo = { //对象的 toString 方法
  toString(){
   return 'foo';
  }
};

console.log(foo + ' bar'); // foobar
```

- `对象 + string` 会调用 `toString` 方法来使对象变为 `string`

↳默认调用

常用操作

字符串内置常用操作方法。

- 字符串连接
- 大小写转换
- 逆序
- 截取
- 查找

```
const a = 'hello';
const b = 'WORLD';
const c = '!';

console.log(a + ' ' + b + c); //字符串连接
// hello WORLD!
console.log(a.toUpperCase() + ' ' + b.toLowerCase() + c);
// HELLO world!
console.log(a.split('').reverse().join('')); //逆序字符串
// olleh
console.log(a.slice(2,3), a.substr(2,3)); //截取子串
// l llo

const d = a + ' ' + b + c;
console.log(d.indexOf(b)); //字符串查找
// 6
console.log(d.replace(a, a.toUpperCase()));
// "HELLO WORLD!"
```

多行文本

ES6之后，JS支持以一对反引号`表示多行文本，同时也是模板字符串。

- 多行文本保留空白符
- 多行文本是模板字符串，可以解析和替换内容

```
const tpl1 = `我所做的馅饼
  是全天下
  最好吃的`;

console.log([tpl1, typeof tpl1]);

{
  let who = '月影', what = '月饼';

  const tpl2 = `${who}所做的${what}
    是全天下
    最好吃的`;

  console.log(tpl2);
}
```

```
["我所做的馅饼
  是全天下
  最好吃的", "string"]
"月影所做的月饼
    是全天下
    最好吃的"
```

- 拓展阅读
- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Boolean
- https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/string
- https://juejin.cn/post/6921614982777929736
- https://juejin.cn/post/6844903584975290381