# 学习资料

https://www.runoob.com/js/js-syntax.html

# 基本概念

1、JavaScript 是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

2、ECMA-262 是 JavaScript 标准的官方名称。

3、HTML 中的 Javascript 脚本代码必须位于 <script> 与 </script> 标签之间。Javascript 脚本代码可被放置在 HTML 页面的 <body> 和 <head> 部分中。

# js与ts的区别

## js的特点与局限

### 特点

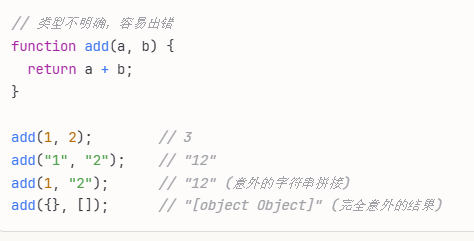
1、动态类型，编写灵活

2、无需编译，直接运行

3、学习门槛相对较低

4、生态丰富，应用广泛

### 局限性



## ts出现的背景

随着JavaScript应用越来越复杂，特别是大型企业级应用，开发者面临以下问题：

1、类型安全问题 - 运行时才发现类型错误

2、代码维护困难 - 大型项目中难以理解函数参数和返回值

3、IDE支持不足 - 缺乏准确的代码提示和重构支持

4、团队协作困难 - 缺乏接口约定，代码意图不明确

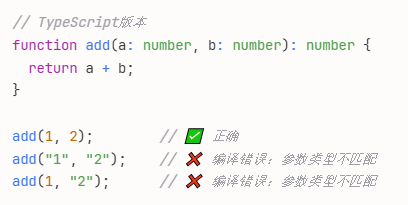
## ts的特点

1、在JavaScript基础上添加了静态类型系统

2、编译时进行类型检查

3、最终编译成纯JavaScript代码

4、完全兼容现有JavaScript代码



## 主要区别

### 1、类型系统

JavaScript (动态类型)



TypeScript (静态类型)



### 2、面向对象编程

JavaScript (ES6+)



TypeScript



### 3、接口定义

JavaScript



TypeScript



### 4、泛型支持

JavaScript

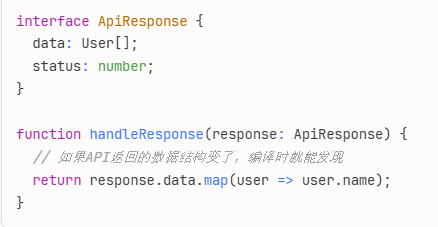


TypeScript



## ts的优势

### 1、更早发现错误



### 2、更好的IDE支持

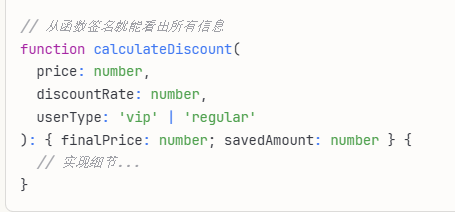
1、准确的代码补全

2、实时错误检查

3、安全的重构操作

4、准确的跳转到定义

### 3、自文档化代码



### 4、渐进式采用



## 选择建议

### 选择JavaScript的场景

1、快速原型开发

2、小型项目

3、学习阶段

4、对构建工具要求简单

### 选择TypeScript的场景

1、大型项目

2、团队协作

3、需要长期维护的代码

4、对代码质量要求高的项目

# 输出

JavaScript 可以通过不同的方式来输出数据：

1、使用 window.alert() 弹出警告框。

2、使用 document.write() 方法将内容写到 HTML 文档中。

3、使用 innerHTML 写入到 HTML 元素。

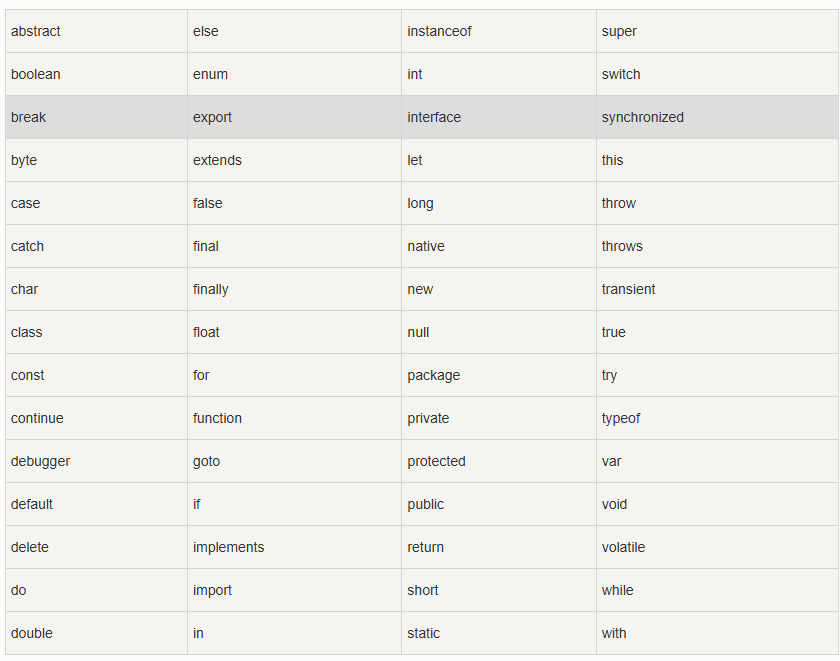
4、使用 console.log() 写入到浏览器的控制台。

# 关键字

## 基本概念

JavaScript 关键字用于标识要执行的操作。和其他任何编程语言一样，JavaScript 保留了一些关键字为自己所用。

## 常用的关键字

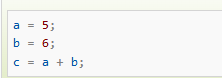


# js字母大小写

JavaScript 对大小写是敏感的。当编写 JavaScript 语句时，请留意是否关闭大小写切换键。函数 getElementById 与 getElementbyID 是不同的。同样，变量 myVariable 与 MyVariable 也是不同的。

# 分号

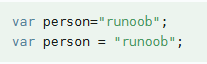
分号用于分隔 JavaScript 语句。通常我们在每条可执行的语句结尾添加分号。使用分号的另一用处是在一行中编写多条语句。



您也可能看到不带有分号的案例。在 JavaScript 中，用分号来结束语句是可选的。

# 空格

JavaScript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性。下面的两行代码是等效的：



# 对代码行进行折行

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：



# 注释

## 单行注释

单行注释以 // 开头，代码示例如下：



## 多行注释

多行注释以 /\* 开始，以 \*/ 结尾。

代码示例：



# 变量

## 基本概念

变量是用于存储信息的"容器"。在 JavaScript 中，变量用于存储数据，并可以在程序执行过程中动态更改。在 JavaScript 中，变量可以存储各种类型的数据，如数字、字符串、对象、函数等。变量名是标识符，用于引用存储在变量中的数据。

JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象。

## 命名规则

1、变量必须以字母开头

2、变量也能以 $ 和 \_ 符号开头（不过我们不推荐这么做）

3、变量名称对大小写敏感（y 和 Y 是不同的变量）

## let

let 变量名 = 值;

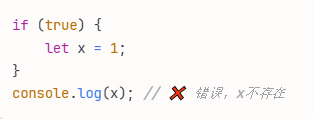
let声明的变量可以被多次赋值，例如：

let a = 100;//初始值是100

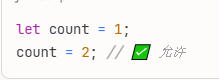
a = 200;//ok 被重新赋值为200

### 关键特点

#### 1、块级作用域



#### 2、可以重新赋值



#### 3、不允许重复声明



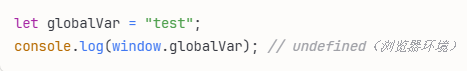
#### 4、存在暂时性死区



#### 5、可以先声明后赋值



#### 6、不会成为全局对象的属性



相比 var，let 解决了变量提升和作用域混乱的问题，是现代JavaScript推荐的变量声明方式。

## const

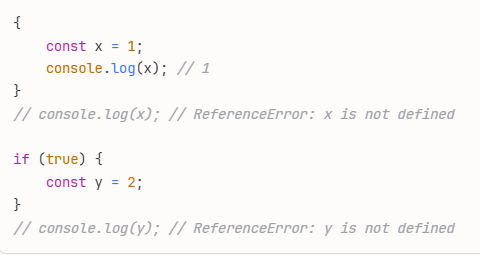
### 基本语法

const 变量名 = 值;

### 核心特性

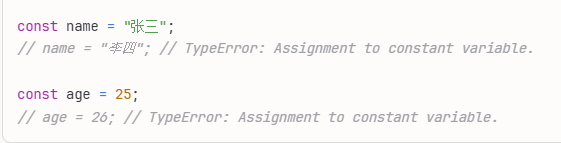
#### 1、块级作用域

const声明的变量具有块级作用域



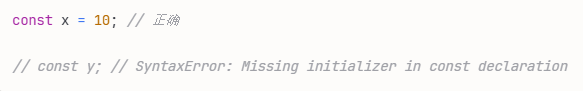
#### 2、不可重新赋值

const声明的变量不能重新赋值。



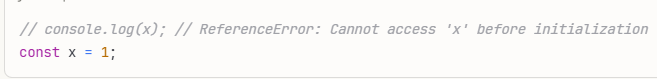
#### 3、必须初始化

const声明时必须立即初始化



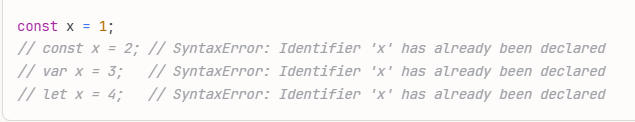
#### 4、不存在变量提升

const不会被提升到作用域顶部



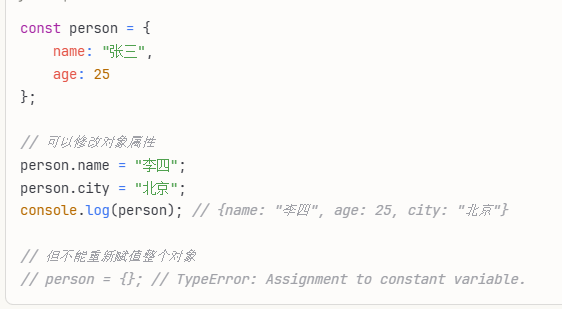
#### 5、不可重复声明

在同一作用域内不能重复声明。

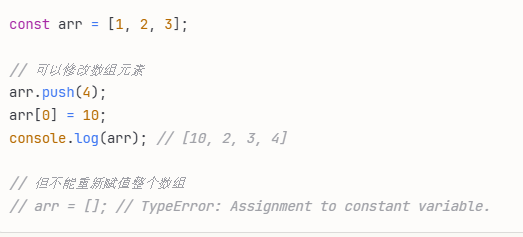


### 对象和数组的特殊情况

#### 对象属性可以修改



#### 数组元素可以修改

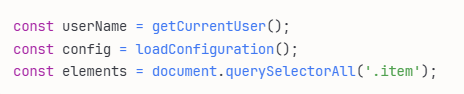


### 使用场景

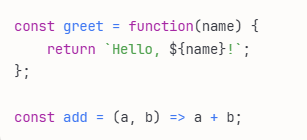
#### 1、声明常量值



#### 2、声明不会重新赋值的变量



#### 3、函数声明



### const {}

#### 基本概念

这是ES6引入的**解构声明**语法，{} 不是普通的对象字面量，而是**解构模式。**

#### 语法



1、const - 声明关键字

2、{ name, age } - 解构模式（不是对象！）

3、= ：赋值操作符

4、person - 数据源对象

#### 解构模式 vs 对象字面量

##### 对象字面量（创建对象）



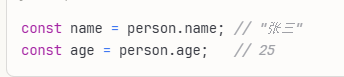
##### 解构模式（提取属性）



##### 执行过程

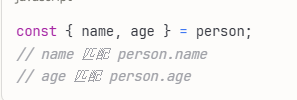


等价于



#### 模式匹配规则

##### 1、按属性名匹配



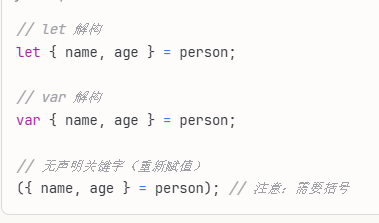
##### 2、顺序无关



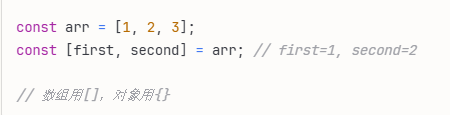
##### 3、部分提取



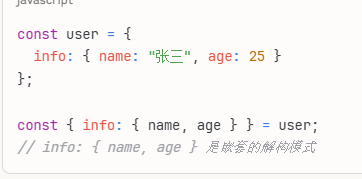
##### 4、其他声明关键字



#### 数组也支持解构



#### 嵌套解构模式



#### 核心概念

{} 在 = 左侧时是**解构模式**，告诉JavaScript：

1、从右侧对象中

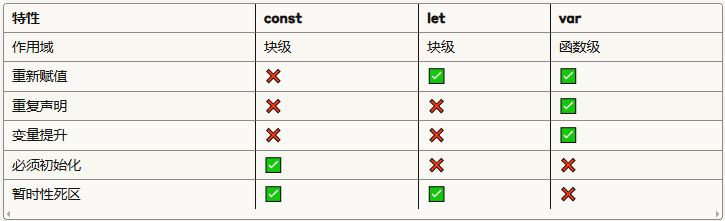
2、按属性名匹配

3、提取对应值

4、创建同名变量

这是一种声明变量的特殊语法，不是创建对象。

### 与let和var的比较



### 最佳实践

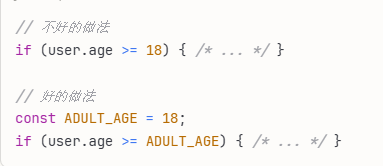
1、优先使用const: 默认使用const，只有当变量需要重新赋值时才使用let。

2、常量命名: 真正的常量使用大写字母和下划线：const MAX\_COUNT = 100

3、对象冻结: 如果需要完全不可变的对象，使用Object.freeze()：



4、**避免魔术数字**: 用命名常量代替直接的数字。



使用const可以让代码更加清晰，减少意外的变量修改，提高代码的可维护性。

## var

var声明的变量可以被多次赋值，例如：

var f = 100;

f=200;

### 特点

1、变量可以重复声明（覆盖原变量）。

2、变量未赋值时，默认值为 undefined。

3、var 声明的变量会提升（Hoisting），但不会初始化。

## let与var的区别

1、能用let就不要用var，var会出现作用域的问题。

# 变量提升

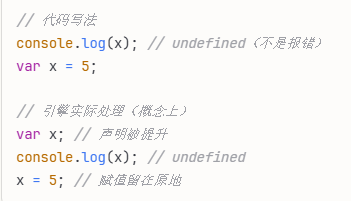
## 基本概念

变量提升（Hoisting）是JavaScript引擎在代码执行前，将变量和函数声明移动到其作用域顶部的行为。

## 提升原理

1、编译阶段：扫描代码，收集所有声明

2、执行阶段：按顺序执行代码



## var变量提升



## 函数作用域提升



## 函数提升

### 函数声明完全提升



### 函数表达式只提升变量



# TDZ

## 基本概念

**TDZ = 暂时性死区**，是指从作用域开始到变量声明完成之前的区域，在此区域内无法访问该变量。

## 存在范围

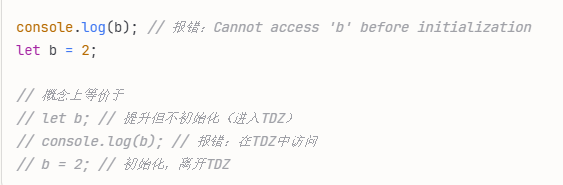


## TDZ vs var的区别

### var没有TDZ



### let/const有TDZ



# 数据类型

## 基本数据类型

字符串（String）、数字(Number)、布尔(Boolean)、空（Null）、未定义（Undefined）、Symbol。

Symbol 是 ES6 引入了一种新的原始数据类型，表示独一无二的值。

## 对象类型

对象(Object)、数组(Array)、函数(Function)，还有两个特殊的对象：正则（RegExp）和日期（Date）

## undefined

函数、表达式、对象的属性不存在时、声明了变量没有赋初值会返回undefined

## null

## string

定义

1、可以用单引号

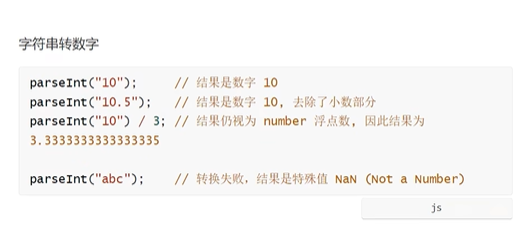
2、可以用双引号

3、可以用反引号



## number

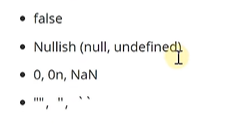




bigint表示整数

## boolean





这些都是falsy

## symbol

## 对象类型

函数、数组、对象



### Function

#### 默认参数的函数



#### 匿名函数



#### 箭头函数



#### 函数是对象





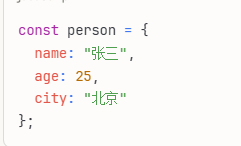
# 对象

## 基本概念

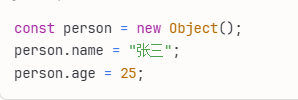
对象是键值对的集合，用于存储和组织相关数据及功能。

## 创建对象语法

### 1、字面量语法



### 2、构造函数语法

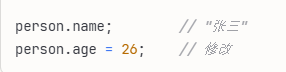


### 3、Object.create()

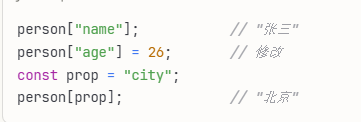


## 属性访问

### 点符号



### 方括号

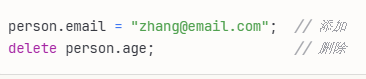


## 方法定义



## 属性操作

### 添加/删除



### 检查属性

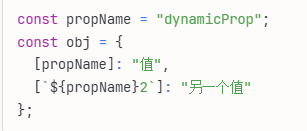


## ES6+新特性

### 属性缩写



### 计算属性名



### 解构赋值



### 嵌套对象



## 解构赋值

### 基本概念

解构赋值是从对象中提取属性值，并赋值给变量的语法糖。

### 语法

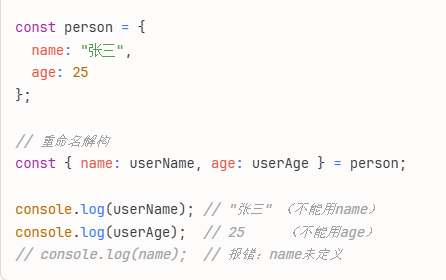


### 重命名语法

const { name: userName, age: userAge } = person;

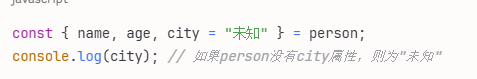
语法结构：

{ 原属性名: 新变量名 }



### 更多用法

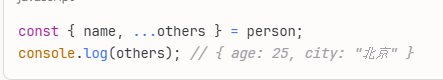
#### 默认值



#### 重命名+默认值



#### 剩余属性



#### 嵌套解构



#### 函数参数解构



#### 核心理解

1、{ name, age } = 创建同名变量

2、{ name: userName } = 创建重命名变量

3、左侧是**解构模式**，右侧是**数据源**

**4、**冒号左边是对象属性，右边是新变量名

这样可以快速提取对象属性到独立变量中使用。

# 函数

## 基本概念

函数是可重复使用的代码块，用于执行特定任务并可返回结果。

## 语法

### 1、函数声明



### 2、函数表达式

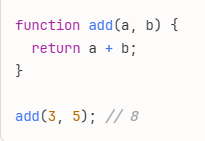


### 3、箭头函数（ES6）



## 参数和返回值

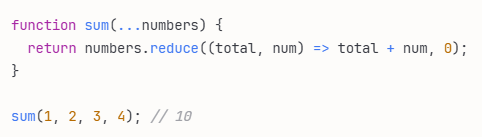
### 基本参数



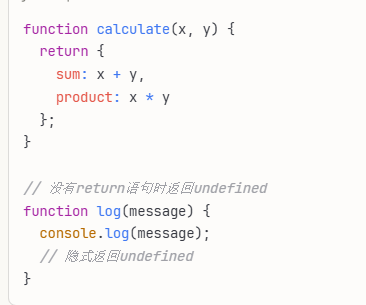
### 默认参数



### 剩余参数



### 返回值



## 函数调用方式

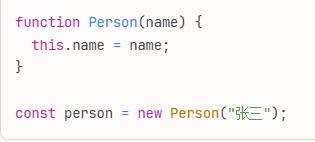
### 直接调用



### 作为对象方法



### 作为构造函数



# 箭头函数

## 基本概念

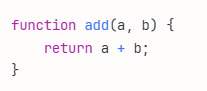
箭头函数是 ES6 中引入的一种新的函数声明方式，它提供了一种更简洁的语法来定义函数。

## 语法

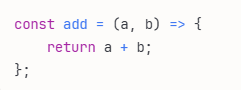
箭头函数的基本语法如下：

(参数列表) => { 函数体 }

### 传统函数写法



### 箭头函数写法



### 更简洁的写法（单表达式）



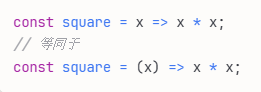
## 语法规则

### 参数规则

#### 无参数



#### 单个参数（可省略括号）



#### 多个参数（必须用括号）



#### 默认参数

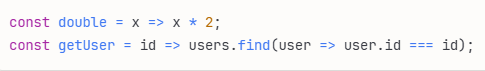


#### 剩余参数

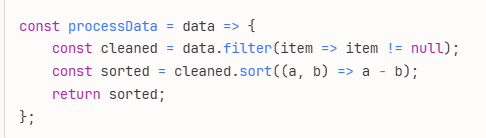


### 函数体规则

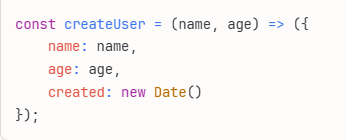
#### ****单行表达式****（自动返回）



#### ****多行代码块（需要显式 return）****



#### ****返回对象字面量**（需要用括号包裹）**



### 重要特性对比

#### 1、this 绑定的差异

**传统函数** - this 动态绑定



**箭头函数** - this 词法绑定



#### 2、不能作为构造函数

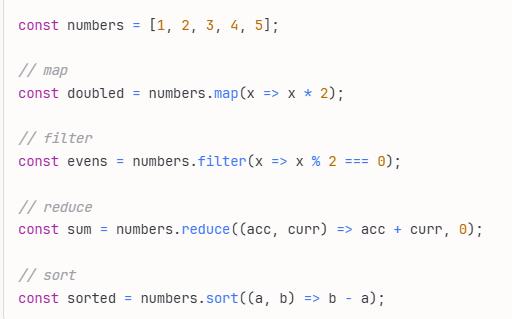


#### 3、没有 arguments 对象



## 常见使用场景

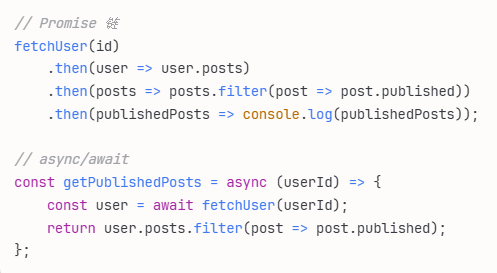
### 1、数组方法



### 2、事件处理



### 3、异步操作



## 最佳实践

### 适合使用

1、简短的工具函数

2、数组方法的回调

3、不需要 this 绑定的场景

4、需要保持外层 this 的回调

### 不适合使用

1、对象方法（需要 this 指向对象）

2、构造函数

3、需要 arguments 对象的场景

4、需要动态 this 的场景

## 总结

箭头函数虽然语法简洁，但理解其特性（特别是 this 绑定）是正确使用的关键。选择使用箭头函数还是传统函数，主要看具体的使用场景和 this 绑定的需求。

# JSX

## 基本概念

JSX 是 JavaScript XML 的缩写，是一种 JavaScript 的语法扩展。它主要用于 React 库中，用来描述用户界面的呈现形式。JSX 是 JavaScript 的语法扩展，它允许在 JavaScript 代码中编写类似 HTML 的标记，用来描述 UI 的结构。例如：



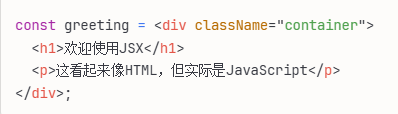
在上面的例子中，<h1>Hello, World!</h1> 就是使用 JSX 编写的类似 HTML 的标记，它会被转译成对应的 JavaScript 代码。

## 语法



## 核心特点

### 1、HTML-like语法



### 2、嵌入JavaScript表达式



### 3、组件使用

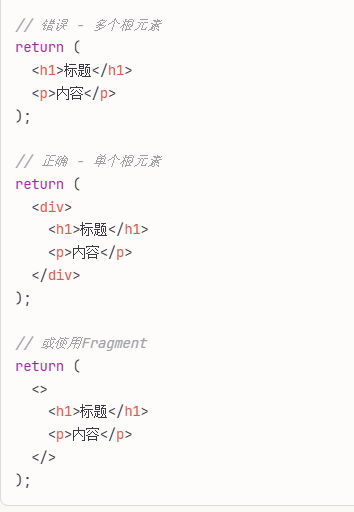


## 语法规则

### 属性命名



### 必须有根元素



## 框架支持

1、**React**: 原生支持JSX

2、**Vue**: 支持JSX语法（需配置）

3、**其他框架**: Preact、Solid等

## 总结

JSX让你在JavaScript中写类似HTML的代码，使组件开发更直观。

# 模版字符串

## 基本概念

在 HTML 中，模板字符串通常是指在 JavaScript 中的模板字面量（template literals），它们是 ES6 引入的一项功能，提供了一种更灵活、更强大的字符串格式化和拼接方式。

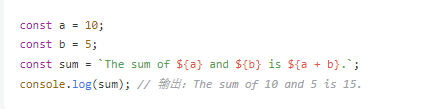
## 基本语法

使用反引号（`）来定义模板字符串，例如：



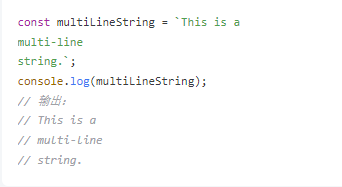
## 插入表达式

在模板字符串中，可以使用 ${} 语法插入 JavaScript 表达式，这些表达式会被计算并替换为结果字符串，例如：



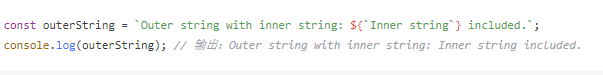
## 多行字符串

模板字符串可以包含多行文本，而无需使用 \n 或 + 连接符号，例如：



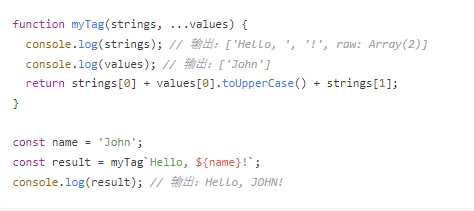
## 嵌套模版字符串

模板字符串可以嵌套在其他模板字符串中，使得字符串拼接更加灵活和直观，例如：



## 标签模版字符串

可以将模板字符串与函数结合使用，这样函数可以处理模板字符串中的内容，例如：



## 总结

总之，JavaScript 模板字符串是一种方便的字符串字面量，可以动态插入表达式、支持多行字符串，使得字符串处理更加简洁和易读。

# ${}

# 运算符

## !=

### 基本概念

!= 运算符是 JavaScript 中的一种比较运算符，表示“不等于”。它用于比较两个值是否不相等，如果两个值不相等，则返回 true，否则返回 false。

### 比较规则

1、当使用 != 运算符进行比较时，JavaScript 首先会对两个操作数进行类型转换，然后再进行比较。如果两个操作数的类型不同，JavaScript 会尝试将它们转换为相同的类型，然后再进行比较。

2、如果两个操作数的类型相同，则直接比较它们的值。如果值不相等，则返回 true，表示不等于；如果值相等，则返回 false，表示相等。

### 类型转换规则

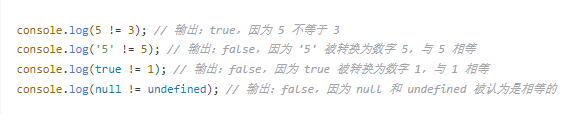
1、如果其中一个操作数是字符串，另一个操作数是数字，则 JavaScript 会尝试将字符串转换为数字，然后进行比较。

2、如果其中一个操作数是布尔值，则 JavaScript 会将布尔值转换为数字（true 转换为 1，false 转换为 0），然后进行比较。

3、如果其中一个操作数是对象，则 JavaScript 会调用该对象的 valueOf() 方法，然后再进行比较。如果 valueOf() 方法返回的是一个原始值，则继续比较；如果 valueOf() 方法返回的是一个对象，则会再次调用对象的 toString() 方法，然后再进行比较。

4、如果两个操作数都是 null 或 undefined，则它们被认为是相等的。

### 示例



## !==

### 基本概念

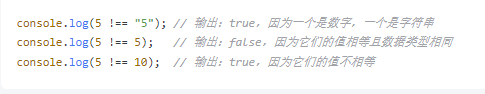
!== 是 JavaScript 中的严格不等运算符（Strict Unequal Operator）。它用于比较两个值是否不相等，且不会进行类型转换。如果两个值的类型和值都不相等，则返回 true；否则返回 false。

### 工作原理

1、当使用 !== 运算符进行比较时，首先会比较两个值的数据类型。如果两个值的数据类型不同，则它们被认为是不相等的，返回 true。

2、如果两个值的数据类型相同，那么会直接比较它们的值。如果值不相等，则返回 true；否则返回 false。

### 示例



### 与!=的区别

1、!== 是严格不等运算符，它会同时比较值和数据类型。

2、!= 是不等运算符，它只会比较值，不会考虑数据类型。在使用 != 运算符时，如果两个值的数据类型不同，JavaScript 会尝试进行类型转换，然后再进行比较。

## ||

### 基本概念

JavaScript 中的逻辑或运算符（||）是一个二元运算符，用于在两个操作数中返回第一个被视为真值的操作数。如果两个操作数都是假值，则返回第二个操作数。

### 语法

expression1 || expression2

### 行为

1、如果 expression1 是真值（不为假值），则结果为 expression1。

2、如果 expression1 是假值，则结果为 expression2。

3、如果 expression1 和 expression2 都是假值，则结果为 expression2。

### 真值和假值

1、在 JavaScript 中，以下值被视为假值（Falsy）：

false

null

undefined

0

NaN

空字符串 '' 或 ""

2、除了上述假值外，所有其他值都被视为真值（Truthy）。

### 应用场景

#### 1、设置默认值

可以使用逻辑或运算符为变量设置默认值。

#### 2、提供备选方案

当第一个表达式无效时，提供备选方案。

#### 3、处理可选参数

在函数中处理可选参数的默认值。

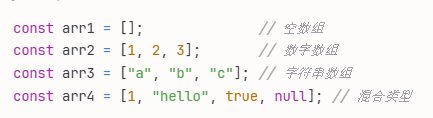
### 示例



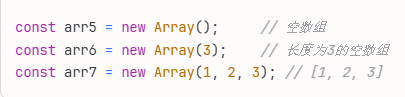
# 数组

## 1、数组创建

### 字面量语法（推荐）



### 构造函数语法

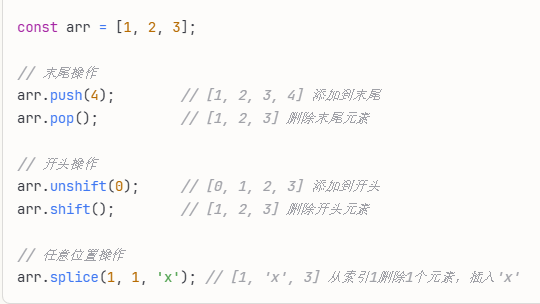


## 2、数组访问和修改

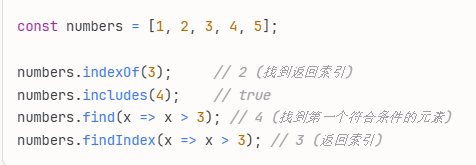


## 3、常用的数组方法

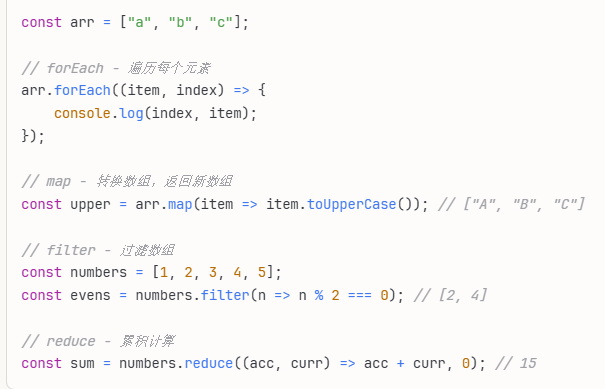
### 添加、删除元素



### 查找方法



### 遍历方法



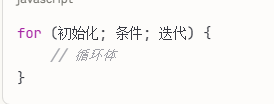
### 连接数组



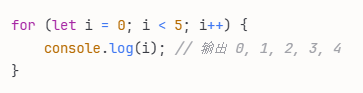
# 循环

## 1、for循环

### 基本语法



### 示例



### 执行流程

1、执行初始化表达式（只执行一次）

2、检查条件，如果为true继续，false则跳出循环

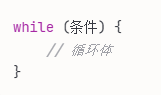
3、执行循环体

4、执行迭代表达式

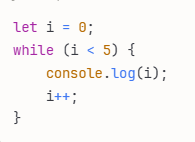
5、回到步骤2

## 2、while循环

### 基本语法



### 示例



### 特点

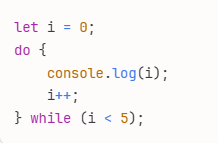
先判断条件再执行，条件为false时不执行循环体

## 3、do…while循环

### 基本语法



### 示例



### 特点

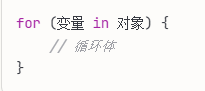
先执行一次循环体，再判断条件，至少执行一次

## 4、for…in循环

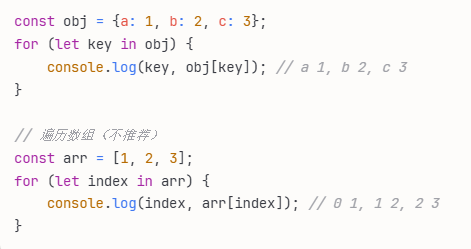
### 用途

遍历对象的可枚举属性。

### 基本语法



### 示例



## 5、for…of循环

### 用途

遍历可迭代对象（数组、字符串、Set、Map等）

### 基本语法



### 示例



## 6、forEach方法

### 基本概念

JavaScript中的forEach循环是数组的一个内置方法，用于遍历数组中的每个元素并对每个元素执行指定的函数。

### 语法



callback: 为数组中每个元素执行的函数，接收三个参数：

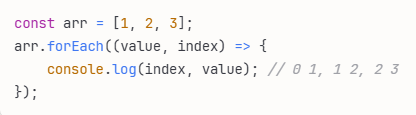
currentValue: 当前正在处理的元素

index (可选): 当前元素的索引

array (可选): 调用forEach的数组

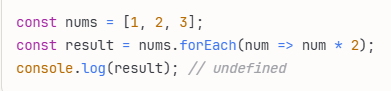
thisArg (可选): 执行callback时的this值

### 示例

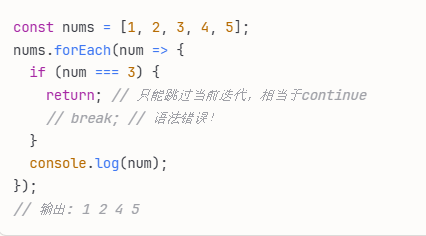


### 重要特性

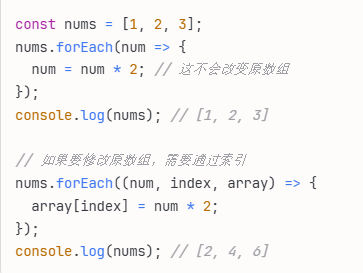
#### 1、无返回值



#### 2、无法使用break或者continue



#### 3、不会改变原数组



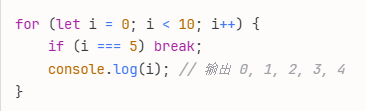
### 总结

forEach适合用于简单的数组遍历操作，特别是当你只需要对每个元素执行某些操作而不需要返回新数组时。如果需要转换数组并返回新数组，建议使用map；如果需要条件控制循环，使用传统的for循环可能更合适。

## 循环控制语句

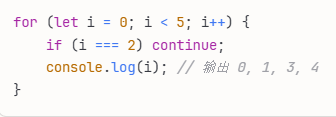
### break语句

跳出当前循环



### continue语句

跳过本次迭代，继续下一次



## 使用建议

1、遍历数组: 推荐使用 for...of 或 forEach

2、遍历对象属性: 使用 for...in

3、需要索引控制: 使用传统 for 循环

4、条件不确定次数: 使用 while

5、至少执行一次: 使用 do...while

## 性能考虑

传统for循环通常性能最好，for...of次之，forEach在某些情况下性能较差，但差异在大多数应用中可以忽略。选择循环方式时应优先考虑代码可读性和维护性。

# import

import 接收名称 from '模块名称'

## 基本概念

将其他 JavaScript 文件（模块）中的功能、变量或默认导出引入到当前文件中，以便在当前文件中可以使用这些导入的内容。

## 导入单个功能或者变量

1、使用 import 关键字，后跟要导入的功能或变量的名称，以及来源模块的路径。可以使用相对路径或绝对路径来指定来源模块的位置。

2、导入的功能或变量名称需与来源模块中导出的名称相匹配。



## 导入所有导出

使用 \* as 关键字，后跟一个名称，将所有导出导入为一个对象。



## 默认导入

1、使用 import 关键字，后跟一个名称，将模块的默认导出导入为该名称。

2、模块的默认导出不需要使用花括号 {} 指定名称。



## 注意事项

1、导入语句必须位于 JavaScript 文件的顶部，不能在函数内部或循环内部进行导入。

2、导入模块的路径必须是字符串常量，不能是变量或表达式。

# export

## 基本概念

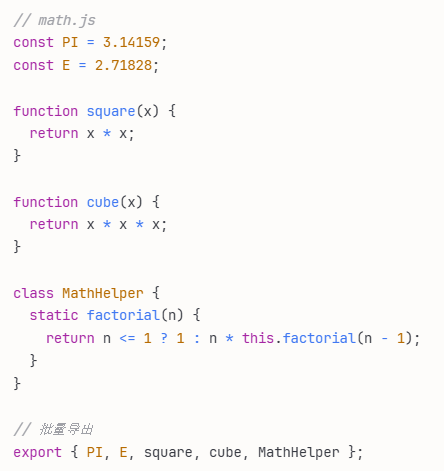
JavaScript 的 export 语法是 ES6 模块系统的导出机制，用于将模块内的功能暴露给外部使用。

## 命名导出

### 直接导出声明



### 先声明后导出



### 重命名导出



## 默认导出

### 语法

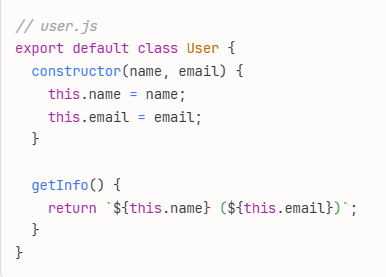
// 一个模块只能有一个默认导出

export default 值/函数/类/对象

### 导出函数



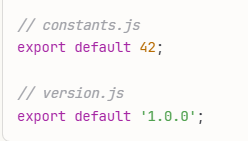
### 导出类



### 导出对象



### 导出值



### 特点

#### 1、唯一性

每个模块只能有一个默认导出

#### 2、匿名性

导出时可以不需要名称

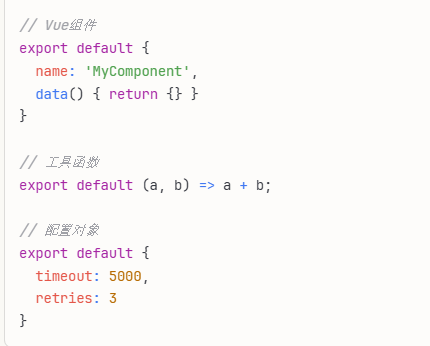
#### 3、灵活性

导入时可以自定义名称

#### 4、主要功能

通常用于导出模块的主要功能。

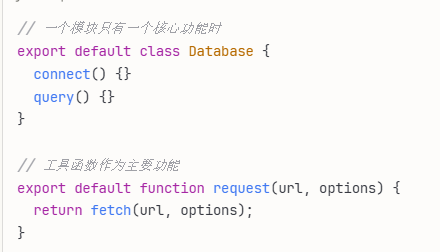
### 常见用法



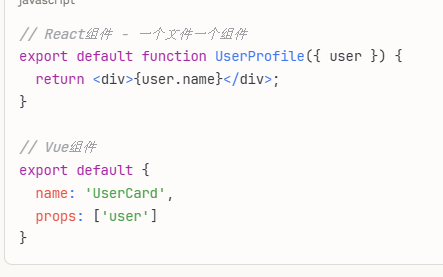
**记忆要点**：默认导出 = 一个模块的主角，导入时可以随意取名。

### 使用场景

#### 1、主要功能模块



#### 2、组件导出



#### 3、配置对象



#### 4、入口模块、主模块



#### 5、单一职责的工具类



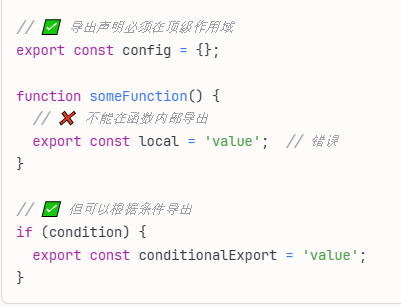
#### 6、常量集合



## 导出语法规则



## 导出时机

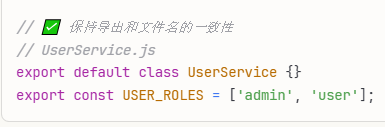


## 导出和导入的对应关系



## 最佳实践

### 1、命名一致性



### 2、合理使用默认导出



### 3、创建统一入口



## 注意事项

一个模块中只能用一次默认导出。

## 总结

export 语法提供了灵活的模块导出方式，合理使用可以让你的代码更好地组织和复用。关键是要根据模块的用途选择合适的导出方式：主要功能用默认导出，辅助功能用命名导出。

# 'use strict'

## 基本概念

在 JavaScript 文件中使用 'use strict'; 这行代码是为了启用严格模式（Strict Mode）。严格模式是 ECMAScript 5 引入的一种更加严格的 JavaScript 执行模式，它有助于消除一些 JavaScript 的静默错误，提高代码的安全性和性能。

## 作用

### 1、消除静默错误

在严格模式下，一些潜在的静默错误会抛出异常，如对只读属性赋值、变量未声明就被使用等。

### 2、提高性能

严格模式下的代码通常比非严格模式下的代码更容易优化，因此可能会获得更好的性能。

### 3、限制不安全的操作

 严格模式限制了一些不安全的 JavaScript 操作，防止一些意外行为的发生。

### 4、增强安全性

通过禁用一些不安全的特性，可以增强代码的安全性，减少潜在的安全漏洞。

## 影响

### 1、变量声明

严格模式要求变量必须先声明后使用，否则会抛出 ReferenceError 错误。

### 2、删除限制

不能删除不可删除的属性，如删除全局对象上的属性或不可配置的属性会抛出错误。

### 3、保留字

禁止使用一些保留字作为变量名，如 eval、arguments 等。

### 4、this指向

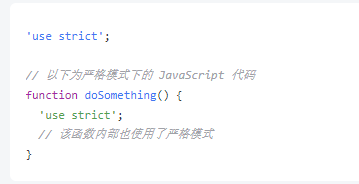
在全局执行环境中，严格模式下函数内部的 this 值为 undefined，而非全局对象。

### 5、eval限制

限制了对 eval 函数的使用，使其在独立作用域中执行。

## 使用方式

只需在 JavaScript 文件的顶部添加 'use strict'; 即可开启严格模式。示例：



总的来说，使用 'use strict'; 可以帮助开发者编写更加规范、安全和高效的 JavaScript 代码，避免一些常见的错误，并提高代码质量和性能。建议在现代 JavaScript 开发中始终启用严格模式，以获得更好的开发体验和代码质量。

# Contents are identical

"Contents are identical" 这句话的意思是“内容相同”或“内容一致”。在计算机领域中，这个短语通常用于指出两个或多个内容、文件或数据之间没有任何差异，它们完全相同，没有发生变化。

1、比较结果： 当进行内容比较时，如果系统或工具输出“Contents are identical”，则表示系统已经比较了两个内容，并确认它们在内容和结构上完全相同，没有发现任何差异。

2、文件比较： 在文件比较工具中，当比较两个文件并且它们的内容完全一致时，会输出这个信息，让用户知晓这两个文件没有任何改变。

3、数据校验： 在数据传输或存储过程中，对数据进行校验时，如果两份数据经过比较后结果为“Contents are identical”，则表示数据在传输或存储过程中没有发生损坏或篡改。

4、验证成功： 在某些情况下，系统需要验证两个内容是否一致，如果验证通过，则会显示“Contents are identical”，表明验证成功。

总的来说，“Contents are identical”是一个常见的提示，用于说明比较的两个内容完全相同，没有发生任何变化或差异。这种信息对于确认数据完整性、文件一致性或比较结果的准确性非常有帮助。

# 问题

## 1、no-undef

no-undef 是 ESLint（JavaScript 代码检查工具）的一个规则，用于检测未定义的变量。

**规则作用**：

no-undef 规则旨在避免在代码中使用未声明的变量，以减少由于拼写错误或者意外的全局变量污染而导致的错误。

**检测对象：**

no-undef 规则会检测 JavaScript 代码中的变量，如果发现使用了未声明的变量，则会给出相应的警告或错误信息。

**使用场景**

在 JavaScript 中，如果引用了一个未声明的变量，那么这个变量会被认为是全局变量，并且在严格模式下会抛出 ReferenceError 错误。no-undef 规则就是为了帮助开发者在编写代码时发现这类问题，提前修复。

**示例**



**解决方法**

1、声明变量：确保在使用变量之前先声明变量，可以使用 var、let 或 const 关键字进行声明。

2、检查拼写：确保变量的拼写正确，不要因为拼写错误而导致变量未定义。

3、使用严格模式：启用严格模式可以在一定程度上防止意外的全局变量污染。