第 5 章

CHAPTER 5

JavaScript 核心知识

微课视频 5

JavaScript 是一种脚本语言，它的正式名称为 ECMAScript[5]。

JavaScript 的基本代码示例如图 5-1 所示。作为一种脚本语言，JavaScript 具有“即插即用”的特点，它的变量不需要事先声明就可以在需要的时候直接使用。

```
let a = 1;
let b;
b = a + 1;
console.log(b) // output is 2
```

图 5-1　JavaScript 的代码示例

JavaScript 的版本发展如图 5-2 所示。目前最常用的版本为 ECMAScript 6。ECMAScript 6 相对于之前的版本变化较大，而 ECMAScript 6 之后的版本则变动较小。所以，掌握 ECMAScript 6 版本的 JavaScript 是学好 JavaScript 的一项必要技能。

JavaScript 版本	发布时间	主要特点
JavaScript 1.0	1996年	首次发布
ECMAScript1/2	1996 — 1998年	JavaScript开始标准化
ECMAScript 3	1999年	当今JavaScript的蓝本
ECMAScript 5	2009年	使用最广泛的JavaScript
ECMAScript 6 (ES6 or ES2015)	2015年	现代的JavaScript
ES2016～ES2019	2016 — 2019年	每年一个新版本迭代

图 5-2　JavaScript 的版本演进

JavaScript 的知识点比较庞杂，本章将重点详细讲解 JavaScript 中的基本语法、运行环境和核心概念。

本章在首先介绍 JavaScript 基本语法的基础上，介绍 JavaScript 在浏览器中的使用方法和关键知识点；然后介绍 JavaScript 中的面向对象编程和函数式编程的基本方法。所有内容都将结合实例进行示范讲解。本章应重点掌握以下要点：

（1）JavaScript 的基本语法；

（2）JavaScript 的面向对象编程；

（3）JavaScript 的函数式编程。

5.1 JavaScript 基本语法

JavaScript 具有脚本语言的优点与缺点。其优点是：入门简单；其缺点是：执行速度相对稍慢。

作为 JavaScript 和其他语言的对比，图 5-3 给出了 JavaScript 和 Java 语言的区别说明。

JavaScript	Java
解释型语言	编译型语言
主要用在前端	主要用在后端
运行在浏览器或Node上	运行在JVM上
在前端实现安全性较困难	在后端实现安全性较容易
动态类型	静态类型
语法更像C	语法更像C++
可以用任何文本编辑器开发	依赖JDK进行开发

图 5-3 JavaScript 和 Java 的不同点

5.1.1 变量和数据类型

变量是计算机内存中一个有名字的用来存储值的存储空间。通过设置变量这种方式，就可以有效地获取某个值。在 JavaScript 程序中，变量的名字也被称为标识符。标识符的命名规则为：必须由字母、下画线(_)、美元符号($)和数字组成，并且只能以字母、下画线(_)和美元符号($)开始。JavaScript 是一种区分大小写的计算机语言，所以大写字母和小写字母在 JavaScript 中表示不同的字符。JavaScript 中共有 3 种声明变量的方式：var、let 和 const，如图 5-4 所示。

```
var x = 1; //定义全局变量x, 并赋值1。
let y = 1; //定义局部变量y, 并赋值1。
const z = 1; //定义局部常量z, 并赋值1。
```

图 5-4 JavaScript 变量声明示例

虽然 JavaScript 有 3 种声明变量的方式，但在使用时通常优先使用 const，如果是需要改变值的变量通常优先使用 let。var 作为 ES6 之前的变量声明方式通常不再建议使用。

数据类型是指变量所存储值的类型。对于静态类型的编程语言(例如 Java)，在声明变量的时候就要指定该变量的数据类型。JavaScript 作为一种动态类型的编程语言，不需要在声明变量的时候指定数据类型。JavaScript 中变量的数据类型会随着变量所存储值的数据类型的变化而动态变化。JavaScript 中有 9 种数据类型，包括 7 种基本数据类型和 2 种索引数据类型，如图 5-5 所示。

JavaScript 中的基本数据类型是指非对象并且没有对象方法的数据类型，其中 null 是一种特殊的基本数据类型。JavaScript 中的索引数据类型是对象型的数据类型，包括

JavaScript数据类型	
基本型数据类型(Primitive)	索引型数据类型(Reference)
Number	Object
String	
Boolean	
null	Function
undefined	
Symbol	
BigInt	

图 5-5 JavaScript 中数据类型示例

Object、Array、Function、Set、WeakSet、Map 和 WeakMap，其中 Function 是一种特殊的索引数据类型。JavaScript 中的数据类型通常可以通过 typeof 操作符来判定，只是要注意 typeof 在处理 null 时会给出值 object，如图 5-6 所示。

```
const a = undefined,
      b = true,
      c = 1,
      d = 'hello',
      e = 1n,
      f = Symbol('Sym'),
      g = null,
      h = {},
      i = function() {};

console.log(typeof a); // output is "undefined"
console.log(typeof b); // output is "boolean"
console.log(typeof c); // output is "number"
console.log(typeof d); // output is "string"
console.log(typeof e); // output is "bigint"
console.log(typeof f); // output is "symbol"
console.log(typeof g); // output is "object"
console.log(typeof h); // output is "object"
console.log(typeof i); // output is "function"
```

图 5-6 JavaScript 数据类型的验证示例

5.1.2 操作符和控制语句

操作符是可以对变量进行某种操作的标识符。JavaScript 中主要的操作符包括赋值操作符、逻辑操作符、数学运算操作符、比较操作符、分组操作符、void 操作符、位运算操作符、in 操作符、管道操作符、逗号操作符、delete 操作符、new 操作符等，如图 5-7 所示。

控制语句分为赋值语句、条件语句和循环语句。

赋值语句是通过赋值操作符对一个变量赋值。JavaScript 在进行赋值操作时将赋值操作符右边的值赋给操作符左边的变量，如图 5-8 所示。

JavaScript中的操作符	
赋值操作符	=
逻辑操作符	&&, ||, !
数学运算操作符	+, -, *, /, %, **, ++, --,
比较操作符	==, ===, !=, !==, >, >=, <, <=
分组操作符	()
void操作符	void
位运算操作符	&, |, ^, ~, <<, >>, >>>
in操作符	in
条件操作符	? :
逗号操作符	,
delete操作符	delete
new操作符	new

图 5-7　JavaScript 中主要的操作符

```
let x = 1;
console.log(x); // output is 1
```

图 5-8　JavaScript 中的赋值语句示例

JavaScript 中的条件语句分为由关键字 if 构成的条件语句和由条件操作符构成的条件语句，这两种实现方式如图 5-9 所示。

```
/*==========================================
The following should output 'Correct!'
==========================================*/
/* if 构成的条件表达式 */
const x = 1;
if(x == 1) {
    console.log('Correct!');
} else {
    console.log('Wrong!');
}
/* 条件操作符构成的条件表达式 */
const y = 2;
console.log(y == 2? 'Correct!' : 'Wrong!');
```

图 5-9　JavaScript 中的条件语句示例

JavaScript 中的循环语句有多种代码实现方式，基本可以分为由操作符构成的循环语句和由函数构成的循环语句两大类。

由操作符构成的循环语句是指由 for、for…in、for…of、while 和 do while 构成的循环语句。图 5-10 给出了由操作符构成的循环语句，实现打印 1～10 的 10 个数字的不同方式。

由函数构成的循环语句是指由 Array 的函数 forEach、map、filter、reduce、every、some、find 等具有循环功能的函数构成的循环实现方式语句。由函数构成的循环语句是函数式编程中经常使用的一种循环语句实现方法。图 5-11 给出了由函数构成的循环语句，实现打印 1～10 的 10 个数字的不同方式。

```
/*=========================================
操作符构成的循环语句
=========================================*/
for (let i = 0; i < 10; i++) {
    // 输出1-10
    console.log(i + 1);
}
let i = 0;
do {
    console.log(i + 1);
    i++;
} while(i < 10)
let j = 0;
while(j < 10) {
    console.log(j + 1);
    j++;
}

let a = [1,2,3,4,5,6,7,8,9,10];
for(let i in a) {
    console.log(a[i]);
}
for(let i of a) {
    console.log(i);
}
```

图 5-10 由操作符构成的循环语句的不同实现方式

```
/*=========================================
函数构成的循环语句
=========================================*/
// 输出1-10
const a = [1,2,3,4,5,6,7,8,9,10];

a.forEach(i => console.log(i));

a.map(i => console.log(i))

a.filter(i => console.log(i));

a.reduce((i,j) => console.log(i = j),a[0]);

a.every(i => console.log(i) === undefined);

a.some(i => console.log(i) !== undefined);

a.find(i => console.log(i) !== undefined);
```

图 5-11 由函数构成的循环语句的不同实现方式

5.1.3 JavaScript 程序示例

本节展示如何使用 JavaScript 中的基本语法来完成一个程序实例。

任务：对于给定的一个由数字组成的数组，找出其中为奇数的数字，生成一个新的数组，将这个新的数组升序排列，将排列好的数组的每一项乘以 2，生成一个新的数组返回。

程序实现：实现该功能的 JavaScript 代码如图 5-12 所示。

```
const a = [2, 9, 5, 16, 3, 8, 4];
function solution(arr) {
    const n_arr = [];
    for(let i of arr) {
        if(i % 2 === 0) {
            n_arr.push(i);
        }
    }
    for(let i = 0; i < n_arr.length; i++) {
        for(let j = 0; j < n_arr.length - i; j++) {
            if(n_arr[j] > n_arr[j+1]) {
                let tmp = n_arr[j+1];
                n_arr[j+1] = n_arr[j];
                n_arr[j] = tmp;
            }
        }
    }
    for(let i in n_arr) {
        n_arr[i] *= 2;
    }

    return n_arr;

}
console.log(solution(a)); // output is [4, 8, 16, 32]
```

图 5-12 JavaScript 实现代码示意图

5.2 JavaScript 的面向对象编程

JavaScript 语言是基于对象的，而不是面向对象的。JavaScript 中有对象的概念，例如，对象数据类型具有对数据和方法封装的功能，但是没有继承的概念。从定义的角度来看，JavaScript 中有两种类型的对象：一种是 JavaScript 语言本身提供的内置对象，例如 Date 对象或浏览器及 Node 运行环境提供的对象；另一种是用户根据需要自己创建的对象。

面向对象编程是指采用创建类和对象的方法来思考问题、解决问题的一种编程方法。JavaScript 中提供了 class 关键字来模拟定义传统面向对象编程中的类的概念。这种通过 class 关键字声明的类本质上是一种基于原型的实现。图 5-13 给出了 JavaScript 中定义类的基本语法。当使用 class 关键字创建好一个类后，可以用 new 关键字来从该类派生出新的对象，对象将继承类的所有属性和方法。

```
/*==========================================
JavaScript class declaration and usage
==========================================*/

class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age
    }
    sayHi () {
        console.log(`My name is ${this.name} and my age is ${this.age}`);
    }
}

const xiaoming = new Person('xiaoming', 21);
xiaoming.sayHi(); // output shoud be 'My name is xiaoming and my age is 21'
```

图 5-13 JavaScript 中类的定义方法

5.2.1 JavaScript 面向对象编程的概念和原则

JavaScript 面向对象编程的概念是基于面向对象编程(OOP)的理论，用 JavaScript 模拟对象、创建对象，使用对象来编程的一种方法。OOP 的基本概念是在程序中使用对象来模拟现实世界中的所有物体。对象可以包含属性和方法，属性通常用来存储数据。

下面用一个简单的例子讲解面向对象编程的基本方法。

假设创建了 Person 这个类，如图 5-13 所示。现在要从此类创建多个不同的 Person，如图 5-14 所示。图 5-14 中的 Person1 和 Person2 都将具有 Person 这个类中的 sayHi 方法。这种从 Person 类创建 Person1 和 Person2 对象的过程叫作 Person 类的对象实例化。

```
/*=========================================
Creating Person instances
=========================================*/

const Person1 = new Person('xiaoming', 21);
const Person2 = new Person('xiaohong', 20);
Person1.sayHi();// output shoud be 'My name is xiaoming and my age is 21'
Person2.sayHi();// output shoud be 'My name is xiaohong and my age is 20'
```

图 5-14 创建 Person 对象实例

更多的情况下，我们不需要从 Person 类创建实例，因为这样会导致所有的实例对象都大同小异。我们可能需要创建更个性化的实例，例如教师和学生。我们可以采用从类创建子类的方法来实现这个要求。每个子类都可以具有不同的个性化的属性和方法。同时，每个子类又可以具有共同父类的所有属性和方法。如图 5-15 所示。图 5-15 中的 Teacher 和 Student 都将具有 Person 这个类中的 name 和 age 属性及 sayHi 方法。但是 Teacher 和 Student 又都具有不同的额外属性和方法。

```
/*=========================================
Creating subclasses from Person
=========================================*/

class Teacher extends Person {
    constructor(name, age, lesson) {
        super(name, age);
        this.lesson = lesson;
    }
    teach() {
        console.log(`I am a teacher and I teach ${this.lesson}`)
    }
}
class Student extends Person {
    constructor(name, age, lesson) {
        super(name, age);
        this.lesson = lesson;
    }
    learn() {
        console.log(`I am a student and I learn ${this.lesson}`)
    }
}
```

图 5-15 由父类创建子类代码示意图

创建好子类 Teacher 和 Student 后，可以从 Teacher 和 Student 这两个子类创建不同的对象实例，这些对象实例除了有共同父类 Person 的方法外还具有各自 Teacher 和 Student 子类的独特属性和方法，如图 5-16 所示。

```
/*==========================================
Creating instances from subclasses
==========================================*/

const teacher1 = new Teacher('laoli', 40, 'English');
teacher1.sayHi(); //output shoud be 'My name is laoli and my age is 40'
teacher1.teach(); //output shoud be 'My name is laoli and I teach English'

const student1 = new Student('xiaoli', 19, 'English');
student1.sayHi(); //output shoud be 'My name is xiaoli and my age is 19'
student1.learn(); //output shoud be 'My name is xiaoli and I learn English'
```

图 5-16　由子类创建实例方法示意图

5.2.2　JavaScript 面向对象编程的程序示例

本节展示如何使用 JavaScript 中的面向对象的方法来完成一个程序实例。

任务：对于给定的一个由数字组成的数组，找出其中为奇数的数字，生成一个新的数组，将这个新的数组升序排列，将排列好的数组的每一项乘以 2，生成一个新的数组返回。

程序实现：实现该功能的 JavaScript 代码如图 5-17～图 5-19 所示。

```
/*==========================================
JavaScript面向对象实现代码：对于给定的一个由数字组成的数组，
找出其中为奇数的数字，生成一个新的数组，将这个新的数组
升序排列，将排列好的数组的每一项乘以2，生成一个新的数组返回。
==========================================*/

class MyArray {
    constructor(arr) {
        this.arr = arr;
        this.result = [];
    }
    getOddNum() {
        for(let i of this.arr) {
            if(i % 2 == 0) {
                this.result.push(i);
            }
        }
    }
}
```

图 5-17　定义父类

```
class MySubArray extends MyArray {
    constructor(arr) {
        super(arr);
        this.result = []
    }
    sortArr() {
        this.result = Array.from(this.arr);
        for(let i = 0; i < this.result.length; i++) {
            for(let j = 0; j < this.result.length - i; j++) {
                if(this.result[j] > this.result[j+1]) {
                    let tmp = this.result[j+1];
                    this.result[j+1] = this.result[j];
                    this.result[j] = tmp;
                }
            }
        }
    }
    mulplyArr() {
        for(let i in this.result) {
            this.result[i] *= 2;
        }
    }
}
```

图 5-18 定义子类

```
const a = new MyArray([2, 9, 5, 16, 3, 8, 4]);
a.getOddNum();
b = a.result;
const c = new MySubArray(b);
c.sortArr();
c.mulplyArr();
const d = c.result;
console.log(d); // output is [4, 8, 16, 32]
```

图 5-19 实例化并获得结果

5.3 JavaScript 的函数式编程

函数式编程是和面向对象编程并列的另一种编程范式。函数式编程的主要概念是不改变原始数据(immutability)和纯函数(pure function)。通过编写不产生任何副作用的纯函数,并且让数据在纯函数间流动,使函数式编程更加有利于系统的可维护性。JavaScript 中函数即是变量的特点使 JavaScript 天生具有可使用函数式编程的便利性。

5.3.1 JavaScript 函数式编程的概念和原则

为了更好地了解函数式编程,需要首先了解纯函数的概念。纯函数有两个特点:一个是给定相同的输入,该函数总是给出相同的输出;二是该函数没有副作用,即不会改变输入的原始数据。图 5-20 给出一个计算圆的面积的纯函数的例子。

```
/*==========================================
Pure function to calculate the area of a circle
==========================================*/

let PI = 3.14;
const calculateArea = radius ⇒ radius ** 2 * PI;
calculateArea(1); //output should be 3.14
```

图 5-20　计算圆面积的纯函数示意图

图 5-20 中的 PI 是一个变量，如果 PI 的值发生了改变，那么该函数 calculateArea 就不再是一个纯函数了。为了避免这种情况发生，可以将图 5-20 中的纯函数修改为鲁棒性更强的纯函数，如图 5-21 所示。

```
/*==========================================
A More robust pure function to calculate the area of a circle
==========================================*/

let PI = 3.14;
const calculateArea = (radius, pi) ⇒ radius ** 2 * pi;
calculateArea(1, PI); //output should be 3.14
```

图 5-21　鲁棒性更强的计算圆面积的纯函数示意图

通过以上例子应该能够很好地理解纯函数的概念。下面来看看函数式编程的另一个重要方面——不改变原始数据性，即不变性。

如果数据具有不可变性，则说明数据被创建好后将不再改变。但是程序的运行将不可避免地改变某些数据，所以我们能做的就是在需要改变数据的时候不断创建新的数据，而让新的数据和原来的数据具有相同的值。这就是函数式编程中实现原始数据不变性的原则。

同样，以图 5-21 计算圆的面积的纯函数为例。如果原始数据是以数组的形式保存的，而函数的参数同样是数组的形式，那么如图 5-21 所示的纯函数将变为如图 5-22 所示的形式。

```
/*==========================================
Input data in the form of Array to calculate the area of a circle
==========================================*/

let arr = [1, 3.14, 0] //radius, PI, and result
const calculateArea = (arr) ⇒ arr[2] = arr[0] ** 2 * arr[1];
calculateArea(arr);
console.log(arr[2]) //output should be 3.14
```

图 5-22　不具有不变性的纯函数示意图

为了使纯函数具有不变性，可以将如图 5-22 所示的纯函数改为如图 5-23 所示的纯函数。

```
/*=========================================
Input data in the form of Array to calculate the area of a circle
=========================================*/

let arr = [1, 3.14, 0] //radius, PI, and result
const calculateArea = (arr) => {
    const n_arr = Array.from(arr);
    n_arr[2] = n_arr[0] ** 2 * n_arr[1];
    return n_arr;
}
const n_arr = calculateArea(arr);
console.log(n_arr[2]) //output should be 3.14
```

图 5-23 具有不变性的纯函数示意图

5.3.2 JavaScript 函数式编程的程序示例

本节展示如何使用 JavaScript 中的面向对象的方法来完成一个程序实例。

任务：对于给定的一个由数字组成的数组，找出其中为奇数的数字，生成一个新的数组，将这个新的数组升序排列，将排列好的数组的每一项乘以 2，生成一个新的数组返回。

程序实现：实现该功能的 JavaScript 代码分别如图 5-24～图 5-26 所示。

```
/*=========================================
JavaScript面向函数实现代码：对于给定的一个由数字组成的数组，
找出其中为奇数的数字，生成一个新的数组，将这个新的数组
升序排列，将排列好的数组的每一项乘以2，生成一个新的数组返回。
=========================================*/

const getOddNum = arr => {
    const n_arr = [];
    for(let i of arr) {
        if(i % 2 == 0) {
            n_arr.push(i);
        }
    }
    return n_arr;
}
```

图 5-24 实现获取奇数功能的纯函数

```
const sortArr = arr => {
    const n_arr = Array.from(arr);
    for(let i = 0; i < n_arr.length; i++) {
        for(let j = 0; j < n_arr.length - i; j++) {
            if(n_arr[j] > n_arr[j+1]) {
                let tmp = n_arr[j+1];
                n_arr[j+1] = n_arr[j];
                n_arr[j] = tmp;
            }
        }
    }
    return n_arr;
}
```

图 5-25 实现排序功能的纯函数

```
const multiplyArr = arr ⇒ {
    const n_arr = Array.from(arr);
    for(let i in n_arr) {
        n_arr[i] *= 2;
    }
    return n_arr;
}

const a = [2, 9, 5, 16, 3, 8, 4];
const b = getOddNum(a);
const c = sortArr(b);
const d = multiplyArr(c);
console.log(d); // output is [4, 8, 16, 32]
```

图 5-26　实现对数字做乘法功能的纯函数及结果

5.4　ES6 基础知识

ES6 的正式名称为 ECMAScript 2015(ES 2015)。与上一个版本 ES5 相比,ES6 增加了很多特性。由于这些特性相较之前的版本有较大变化,因此本节将详细讲解 ES6 中增加的主要特性。

本节将要讲解的 ES6 中的主要特性为:函数的默认参数、字符串模板、多行字符串、解构赋值、对象表达式、箭头函数、Promise、let 和 const、类、模块、可计算的对象属性、for…of 语句、getters 和 setters。

5.4.1　ES6 的主要特性

1. 函数的默认参数

在 ES6 中,我们可以在声明定义函数时给出函数参数的默认值,而在 ES5 中我们只能在函数体中通过编程实现,如图 5-27 所示。

```
/* Default parameters in ES6 */
const sayHi = function(name = 'xiaoming') {
    console.log('Hello ' + name);
}

/* Default parameters in ES5 */
const sayHello = function(name) {
    console.log(console.log(name || 'xiaoming'))
}
```

图 5-27　ES6 和 ES5 中函数参数默认值的程序示例

2. 字符串模板

在 ES6 中，我们可以用反引号来创建字符串，在反引号中可以通过 ${} 来获取变量的值，而在 ES5 中只能通过字符串的拼接实现相同的功能，如图 5-28 所示。

```
/* Template Literals in ES6 */
const name = 'xiaoming';
const greetings = `Good morning ${name}`;

/* Template Literals in ES5 */
const name = 'xiaoming';
const greetings = 'Good morning ' + name;
```

图 5-28 ES6 和 ES5 中字符串模板的程序示例

3. 多行字符串

在 ES6 中，我们可以用反引号来创建多行字符串，而在 ES5 中只能通过对字符串的拼接实现相同的功能，如图 5-29 所示。

```
/* Multi-line Strings in ES6 */
const greetings = `Good morning xiaoming.
                   How are you these days?`;

/* Multi-line Strings in ES5 */
const greetings = 'Good morning xiaoming.\n'
                + 'How are you these days?';
```

图 5-29 ES6 和 ES5 中多行字符串的程序示例

4. 解构赋值

在 ES6 中，我们可以对数组或者对象进行解构，并将数组的元素的值赋给具有同样结构的变量数组，或者将对象的属性值赋给具有同样结构和名称的变量对象。这种解构赋值在 ES5 中只能通过编程来实现，如图 5-30 所示。

5. 对象表达式

在 ES6 中，当对象的属性名称和表示属性值的变量名称相同时，可以采用只写属性名称的简写方式。而在 ES5 中不存在这种简写方式，如图 5-31 所示。

6. 箭头函数

ES6 中引入了函数表达式的新的写法——箭头函数。箭头函数除了可以使函数表达式的写法更加简洁之外，还可以使函数中出现的 this 有固定的指代——调用该函数的上下文对象。这有别于非箭头函数中 this 在不同使用环境下具有不同指代对象的问题。ES6 中的箭头函数和 ES5 中的普通函数的写法的程序示例如图 5-32 所示。

```
/* Destructuring Assignment in ES6 */
const obj = {name: 'xiaoming', age: 21}
const arr = ['xiaoming', 21];
const {name, age} = obj;
const [aName, aAge] = arr;
console.log(name); // 'xiaoming'
console.log(age); // 21
console.log(aName); // 'xiaoming'
console.log(aAge); // 21

/* Destructuring Assignment in ES5 */
const obj = {name: 'xiaoming', age: 21}
const arr = ['xiaoming', 21];
const name = obj.name;
const age = obj.age;
const aName =arr[0];
const aAge = arr[1];
console.log(name); // 'xiaoming'
console.log(age); // 21
console.log(aName); // 'xiaoming'
console.log(aAge); // 21
```

图 5-30　ES6 和 ES5 中解构赋值的程序示例

```
/* Enhanced Object Literals in ES6 */
function buyFood(fish, meat, vegetable) {
    return {
        fish,
        meat,
        vegetable
    }
}

buyFood("Salmon", "Beef", "Carot");

/* Object Literals in ES5 */
function buyFood(fish, meat, vegetable) {
    return {
        fish: fish,
        meat: meat,
        vegetable: vegetable
    }
}

buyFood("Salmon", "Beef", "Carot");
```

图 5-31　ES6 和 ES5 中对象表达式的程序示例

```
/* Arrow Functions in ES6 */
const sayHi = name ⇒ console.log(name);

sayHi();

/* Functions in ES5 */
const sayHi = function(name) {
    console.log(name);
};

sayHi();
```

图 5-32　ES6 和 ES5 中箭头函数的程序示例

7. Promises

Promises 是用来完成异步功能的对象，通过使用该对象的返回值的不同方法，可以获取 Promises 处理的值。ES6 中提供了原生的 Promises 对象。而在 ES5 中只能通过复杂的编程或者借用其他的库的方法实现相同的功能。由于 ES5 实现代码的复杂性，这里只是给出 ES6 代码的实现，如图 5-33 所示。

8. 块级作用域的变量声明关键字

ES6 中引入了块级作用域的变量声明关键字 let 和 const。所谓块级作用域，就是由花括号包围的区域。所以，let 和 const 与 ES5 中的 var 的主要区别就是：var 的作用域是由最近的函数体决定的，而 let 和 const 的作用域是由最近的花括号和 var 相比决定的。另外，

```
/* Promises in ES6 */
const p = new Promise((resolve, reject) ⇒ {
    setTimeout(() ⇒ resolve('success'), 2000);
});
p.then((data) ⇒ console.log(data));
```

图 5-33 ES6 中 Promise 的程序示例

let 和 const 还具有只能声明一次、不具有变量提升的特点，const 可以用来声明常量。在学习曲线方面，let 和 const 比 var 更加简单。很多时候需要运行程序后才能知道 var 带来的准确含义，而 let 和 const 在没有运行的时候就能够让学习者了解不同变量的作用范围。ES6 和 ES5 中不同变量声明关键字的使用程序示例如图 5-34 和图 5-35 所示。

```
/* Block-Scoped Constructs Let and Const in ES6 */
const num = 10;
for(let i = num; i < num; i++) {
}
console.log(i); // Uncaught ReferenceError: i is not defined

/* Function-Scoped Constructs Var in ES5 */
const num = 10;
for(var i = num; i < num; i++) {
}
console.log(i); // 10
```

图 5-34 ES6 和 ES5 中块级作用域实现的程序示例 1

```
/* Block-Scoped Constructs Let and Const in ES6 */
function getNum(value) {
    let num = 0;
    if(value) {
        let num = 1;
    }
    return amount;
 }
 console.log(getNum(true)); // output: 0

/* Function-Scoped Constructs Var in ES5 */
function getNum(value) {
    var num = 0;
    if(value) {
        var num = 1;
    }
    return amount;
}
console.log(getNum(true)); // output: 1
```

图 5-35 ES6 和 ES5 中块级作用域实现的程序示例 2

虽然 const 是定义一个常量，但是如果该常量是对象，那么 const 保存的就是该对象的索引。用 const 定义对象的问题如图 5-36 所示，解决方案如图 5-37 所示。

```
/* problem of const */
const obj = {name: 'xiaoming'};
console.log(obj.name); // 'xiaoming'

obj.name = 'xiaohong';
console.log(obj.name); // 'xiaohong
```

图 5-36　使用 const 声明对象的程序示例

```
/* solution of const */
const obj = {name: 'xiaoming'};
Object.freeze(obj);
console.log(obj.name); // 'xiaoming'

obj.name = 'xiaohong';
console.log(obj.name); // 'xiaohong
```

图 5-37　使用 const 声明对象时解决方案的程序示例

9. 类的实现

在 ES6 中，我们可以用关键字 class 来创建类，虽然 JavaScript 是一种基于原型的编程语言，但是关键字 class 和其他相关关键字给 JavaScript 提供了模拟面向对象的编程语法。而在 ES5 中只能通过编程的方式来模拟实现面向对象的编程范式。图 5-38 给出了 ES6 中类的定义与使用以及 ES5 中对类的定义与使用的示例。图 5-39 给出了 ES6 中子类的定义与使用以及 ES5 中对子类的定义与使用的示列。

```
/* Classes in ES6 */
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    getName() {
        return this.name;
    }
}
const person1 = new Person('xiaoming', 21);
console.log(person1.getName()); // 'xiaoming'
/* Classes in ES5 */
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.getName = function() {
        return this.name;
    }
}
const person1 = new Person('xiaoming', 21);
console.log(person1.getName()); // 'xiaoming'
```

图 5-38　ES6 和 ES5 中类的实现的程序示例

需要注意的是，有些类的特性，例如类中的私有方法，还处于实验阶段，暂时无法在浏览器中使用。遇到这种情况，使用 babel 进行转义是通常使用的一种方法。同理，其他

```
/* Subclasses in ES6 */
class Student extends Person {
    constructor(name, age, score) {
        super(name, age);
        this.score = score;
    }
    getScore() {
       return `${this.getName()} has score of ${this.score}`;
    }
}
const student1 = new Student('xiaoming', 21, 90);
console.log(student1.getScore()); // 'xiaoming has score of 90'
/* Subclasses in ES5 */
function Student(name, age, score) {
    Person.call(this, name, age);
    this.score = score;
    this.getScore = function() {
        return `${this.getName()} has score of ${this.score}`;
    }
}
const student1 = new Student('xiaoming', 21, 90);
console.log(student1.getScore()); // 'xiaoming has score of 90'
```

图 5-39 ES6 和 ES5 中子类的实现的程序示例

暂时在浏览器中不能够使用的新特性往往也可以通过 babel 转义来实现。使用 babel 来实现类中的私有属性的程序示例如图 5-40 所示。使用 babel 的 package.json 文件和命令如图 5-41 所示。使用 npm install 命令安装依赖包后在命令行进行文件转义的命令如图 5-42 所示。

```
class Person {
    constructor(name) {
        this.name = name;
    }
    #hi() {
        return 'hello world';
    }
    sayHi() {
        return `${this.#hi()} ${this.name}`;
    }
}
const person1 = new Person('xiaoming');
console.log(person1.sayHi()); // 'hello world xiaoming'
```

图 5-40 ES6 和 ES5 中子类的实现的程序示例

10. 模块的实现

开发大型复杂程序的时候往往需要将程序的功能分别用多个不同的 JavaScript 文件实现，这些不同的 JavaScript 文件就构成了 JavaScript 程序的模块。不同模块间的程序联合工作需要用到 import 和 export 关键字。在 ES5 中只能使用额外的工具来辅助实现该功

```
{
  "name": "babel",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "devDependencies": {
    "@babel/cli": "^7.0.0",
    "@babel/core": "^7.0.0",
    "@babel/plugin-proposal-private-methods": "^7.8.3"
  },
  "license": "ISC"
}
```

图 5-41　使用 babel 的 package. json 文件示例

```
./node_modules/.bin/babel --plugins @babel/plugin-proposal-private-methods script.js -o result.js
```

图 5-42　使用 babel 的命令示例

能。在 ES6 中将模块变成了 JavaScript 语言本身的一个特性。在浏览器中 ES6 模块的具体使用方法如图 5-43～图 5-45 所示。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>module</title>
</head>
<body>
    <script type="module" src="module1.js"></script>
    <script type="module" src="module2.js"></script>
</body>
</html>
```

图 5-43　ES6 中模块在浏览器的使用的程序示例 1

```
export const name = 'xiaoming';
export function sayHi() {
    return 'hello world';
};
```

图 5-44　ES6 中模块在浏览器的使用的程序示例 2

```
import {name, sayHi} from './module1.js';

console.log(`${sayHi()} ${name}`);
```

图 5-45　ES6 中模块在浏览器的使用的程序示例 3

11. 可计算的对象属性

在 ES6 中,可以使用表达式作为对象属性。这时需要使用[]来计算表达式的值。图 5-46 给出了通过表达式来计算对象属性的程序示例。

```
/* Computed property keys in ES6 */
let name = 'student';
let getAge = () => 'age';

let obj = {
    [name]: 'xiaoming',
    [getAge()]: 21
}

console.log(obj.student); // 'xiaoming'
console.log(obj[name]); // 'xiaoming'
console.log(obj.age); // 21
console.log(obj[getAge()]); // 21
```

图 5-46 ES6 中表达式作为对象属性的程序示例 3

12. for…of 语句

ES6 中提供的 for…of 语句可以用来代替 for…in 语句和数组的 forEach 方法。for…of 语句可以用来遍历各种数据结构,例如,数组、字符串、Maps、Sets 等。在某些情况下,for…of 比 for…in 和 forEach 具有更多的优点。图 5-47 和图 5-48 给出了 for…of 遍历部分不同数据结构的程序示例。

```
/* for ... of statement */

/* Iterating over an Array */
/* The results are 1, 2, 3 */
const iterable = [1, 3, 3];
for(const value of iterable) {
    console.log(value)
}

/* Iterating over a String */
/* The results are 'h', 'e', 'l', 'l', 'o' */
const iterable = 'hello';
for(const value of iterable) {
    console.log(value)
}

/* Iterating over a Set */
/* The results are 1, 2, 3 */
const iterable = new Set([1,2,3]);
for(const value of iterable) {
    console.log(value)
}
```

图 5-47 ES6 中 for…of 语句的程序示例 1

```
/* Iterating over a Map */
/* The results are 1, 2, 3 */
const iterable = new Map([['a', 1], ['b', 2], ['c', 3]]);
for(const value of iterable) {
    console.log(value)
}
/* Iterating over arguments object */
/* The results are 'a', 'b', 'c' */
(function() {
    for (let value of arguments) {
        console.log(value);
    }
})('a', 'b', 'c');
/* Iterating over generator function */
/* The results are 0, 1, 2 */
function* iterable() {
    for(let i = 0; i < 3; i++) {
        yield i;
    }
}
for(let value of iterable()) {
    console.log(value);
}
```

图 5-48　ES6 中 for…of 语句的程序示例 2

13. Getters 和 Setters

Getters 和 Setters 可以让我们在定义对象的属性时增加额外的数据处理功能。Getters 和 Setters 既可以用在类的声明中，也可以用在对象的声明中。图 5-49 给出了在类中使用 Getters 和 Setters 的程序示例。图 5-50 给出了在对象中使用 Getters 和 Setters 的程序示例。

```
/* Getters & Setters in Class */
class Student {
    constructor(age) {
        this.age = age;
    }
    get score() {
        return this.age * 2 >100 ? 100 : this.age * 2;
    }
    set score(num) {
        this.age = num;
    }
}
const stu1 = new Student(21);
console.log(stu1.score); // 42
stu1.score = 70;
console.log(stu1.score); // 100
```

图 5-49　在类中使用 Getters 和 Setters 的程序示例

```
/* Getters & Setters in Object */
const student = {
    age: 21,
    get score() {
        return this.age * 2 >100 ? 100 : this.age * 2;
    },
    set score(num) {
        this.age = num;
    }
}
console.log(student.score); // 42
student.score = 70;
console.log(student.score); // 100
```

图 5-50 在对象中使用 Getters 和 Setters 的程序示例

5.4.2 ES6 程序示例

本节展示如何使用 ES6 的知识来完成一个程序示例。

任务：请对比 ES5 和 ES6 在实现(模拟实现)类的静态方法(属性)、实例方法(属性)、私有方法(属性)的区别。

程序实现：静态方法(属性)的代码如图 5-51 所示，实例方法(属性)的代码如图 5-52 所示，私有方法(属性)的代码如图 5-53 所示。

```
/* Static property and methods in ES5 */
function Student() {
}
Student.grade = 'Three';
Student.getAge = function() {
    return 'between 8 and 10';
}
console.log(Student.grade); // 'Three'
console.log(Student.getAge()); // 'between 8 and 10'

/* Static property and methods in ES6 */
class Student {
    static grade = 'Three';
    static getAge() {
        return 'between 8 and 10';
    }
}
console.log(Student.grade); // 'Three'
console.log(Student.getAge()); // 'between 8 and 10'
```

图 5-51 ES5 和 ES6 静态方法和属性的代码实现图

```
/* Instance property and methods in ES5 */
function Student(name, age) {
    this.name = name;
    this.age = age;
    this.sayHi = function() {
        return 'I am ' + this.name + ' and I am ' + this.age + ' years old';
    }
}
var stu1 = new Student('xiaoming', 21);
console.log(stu1.name); // 'xiaoming'
console.log(stu1.sayHi()); // 'I am xiaoming and I am 21 years old'

/* Instance property and methods in ES6 */
class Student {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    sayHi() {
        return 'I am ' + this.name + ' and I am ' + this.age + ' years old';
    }
}
var stu1 = new Student('xiaoming', 21);
console.log(stu1.name); // 'xiaoming'
console.log(stu1.sayHi()); // 'I am xiaoming and I am 21 years old'
```

图 5-52　ES5 和 ES6 实例方法和属性的代码实现图

```
/* Private property and methods in ES5 */
function Student(name) {
    this.name = name;
    var greetings = 'hello'; // private property
    var concating = function(arr) {
        return arr.join(' ');
    } // private methods
    this.sayHi = function() {
        return concating([greetings, 'I am', this.name]);
    }
}
var stu1 = new Student('xiaoming');
console.log(stu1.sayHi()); // 'hello I am xiaoming'

/* Private property and methods in ES6 */
class Student {
    constructor(name) {
        this.name = name;
    }
    #greetings = 'hello';
    #concating = arr => {
        return arr.join(' ');
    }
    sayHi() {
        return this.#concating([this.#greetings, 'I am', this.name]);
    }
}
var stu1 = new Student('xiaoming');
console.log(stu1.sayHi()); // 'hello I am xiaoming'
```

图 5-53　ES5 和 ES6 私有方法和属性的代码实现图

本章小结

JavaScript 是 Web 开发的核心支柱。本章重点讲解了 JavaScript 中的基本语法、如何使用 JavaScript 进行面向对象编程以及如何使用 JavaScript 进行函数式编程。本章还介绍了 ES6 中的各种主要新特性，包括函数的默认参数、字符串模板、多行字符串、解构赋值、改进的对象表达式、箭头函数、Promise、let 与 const、类、模块、可计算的对象属性、for…of 语句、Getters 和 Setters。在实例部分，完成了 ES5 和 ES6 在实现类的各种不同特性方面的代码实现。有了本章的这些基础，读者就可以在 Web 上用 JavaScript 进行编程，或者进行其他需要进行 JavaScript 编程的工作。

本章主要介绍了如下内容：

(1) JavaScript 的基本语法。

(2) JavaScript 中面向对象编程的概念和编程范式。

(3) JavaScript 中函数式编程的概念和编程范式。

(4) ES6 中各种主要的新特性的基本概念和用法。

(5) ES5 和 ES6 在实现类的各种不同特性方面的代码实现。

(6) 用 JavaScript 解决实际问题的方法。

思考题

(1) JavaScript 中有哪些数据类型？

(2) JavaScript 中的操作符“==”和“===”有什么区别？

(3) 请简述 JavaScript 中的函数式编程和面向对象编程的基本概念。

(4) 请解释 ES6 中字符串模版的使用方法。

(5) 请解释 ES6 中箭头函数和 ES5 中函数的异同。

(6) 请对比 ES5 和 ES6 在实现(模拟实现)类的静态方法(属性)、实例方法(属性)、私有方法(属性)的区别。