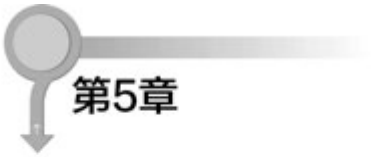

对象和数组

面向对象是当今软件编程中最为主流的软件设计和编码的解决方案，数组是程序中面向数据处理的应用最为广泛的数据结构之一。本章将阐述面向对象和数组的基本概念，同时介绍在 JavaScript 中定义对象、应用对象的方法，以及定义数组和应用数组的方法和技巧。

【学习目标】

(1) 了解什么是对象，以及关于对象的面向对象编程的概念。

(2) 学习对象的创建和应用，理解对象与对象原型的关系。

(3) 理解数组的概念，学习数组的创建和遍历方法。

(4) 学习实现数组迭代的各种高阶方法，掌握运用高阶方法完成数组的复杂操作。

5.1 面向对象编程

5.1.1 面向对象编程概述

面向对象编程(Object-Oriented Programming，OOP)是一种编程范式或编程风格，它使用“对象”来设计软件。在面向对象编程中，程序被组织成一组相互协作的对象，每个对象都包含数据(属性)和代码(方法)，用于操作这些数据。这种范式强调将现实世界或问题域中的事物抽象为对象，并通过这些对象之间的交互来解决问题。例如，要设计一个图书借阅的软件，通过场景分析得到完成一次图书借阅所涉及的包括读者、管理员、图书、借书卡 4 个对象，读者申请并办理借书卡，读者提交需借阅图书给管理员，管理员在借书卡上填写借阅信息，如图 5-1 和图 5-2 所示。

其中，图 5-1 展示了图书借阅过程中 4 个对象的关系，图 5-2 则具体描述了借书卡对象的属性(卡号、借阅记录和持卡人)，以及它包含的借书卡方法(记录借阅图书)。

面向对象编程是一种站在宏观管理视角分析和设计复杂问题构成的一种编程方法，通过问题的对象化抽象来明确问题的数据和业务处理范围。面向对象是当今软件设计中最为

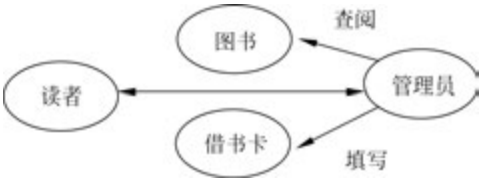

图 5-1 图书借阅软件的对象及其协作关系

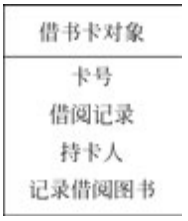

图 5-2 借阅卡对象的属性和方法

主流的编程思想与方法。C++、C#、Java、Python 等热门程序语言属于面向对象的程序语言范围，JavaScript 作为当今主流的脚本语言，也支持面向对象编程，这样就方便与 Java、C#等面向对象的程序语言的协作编程，并实现对象数据的通信。

5.1.2 面向对象编程中的一些概念

要想了解面向对象编程，必须先了解一些面向对象的基本概念，包括对象、类、封装、继承、多态。下述是这些概念的一些说明。

(1) 对象(Object)：对象是类的实例，是现实世界中实体的抽象，如图书对象、管理员对象、借书卡对象、读者对象。每个对象都由状态(数据/属性)和行为(方法)构成。

(2) 类(Class)：类是一个模板或蓝图，它是对象的类型，因为它定义了对象的属性和方法。每个对象必属于一个类(类型)；在面向对象编程中，通常需要先定义类，再用类创建对象，一个类可以创建任意多个同一类型的对象。

(3) 封装(Encapsulation)：封装是面向对象三大特性之一(另两个是继承和多态)。它是对象的设计过程，是将对象的状态信息隐藏在对象内部，不允许外部直接访问对象的内部信息，而通过该对象提供的方法来实现对内部信息的操作和访问。需要注意的是，对象的封装设计是由类来实现的。它通过将对象的内部状态(数据)和实现细节隐藏在公开接口之后，有效地隐藏了对象结构和业务逻辑的复杂性，提高代码的安全性和可维护性。

(4) 继承(Inheritance)：继承也是面向对象三大特性之一。它提供一种使用旧类创建新类的方式，在继承机制中，新创建的类称为子类或派生类，被继承的类称为父类或基类。子类可以继承父类的属性和方法，也可以定义自己的属性和方法。继承是实现代码复用的重要手段之一。

(5) 多态(Polymorphism)：多态也是面向对象特性之一，多态允许同一个方法或者同类的对象在不同的环境和条件下做出不同的响应。在面向对象编程中，多态性通常通过方法的重写(Override)和重载(Overload)来实现。多态性增强了程序的灵活性和可扩展性。

面向对象编程是当今软件开发的主要的程序设计方法，它在实现代码的重用、维护、扩展和提高开发效率方面有着极大的优势，同时面向对象编程将现实世界中的事物抽象为对象，使开发者可以更加直观地理解和设计程序。总之，面向对象编程是一种强大的编程范式，它通过对象、类、封装、继承和多态等核心概念，提供了一种更加直观、灵活和可扩展的方式来设计软件。

5.2 JavaScript 面向对象编程

JavaScript 支持面向对象编程，尽管它采用了一种基于原型的继承机制，与传统的基于类的继承有所不同。在 JavaScript 中，对象可以被视为属性的集合，其中属性是数据(值)或函数(方法)。面向对象编程在 JavaScript 中的核心概念包括：对象、构造函数、原型、继承等。

5.2.1 对象的创建

在 JavaScript 中，常用的 3 种创建对象的方式如下：

(1) 基于{ }语法创建对象；

(2) 使用 Object 类创建对象；

(3) 基于构造函数创建对象。


视频讲解

1. 基于“{ }”语法创建对象

基于“{ }”语法创建对象属于字面量创建对象方式，{ }为对象体的边界，对象的属性使用键值对方式进行定义，如属性名：值。对象体中可以定义对象的成员方法，成员方法与普通函数略有不同，即不需要使用 function 关键字定义。

对象的使用，即对对象的属性和方法的调用主要采用“.”运算符或[ ]来实现，其中对对象属性的调用可以是对象名.属性或对象[“属性名”]，对方法的调用使用对象名.方法( )来调用。

【例 5-1】 基于{ }来创建对象，以及调用对象的 JavaScript 程序。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=
1.0">
        <title>Document</title>
        <script type="text/javascript">

            const time = {
                hour: 21,          //小时
                minute: 34,        //分
                second: 17,        //秒
                setTime(hour,minute,second) {
                    this.hour = hour;
                    this.minute = minute;
                    this.second = second;
                },
                showTime( ){
                    return this.hour + ":" + this.minute + ":" + this.second;
                }
            };
            document.write("<p>" + time.showTime( ) + "</p>");
```

```
23              time.setTime(8,15,59);
24              document.write("<p>" + time.showTime( ) + "</p>");
25               document.write("<p>" + time.hour + "时" + time["minute"] + "分" + time.
    second + "秒</p>");
26
27          </script>
28      </head>
29      <body>
30      </body>
31  </html>
```

在例 5-1 中，使用{}创建了一个代表时间的对象，对象包含 hour、minute、second 这 3 个属性，以及设置时间 setTime 和显示时间 showTime 两个函数，并用字面量变量 time 来引用。对象创建后，通过调用 time 变量（实际可以认为就是对象）的函数和成员来实现时间的设置和展示，该程序运行结果如图 5-3 所示。

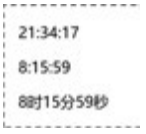

图 5-3 基于{}语法创建对象的运行结果

2. 使用 Object 类创建对象

Object 是 JavaScript 中所有对象的基类模板。因此 JavaScript 提供了一种使用 Object 作为模板来创建对象的机制。即首先创建一个 Object 类型的对象，再在对象中添加对象的属性和函数。


视频讲解

【例 5-2】 使用 Object 创建 time 类的程序示例。

```
01  <!DOCTYPE html>
02  <html lang="en">
03      <head>
04          <meta charset="UTF-8">
05              <meta name="viewport" content="width=device-width, initial-scale=
    1.0">
06          <title>Document</title>
07          <script type="text/javascript">
08
09              var time = new Object( );
10
11              time.hour = 9;
12              time.minute = 30;
13              time.second = 42;
14              time.showTime = function( ){
15                  return time.hour + ":" + time.minute + ":" + time.second;
16              };
17
18              document.write("<p>" + time.showTime( ) + "</p>");
19          </script>
20      </head>
21      <body>
22      </body>
23  </html>
```

例 5-2 的 time 类在结构上与例 5-1 相同，区别仅仅是对象创建方式不同而已。运行

图 5-4　使用 Object 类创建对象的运行结果


视频讲解

例 5-2 的程序代码的结果如图 5-4 所示。

3. 基于构造函数创建对象

学习过面向对象的程序员都知道，每个对象都必须拥有至少一个构造函数，构造函数用于创建对象，且必须使用 new 关键字来调用。构造函数可以拥有参数，通常用于创建对象属性的同时赋予其初始值。

【例 5-3】 定义并使用构造函数创建时间对象的程序示例。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <script type="text/javascript">

            function time(hour=0, minute=0, second=0) {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
                this.showTime = function(){
                return this.hour+":"+this.minute+":"+this.second;
                };
            }
            const t1 = new time(9,30,42);
            const t2 = new time(17,56,32);

            document.write("<p>"+t1.hour+":"+t1.minute+":"+t1.second+"</p>");
            document.write("<p>"+t2.showTime()+"</p>");
        </script>
    </head>
    <body>
    </body>
</html>
```

例 5-3 中定义了一个名为 time 的对象构造函数，通过 3 个参数 hour、minute、second 来为 time 对象的 3 个属性 hour、minute、second 赋予初值。创建 time 对象则使用 new time( )来调用构造函数创建所需的对象。该程序的运行结果如图 5-5 所示。

图 5-5　基于构造函数创建对象的运行结果

5.2.2　对象的 prototype 属性

在 JavaScript 中，每个对象都有一个内部属性(prototype)，prototype 属性意思为原型属性，它是对象的一个非常重要的概念，prototype 为对象提供了继承和共享属性的机制。每个 JavaScript 对象都有一个与之关联的原型对象，通过原型对象，可以实现属性和方法的共享，从而减少内存占用。

原型也是一个对象，它是其他对象的模板或蓝图。当一个对象试图访问一个属性或方法时，如果在该对象自身没有找到，JavaScript 会沿着原型链向上查找，直到找到对应的属性或方法，或者到达原型链的顶端 null 为止。

原型自身也有 prototype 属性，它指向当前的原型对象，所有的对象正是通过这样的一条原型链来实现继承。原型链的最顶端是 Object 对象。对象的 prototype 属性的继承和共享机制如图 5-6 所示。

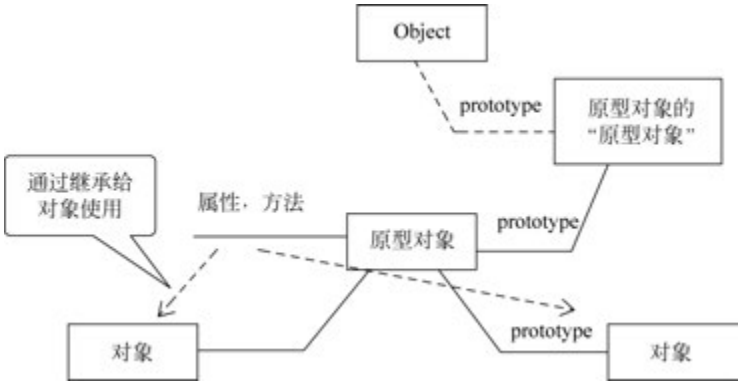

图 5-6 对象的 prototype 属性的继承和共享机制

5.2.1 节中介绍了基于构造函数创建对象的机制，与使用 Object 类创建对象的机制不同，基于构造函数创建的对象原则上是不能在对象创建后再次添加新的属性和方法的。但凡事都有例外，JavaScript 虽规定不能在对象创建后修改对象，但是却允许在对象的原型中添加。当对象的 prototype 属性指向的原型对象添加方法后，对象通过继承也就拥有了该方法。

【例 5-4】 使用对象的 prototype 属性指向的原型对象来修改 time 类的结构。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <script type="text/javascript">

            function time(hour=0, minute=0, second=0) {
                this.hour = hour;
                this.minute = minute;
                this.second = second;
                this.showTime = function( ){
                return this.hour+":"+this.minute+":"+this.second;
                };
            }


```

```
            time.prototype.showTime12 = function( ) {
                if(this.hour > 12){
                    _hour = this.hour - 12;
              return _hour + ":" + this.minute + ":" + this.second + " PM";
                }else{
              return this.hour + ":" + this.minute + ":" + this.second + " AM";
                }
            };
            const t1 = new time(17,56,32);

            document.write("<p>" + t1.showTime( ) + "</p>")
            document.write("<p>" + t1.showTime12( ) + "</p>")
        </script>
    </head>
    <body>
    </body>
</html>
```

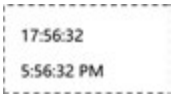

图 5-7　通过原型实现对象方法添加后的程序运行结果

在例 5-4 中，程序使用 time.prototype.获得了 time 类的原型，并通过向原型添加 showTime12( )函数来增加属于 time 类的成员方法。之后当 time 类对象被创建后，可以通过 time 类对象 t1 来调用 showTime12( )来按照 12 小时制显示时间。程序运行结果如图 5-7 所示。

使用对象原型修改对象类型的机制，不仅可以发生在对象创建前，在对象已经创建后通过 prototype 指向的原型进行的对象方法的修改也能被识别和生效。例如，接例 5-4，在已经创建了 t1 对象后，再次通过 prototype 对 showTime12( )方法进行修改为中文的时分秒的显示方式，则 t1 对象调用 showTime12( )方法也将发生改变。

【例 5-5】 接例 5-4 在对象创建后再次通过对象原型进行方法修改的示例。

```
...
const t1 = new time(17,56,32);

document.write("<p>" + t1.showTime( ) + "</p>")
document.write("<p>" + t1.showTime12( ) + "</p>")

time.prototype.showTime12 = function( ) {
  if(this.hour > 12){
     _hour = this.hour - 12;
     return _hour + "时" + this.minute + "分" + this.second + "秒 下午";
  }else{
     return this.hour + "时" + this.minute + "分" + this.second + "秒 上午";
  }
};

document.write("<p>" + t1.showTime12( ) + "</p>")
</script>
...
```

例 5-5 的程序运行结果如图 5-8 所示。

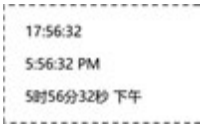

图 5-8 通过原型修改方法后的程序运行结果

5.3 数组

在 JavaScript 中,数组(Array)是一种特殊的对象类型,用于存储元素集合。这些元素可以是任何类型,包括数字、字符串、对象等。数组提供了许多内置的方法来操作和管理其元素。

JavaScript 的数组与 Java 的数组在结构上没有多少差别,但在创建方法和操作方法上,相比 Java 要更加简单和丰富。

5.3.1 数组对象的创建

JavaScript 的数组可以使用两种方法进行创建,一种是使用 Array 构造函数进行显式创建,另一种是使用数组字面量进行隐式创建。两种方法在功能上非常相似,但它们在某些细节和性能方面可能存在一些差异,如使用 Array 构造函数创建数组对象可以指定数组长度,而使用数组字面量则不可以,同时在创建时,使用字面量创建数组对象在性能上优于使用 Array 构造函数创建数组对象。以下是这两种方式的数组对象创建。

1. 使用 Array 构造函数创建数组对象


视频讲解

JavaScript 提供了使用 Array 构造函数创建数组对象的 4 种方式,即默认方式创建一个数组,创建一个指定初始化长度的数组,使用初始值创建一个数组,以及使用 Array 类的静态方法 of( )来创建一个数组。

【例 5-6】 使用 Array 构造函数创建数组对象的程序示例。

```
01  //定义一个数组
02  var ary1 = new Array( );
03  ary1[0] = "北京";
04  ary1[1] = "上海";
05  ary1[2] = "广州";
06
07  //定义长度为 10 的数组
08  var ary2 = new Array(10);
09  ary2[0] = 67.2;
10  ary2[1] = 97.6;
11  ary2[2] = 77.1;
12
13  //定义长度为 3 的数组,元素值为"red","yellow","blue"
14  var ary3 = new Array("red","yellow","blue");
15
16  //定义长度为 3 的数组,元素值为"apple","banana","orange"
17  let ary4 = Array.of("apple","banana","orange");
```

```
18
19   document.write("<p>ary1 = " + ary1 + "</p>");
20   document.write("<p>ary2 = " + ary2 + "</p>");
21   document.write("<p>ary3 = " + ary3 + "</p>");
22   document.write("<p>ary4 = " + ary4 + "</p>");
23
```

图 5-9　Array 构造函数创建数组对象的运行结果

例 5-6 中的程序创建了 4 个数组对象,并为每个数组的元素赋予了值,其运行结果如图 5-9 所示。

数组与变量相比,它可以存储更多值,且只需要使用一个“变量”名,即数组名来定义,这个变量可以理解为一个集合变量,因为该“变量”是由多个拥有相同名称的子变量,即元素按照线性方式组合而成,每个元素又使用一个从 0 开始的有序序号,即下标或索引来进行区分。因此可以理解数组 ary1 是由 ary1[0]、ary1[1]、ary1[2]三个元素组成,ary2 是由 ary2[0]、ary2[1]、ary2[2]……ary2[9]共 10 个元素组成。

与 Java 语言略有不同的是,JavaScript 的数组在数组长度(元素数量)上的约束更加宽松,即使在数组创建时已经指定了长度,但也仍然可以通过向数组追加额外的数据来改变数组的长度,如例 5-7 所示。

【例 5-7】 对超出长度的数组元素赋值以改变数组原有长度的程序示例。

```
01   ary2[10] = "101.5";
02   ary3[3] = "black";
03
04   document.write("<p>ary2 = " + ary2 + "</p>");
05   document.write("<p>ary3 = " + ary3 + "</p>");
```

例 5-7 中,ary2 数组的长度在定义时为 10,但向第 11 个数组元素追加数据后,ary2 的长度就变为 11,同理 ary3 也是通过追加一个新数据后,其长度从 3 变为 4,这个改变可以从两次的输出中看到,如图 5-10 所示。

图 5-10　通过追加新数据改变数组长度的运行结果


视频讲解

2. 使用数组字面量创建数组对象

字面量是一种直接表示数据值的语法形式,无须计算或引用其他变量。字面量可以直接在代码中使用,以表达一个具体的值。数组字面量是用于直接表示一个数组的对象创建方法。使用[ ]括号来表示数组,数组中元素的值在[ ]中定义,用逗号分隔,例 5-8 介绍了使用数组字面量创建数组对象的方法。

【例 5-8】 使用数组字面量创建数组对象的程序示例。

```
01   //创建一个空数组
02   let ary4 = [];
03   //创建一个包含 2 个元素的数组
04   let ary5 = ["男","女"];
```

```
05  //创建一个包含 3 个元素的数组
06  let ary6 = [5,9,7];
07
08  document.write("<p>ary4 = " + ary4 + "</p>");
09  document.write("<p>ary5 = " + ary5 + "</p>");
10  document.write("<p>ary6 = " + ary6 + "</p>");
```

例 5-8 的程序使用字面量[ ]创建了 3 个数组，分别包含了空内容，“男”，“女”和 5，9，7。运行后的数组输出内容如图 5-11 所示。

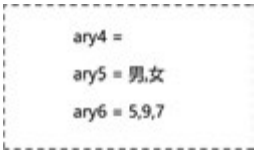

图 5-11 使用数组字面量创建数组对象的运行结果

5.3.2 数组元素的访问

数组是由数组元素的集合构成的，数组元素为数组名[索引]，索引从 0 开始，一个长度为 10 的数组，其数组元素是从[0]到[9]。对数组的操作和访问就是对数组元素的访问，例 5-9 介绍了对数组的访问和修改。

【例 5-9】 访问和修改数组元素的程序示例。

```
01  let ary = [];          //创建一个空数组
02  ary[0] = "北京";       //添加第一个元素
03  ary[1] = "上海";       //添加第二个元素
04  ary[2] = "杭州";       //添加第三个元素
05
06  document.write("<p>" + ary[0] + "," + ary[1] + "," + ary[2] + "</p><br>");
07
08  ary[2] = "成都";       //修改第三个元素的值
09
10  //使用循环遍历(按顺序)输出数组中所有元素的值
11   for(i = 0;i < ary.length;i++){
12       document.write("<p>" + ary[i] + "</p>");
13   }
```

例 5-9 的程序运行结果如图 5-12 所示。

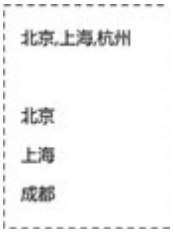

图 5-12 对数组的访问和修改的运行结果

5.3.3 对数组中元素的常见操作

JavaScript 对于数组还提供了非常丰富的方法，这些方法用于提供对数组的一些常见操作，以降低数组的操作和使用难度，以下是一些常见的数组操作和示例代码。

1. 获取数组长度

数组的 length 属性存储了数组的实际长度，可以使用数组名.length 来返回数组的长度。

【例 5-10】 获取数组长度的程序示例。

```
01  let ary = [10, 20, 30];
02  document.write("<p>" + ary.length + "</p>"); //输出 3
```

2. 添加数组元素

数组提供了一组方法：push( )、unshift( )、splice( )来实现在数组末尾，开头和指定位置添加元素，相比直接对单个数组元素的操作，效率和安全性更高。

1）在数组末尾添加

使用 push(Object item)方法，该方法用于向数组末尾追加一个指定的元素，其值为参数 item。

【例 5-11】 在数组末尾添加新元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.push(40);
03  document.write("<p>" + ary + "</p>"); //输出[10,20,30,40]
```

2）在数组开头添加

使用 unshift (Object item)方法，该方法用于向数组开头插入一个指定的元素，其值为参数 item，数组中原有元素向后位移一位。

【例 5-12】 在数组开头添加新元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.unshift(5);
03  document.write("<p>" + ary + "</p>"); //输出[5,10,20,30]
```

3）在数组指定位置添加

使用 splice( )方法用于实现向数组指定位置插入或删除指定数量的元素。splice( )方法中的参数有 3 个，第一个参数是插入元素的位置，如例 15-13 中第 2 行代码中的参数 1 代表在第二个元素位置插入一个值。第二个参数是指定需要删除元素的数量，如果该参数是 0；表示不删除任何一个元素，因此就是一个简单的插入操作，而如果该参数＞0，则表示插入值的后面要删除指定数量的元素。第三个参数则是指定位置要插入的具体数值，如例 5-13 中表示要插入的值为 15。

【例 5-13】 在数组指定位置插入新元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.splice(1, 0, 15);                    // 在索引 1 的位置插入 15
03  document.write("<p>" + ary + "</p>");  //输出[10,15,20,30]
```

3. 删除数组元素

1）删除数组末尾元素

使用 pop( )方法将数组中末尾元素返回，同时该元素从数组中删除。

【例 5-14】 删除数组末尾元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.pop( );
03  document.write("<p>" + ary + "</p>"); //输出[10,20]
```

2）删除数组开头元素

使用 shift( )方法删除数组最开头(第一个)元素，并返回该元素。

【例 5-15】 删除数组开头元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.shift( );
03  document.write("<p>" + ary + "</p>"); //输出[20,30]
```

3）删除数组指定位置元素

splice( )方法可以用于删除元素，这里只需要两个参数：删除元素的索引及元素数量。

【例 5-16】 删除数组指定位置元素的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.splice(1, 1);                          // 从索引 1 开始删除 1 个元素
03  document.write("<p>" + ary + "</p>");      //输出[10,30]
```

4. 数组排序

使用 sort( )方法可以让数组进行正向(从小到大)排序。

【例 5-17】 对数组中的元素进行正向排序的程序示例。

```
01  let ary = [5,9,1,6,4];
02  ary.sort( )                                //正向排序
03  document.write("<p>" + ary + "</p>");  //输出[1,4,5,6,9]
```

5. 数组转字符串

数组的 toString( )方法是将数组的所有元素内容转换为一个字符串。

【例 5-18】 将数组的所有元素转换为一个字符串的程序示例。

```
01  let ary = ["上海","深圳","广州"];
02  document.write("<p>" + ary.toString( ) + "</p>"); //输出 "上海","深圳","广州"
```

5.4 数组的迭代函数与高阶函数

在 JavaScript 中，数组的迭代和高阶函数是处理数组数据的强大工具。迭代函数允许遍历数组的元素并执行一些操作，而高阶函数则通常接收一个或多个函数作为参数，或者返

回一个函数作为结果。以下是一些常用的数组迭代函数和高阶函数。

1. forEach 函数

forEach 函数是数组用于按顺序遍历元素的迭代函数。该函数在遍历数组中的每一个元素的同时,还可为每个元素提供针对元素数据进行特殊处理的回调函数。其回调函数的参数 currentValue 代表当前处理的数组元素; index 是一个可选项,代表当前元素的索引; array 也是一个可选项,为调用 forEach( )的数组本身; thisArg 也是可选项,为执行回调时用作 this 的值。

【语法】

```
array.forEach(callback(currentValue, index, array), thisArg)
```

【例 5-19】 遍历数组的每个元素,然后调用 console.log( )函数输出每个元素的值的程序示例。

```
01  let ary = [10, 20, 30];
02  ary.forEach(item => console.log(item))      //输出 10,20,30
```

【例 5-20】 遍历数组的每个元素,然后调用自定义函数对每个元素的值进行累加,最后输出结果,结果如图 5-13 所示。

```
01  var summary = 0;
02
03      let ary = [10, 20, 30];
04
05      ary.forEach(function(item){
06          summary = summary + item;        //累加元素的值
07          console.log("summary = " + summary);
08      });
09
10  document.write("<p>" + summary + "</p>");
```

图 5-13 对数组进行迭代后计算累加值的运行结果

2. map 函数

map 函数用于创建一个新数组,其结果是通过遍历数组中的每个元素,且调用一个提供的函数后的返回值。

【语法】

```
array.map(callback(currentValue, index, array), thisArg)
```

【例 5-21】 遍历数组,将每个元素乘 2 后再生成一个新数组的程序示例。

```
let ary = [1, 2, 3];
    newary = ary.map(number => number * 2);
document.write("<p>" + newary + "</p>"); // [2, 4, 6]
```

【例 5-22】 遍历数组，调用自定义函数将数组元素中的省名后加入“省”字后构造一个新数组的程序示例。

```
let ary = ["云南","贵州","四川"];

    ary3 = ary.map(function(item){
        return item + "省";
    })
document.write("<p>" + ary3 + "</p>"); // 输出["云南省","贵州省","四川省"]
```

3. filter 函数

filter 函数用于返回一个新数组，新数组包含通过所提供函数进行筛选或处理后的所有元素。

【语法】

```
array.filter(callback(currentValue, index, array), thisArg)
```

【例 5-23】 使用 filter 函数进行数组元素值处理的程序示例。

```
let ary = [1, 2, 3, 4, 5, 6, 7];
//返回 ary 数组中的所有偶数
evens = ary.filter(number => number % 2 === 0);
document.write("<p>" + evens + "</p>");   // [2, 4, 6]
```

该示例结果中新数组 evens 的元素有 3 个，元素值为 2、4、6，因为 ary 数组中只有 2、4、6 通过了筛选处理。

4. reduce 函数

该函数将一个数组中的元素进行累加，并将累加结果汇总为单个的返回值。其回调函数的 accumulator 为累加器，包含上一次累加的值；currentValue 为当前遍历的数组元素；index 为可选项，代表指定一个数组开始遍历的索引；array 也是一个可选项，代表调用 reduce( )的数组本身。

【语法】

```
array.reduce(callback(accumulator, currentValue, index, array), initialValue)
```

【例 5-24】 使用 reduce 函数进行所有数组元素值求和处理的程序示例。

```
let ary = [1, 2, 3, 4, 5];
sum = ary.reduce((accumulator, currentValue) => accumulator + currentValue, 0);
document.write("<p>" + sum + "</p>"); // 15
```

该程序运行返回结果为15,其中accumulator代表累加器变量,currentValue代表数组遍历到的当前值。数组从元素下标0开始遍历,数组遍历完成后,所有元素的值进行了累加,结果为15。

5. every函数

every函数用于测试数组的所有元素是否都能通过指定函数的测试。如果所有元素都能通过,返回true;只要有一个无法通过就返回false。其回调函数的参数currentValue代表当前处理的数组元素;index是一个可选项,代表当前元素的索引;array也是一个可选项,为调用every( )的数组本身。

【语法】

```
array.every(callback(currentValue, index, array), thisArg)
```

【例5-25】 使用every函数测试数组ary中所有元素是否都大于3的程序示例。

```
01  let ary = [1, 2, 3, 4, 5];
02  result = ary.every(item => item > 3);
03  document.write("<p>" + result + "</p>");   // false
```

该程序运行结果返回false,因为数组元素值1、2、3不满足测试。

6. some函数

some函数用于测试数组中是否有某些元素能通过指定函数的测试,如果有这样的元素就返回true,没有返回false。其回调函数的参数与forEach( )的相同。

【语法】

```
array.some(callback(currentValue, index, array), thisArg)
```

【例5-26】 使用some函数测试数组ary中是否存在元素大于3的程序示例。

```
01  let ary = [1, 2, 3, 4, 5];
02  result = ary.some(item => item > 3);
03  document.write("<p>" + result + "</p>"); // true
```

该程序运行结果返回true,因为至少有4、5都大于3。

7. find函数

返回数组中满足提供的测试函数的第一个元素的值。否则返回undefined。

【语法】

```
array.find(callback(currentValue, index, array), thisArg)
```

【例5-27】 使用find函数查找第一个大于数字3的元素的值。

```
01  let ary = [1, 2, 3, 4, 5];
02  value = ary.find(item => item > 3);
03  document.write("<p>" + value + "</p>"); // 4
```

按照数组值的查找，第一个大于 3 的元素的值是 4。因此程序运行结果为 4。

8. findIndex 函数

findIndex 函数用于返回数组中满足提供的测试函数的第一个元素的索引。否则返回 −1。

【语法】

```
array.findIndex(callback(currentValue, index, array), thisArg)
```

【例 5-28】 使用 findIndex 函数查找第一个大于数字 3 的元素的索引。

```
01  let ary = [1, 2, 3, 4, 5];
02  index = ary.findIndex(item => item > 3);
03  document.write("<p>" + index + "</p>"); // 3
```

该程序运行结果返回 3，因为第一个大于数字 3 的元素是 4，该元素索引为 3。

5.5 本章小结

对象和数组是 JavaScript 中两个比较重要的数据类型，对象属于面向对象范畴。

对象可以表示一个复杂的数据类型，存储构成类型的多个数据。JavaScript 的对象可以使用通过“{ }”语法、Object 类及构造函数来创建对象。在 JavaScript 的对象中，有一个特殊的 prototype 属性，它指向对象的原型，可以通过 prototype 链来查找对象在继承链中的成员，也可以通过 prototype 向原型中添加对象所需的成员。

数组也是一个复杂的数据类型，它表示和存储一个序列的数据集合，数组是一个集合容器，它由众多叫作数组元素的变量按照线性顺序排列组合而成。每个元素都有着唯一序号，序号从 0 开始，依次递增，该序号又叫作元素的下标或索引。所有元素都使用数组名作为它们的唯一名称，元素使用“数组名[索引]”进行访问。相比 Java 语言，JavaScript 为数组提供了大量操作函数，以及实现数组迭代操作的高阶函数，这些函数不仅降低了数组的操作难度，更增强了数组的操作能力。

5.6 课后练习

一、选择题（单选和多选均有）

1. 下列哪个选项是创建 JavaScript 对象的有效方法？（　　）

   A. var obj＝{}；

   B. var obj＝new Object( )；

   C. var obj＝new function( ){...}

   D. 以上都是

2. 如何访问对象 person 中的 name 属性？（　　）

   A. person["name"]　　B. person. name

   C. person(name)　　D. person{name}

3. 当你尝试访问一个对象中不存在的属性时，返回的结果是(　　)。

A. null　　B. undefined　　C. NaN　　D. 抛出错误

4. 下列哪个选项是创建 JavaScript 数组的有效方法？(　　)

A. var arr=[];　　B. var arr=new Array( );

C. var arr=Array. of(1, 2, 3);　　D. 以上都是

5. 下列哪个方法不会改变原数组？(　　)

A. push( )　　B. pop( )　　C. map( )　　D. shift( )

二、判断题

1. 对象字面量是创建对象的一种简洁方式，它允许我们在{}中定义对象的属性和方法。(　　)

2. JavaScript 中的每个对象都是从 Object. prototype 继承属性和方法的。(　　)

3. JavaScript 中的数组索引是从 0 开始的。(　　)

4. 使用 push( )方法可以向数组的末尾添加一个或多个元素，并返回新的数组长度。(　　)

5. forEach( )方法会创建一个新数组。(　　)

三、编程题

1. 定义一个学生对象，对象有 name，age，agend 共 3 个属性，为对象赋予“王海”，18，“男”的初始值，另外为对象定义一个成员方法 introSelf( )，用于输出对象的属性信息。调用该方法显示该学生对象的信息。

2. 定义一个数组，存储 10 个学生的某门课程的考试成绩，成绩为 84、62、79、53、38、92、90、68、72、88。编写程序将大于或等于 60 分的成绩放到一个新数组中，把小于 60 分的成绩放到另一个数组中。并按照从小到大的顺序进行排序，最后输出这 3 个数组的值。