第 3 章

JavaScript面向对象程序设计

JavaScript 脚本是面向对象的编程语言，它可以将属性和代码集成在一起，定义为类，从而使程序设计更加简单、规范、有条理。通过对象的访问可大大简化 JavaScript 程序的设计，并提供直观、模块化的方式进行脚本程序开发。本章主要介绍 JavaScript 的面向对象编程思想以及有关对象的基本概念，并引导读者创建和使用自定义的类和对象。


视频讲解

3.1 面向对象程序设计思想简介

3.1.1 什么是对象

对象是客观世界存在的人、事和物体等实体。现实生活中存在很多的对象，如猫、汽车等。不难发现它们有两个共同特征：状态和行为。例如，猫有自己的状态（名字、颜色、饥饿与否等）和行为（爬树、抓老鼠等）；汽车也有自己的状态（挡位、速度等）和行为（刹车、加速、减速、改变挡位等）。若以自然人为例，构造一个对象，可以用图 3-1 表示，其中，属性表示对象状态，动作（方法）表示对象行为。

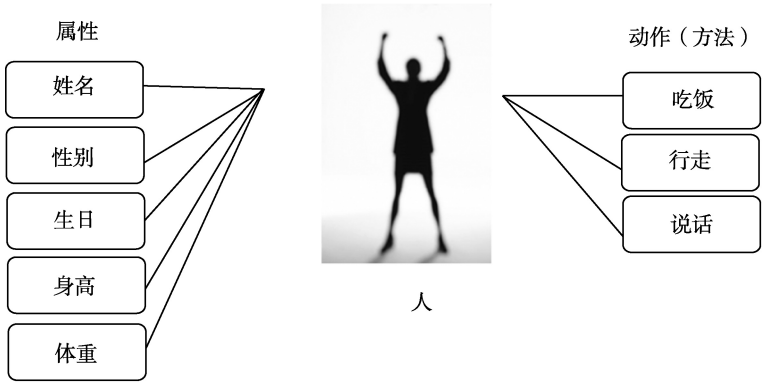

图 3-1　以自然人构造的对象

综上所述，凡是能够提取一定度量数据并能通过某种途径对度量数据实施操作的客观存在都可以构成一个对象，且用属性来描述对象的状态，使用方法和事件来处理对象的各种行为。下面介绍一些概念。

（1）对象（Object）：面向对象程序设计思想可以将一组数据和与这组数据有关的操作组装在一起，形成一个实体，这个实体就是对象。

（2）属性：用来描述对象的状态。通过定义属性值，可以改变对象的状态。如图 3-1 中，可以定义 height 表示该自然人身高，字符串 HungryOrNot 来表示该自然人饥饿的状态，HungryOrNot 成为自然人的某个属性。

（3）方法：也称为成员函数，是指对象上的操作。如图 3-1 中，可以定义方法 Eat( )处理自然人很饿的情况，Eat( )成为自然人的某个方法。

（4）事件：由于对象行为的复杂性，对象的某些行为需要用户根据实际情况来编写处理该行为的代码，该代码称为事件。在图 3-1 中，可以定义事件 DrinkBeforeEat( )处理自然人又饿又很渴需要先喝水后进食的情况。

3.1.2 面向对象编程

面向对象编程是一种计算机编程架构，具有三个最基本的特点：封装、重用性（继承）、多态。面向对象编程主要包含以下重要的概念。

（1）类（class）：具有相同或相似性质的对象的抽象就是类。因此，对象的抽象是类，类的实例化就是对象。例如，如果人类是一个类，则一个具体的人就是一个对象。

（2）封装：将数据和操作捆绑在一起，定义一个新类的过程就是封装。

（3）继承：继承描述了类之间的关系。在这种关系中，一个类共享了一个或多个其他类定义的属性和行为。子类可以对基类的行为进行扩展、覆盖、重定义。如果人类是一个类，则可以定义一个子类“男人”。“男人”可以继承人类的属性（例如姓名、身高、年龄等）和方法（即动作。例如，吃饭和走路），在子类中就无须重复定义了。

（4）多态：从同一个类中继承得到的子类也具有多态性，即相同的函数名在不同子类中有不同的实现。就如同子女会从父母那里继承到人类共有的特性，而子女也具有自己的特性。

实际上，JavaScript 语言是通过一种叫作原型（prototype）的方式来实现面向对象编程的。下面讨论基于类的（class-based）面向对象和基于原型的（prototype-based）面向对象这两种方式在构造客观世界的方式上的差别。

在基于类的面向对象方式中，对象（object）依靠类（class）来产生。而在基于原型的面向对象方式中，对象（object）则是依靠构造函数（constructor）利用原型（prototype）构造出来的。

举个客观世界的例子来说明两种方式的差异。例如，工厂造一辆车，一种基于类的面向对象方式，工人必须参照一张工程图纸，设计规定这辆车应该如何制造。这里的工程图纸就好比是编程语言中的类（class），而车就是按照这个类（class）制造出来的；另一种基于原型的面向对象方式，工人和机器（相当于 constructor）利用各种零部件如发动机、轮胎、方向盘

（相当于 prototype 的各个属性）将汽车构造出来。

3.2　JavaScript 类的定义和实例化

严格地说，JavaScript 是基于对象的编程语言，而不是面向对象的编程语言。在面向对象的编程语言中（如 Java、C++、C#、PHP 等），声明一个类使用 class 关键字。

例如：

```
public class Person
{
}
```

但是在 JavaScript 中，没有声明类的关键字，也没有办法对类的访问权限进行控制。JavaScript 使用函数来定义类。注意 ES6 版本开始提供 class 关键字，详见 3.6.4 节。

3.2.1　类的定义

类定义的语法：

```
function className(){
    //具体操作
}
```

例如，定义一个 Person 类：

```
function Person() {
   this.name="张三";           //定义一个属性 name
   this.sex="男";              //定义一个属性 sex
   this.say=function(){       //定义一个方法 say()
        console.log("我的名字是 " + this.name + " ，性别是 " + this.sex + "。");
   }
}
```

说明：this 关键字是指当前的对象。

3.2.2　创建对象（类的实例化）

创建对象的过程也是类实例化的过程。

在 JavaScript 中，创建对象（即类的实例化）使用 new 关键字。

创建对象语法：

```
new className();
```

将上面的 Person 类实例化：

```
var zhangsan=new Person();
zhangsan.say();
```

运行代码，输出如下内容。

```
大家好，我的名字是张三，性别是男。
```

定义类时可以设置参数，创建对象时也可以传递相应的参数。

下面将 Person 类重新定义。

```
function Person(name,sex) {
    this.name=name;        //定义一个属性 name
    this.sex=sex;          //定义一个属性 sex
    this.say=function(){  //定义一个方法 say()
         console.log("大家好，我的名字是 " + this.name + " ，性别是 " + this.sex);
    }
}
var zhangsan=new Person("小丽","女");
zhangsan.say();
```

运行代码，输出如下内容。

```
大家好，我的名字是小丽，性别是女。
```

当调用该构造函数时，浏览器给新的对象 zhangsan 分配内存，并隐性地将对象传递给函数。this 操作符是指向新对象引用，用于操作这个新对象。下面的句子：

```
this.name=name;    //赋值右侧是函数参数传递过来的name
```

使用作为函数参数传递过来的 name 值在构造函数中给该对象 zhangsan 的 name 属性赋值。对象实例的 name 属性被定义和赋值后，就可以访问该对象实例的 name 属性。

3.2.3 通过对象直接初始化创建对象

通过对象直接初始化来创建对象，与定义对象的构造函数方法不同的是，该方法不需要 new 生成此对象的实例，改写 zhangsan 对象：

```
//直接初始化对象
var zhangsan={
    name:"张三",
    sex:"男",
    say:function (){//定义对象的方法
        console.log("大家好，我的名字是 " + this.name + " ，性别是 " + this.sex);}
}
zhangsan.say();
```

可以通过对象直接初始化创建对象是一个“名字/值”对列表，每个“名字/值”对之间用逗号分隔，最后用一个大括号括起来。“名字/值”对表示对象的一个属性或方法，名字和值之

间用冒号分隔。

上面的 zhangsan 对象，也可以这样来创建：

```
var zhangsan={}
zhangsan.name = "张三";
zhangsan.sex = "男";
zhangsan.say = function(){ return "嗨！大家好，我来了。";  }
```

该方法在只需生成一个对象实例并进行相关操作的情况下使用时，代码紧凑，编程效率高，但致命的是，若要生成若干个对象实例，就必须为生成每个对象实例重复相同的代码结构，代码的重用性比较差，不符合面向对象的编程思路，应尽量避免使用该方法创建自定义对象。

3.3　JavaScript 访问和添加对象的属性和方法

属性是一个变量，用来表示一个对象的特征，如颜色、大小、重量等；方法是一个函数，用来表示对象的操作，如奔跑、呼吸、跳跃等。

对象的属性和方法统称为对象的成员。

3.3.1　访问对象的属性和方法

在 JavaScript 中，可以使用“.”和“[ ]”来访问对象的属性。

1. 使用“.”来访问对象属性

语法：

```
objectName.propertyName
```

其中，objectName 为对象名称，propertyName 为属性名称。

2. 使用“[ ]”来访问对象属性

语法：

```
objectName[propertyName]
```

其中，objectName 为对象名称，propertyName 为属性名称。

3. 访问对象的方法

在 JavaScript 中，只能使用“.”来访问对象的方法。

语法：

```
objectName.methodName()
```

其中，objectName 为对象名称，methodName() 为函数名称。

【例 3-1】 创建一个 Person 对象并访问其成员。

```
function Person() {
    this.name="张三";                //定义一个属性name
    this.sex="男";                   //定义一个属性sex
    this.age=22;                     //定义一个属性age
    this.say=function(){             //定义一个方法 say()
        return "我的名字是 " + this.name + " ，性别是" + this.sex + "，今年"
          + this.age +"岁!";
    }
}
var zhangsan=new Person();
console.log("姓名: "+zhangsan.name);     //使用“.”来访问对象属性
console.log("性别: "+zhangsan.sex);
console.log("年龄: "+zhangsan["age"]);   //使用“[ ]”来访问对象属性
console.log(zhangsan.say());             //使用“.”来访问对象方法
```

实际项目开发中，一般使用“.”来访问对象属性；但是在某些情况下，使用“[ ]”会方便很多，例如，JavaScript 遍历对象属性和方法。

JavaScript 可使用 for in 语句来遍历对象的属性和方法。for in 语句循环遍历 JavaScript 对象，每循环一次，都会取得对象的一个属性或方法。

语法：

```
for(valueName  in  ObjectName){
    //代码
}
```

其中，valueName 是变量名，保存着属性或方法的名称，每次循环，valueName 的值都会改变。

【例 3-2】 遍历 zhangsan 对象的属性或方法。

```
//直接初始化对象
var zhangsan={}
zhangsan.name = "张三";
zhangsan.sex = "男";
zhangsan.say = function(){
    return "嗨! 大家好，我来了。";
}
var strTem="";  //临时变量
for(value in zhangsan){
    strTem+=value+': '+zhangsan[value]+"\n";
}
console.log(strTem);
```

程序运行结果如图 3-2 所示。

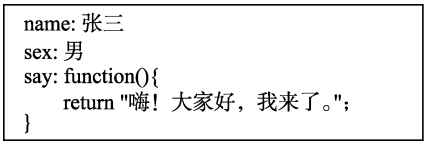

图 3-2　遍历 zhangsan 对象的属性或方法

3.3.2　向对象添加属性和方法

JavaScript 可以在定义类时定义属性和方法，也可以在创建对象以后动态添加属性和方法。动态添加属性和方法在其他面向对象的编程语言（C++、Java 等）中是难以实现的，这是 JavaScript 灵活性的体现。

【例 **3-3**】 用 Person 类创建一个对象，向其添加属性和方法。

```
//定义类
function Person(name,sex){
    this.name=name;                          //定义一个属性 name
    this.sex=sex;                            //定义一个属性 sex
    this.say=function(){                     //定义一个方法 say()
        return "大家好，我的名字是 " + this.name + " ，性别是 " + this.sex + " 。";
    }
}
//创建对象
var zhangsan=new Person("张三","男");
zhangsan.say();
//动态添加属性和方法
zhangsan.tel="029-81892332";                 //动态添加属性tel
zhangsan.run=function(){                     //动态添加方法run
   return  " 我跑得很快！ ";
}
//输出
console.log("姓名：" +zhangsan.name);
console.log("性别：" +zhangsan.sex);
console.log(zhangsan.say());
console.log("电话：" +zhangsan.tel);
console.log(zhangsan.run());
```

可见，JavaScript 动态添加对象实例的属性 tel 和方法 run 的过程十分简单。注意动态添加该属性仅在此对象实例 zhangsan 中才存在，而其他对象实例不存在属性 tel 和方法 run。

例如：

```
var lisi=new Person("李四","男");
console.log(lisi.run());//出现错误 Uncaught TypeError: lisi.run is not a
    function
```

也可以通过原型方法将某个方法动态添加给所有对象实例。

【例 3-4】 通过原型方法将 run()方法动态添加给所有对象实例。

```
//定义类
function Person(name,sex) {
    this.name=name;          //定义一个属性 name
    this.sex=sex;            //定义一个属性 sex
    this.say=function(){     //定义一个方法 say()
       return "大家好，我的名字是 " + this.name + " ，性别是 " + this.sex + " 。";
    }
}
//添加原型属性和原型方法
Person.prototype.grade="2016";
Person.prototype.run=function(name){
   return  name+"我跑得很快！ ";
}
//创建对象
var zhangsan=new Person("张三","男");
zhangsan.tel="029-81892332";
//弹出警告框
console.log("姓名: "+zhangsan.name);
console.log("性别: "+zhangsan.sex);
console.log(zhangsan.say());
console.log("年级: "+zhangsan.grade);
console.log(zhangsan.run(zhangsan.name));   //正确
var lisi=new Person("李四","男");
console.log(lisi.run(lisi.name));           //正确
```

程序调用对象的 prototype 属性给对象添加新属性 grade 和新方法 run()。

```
Person.prototype.grade="2016";
Person.prototype.run=function(name){
   return  name+"我跑得很快！ ";
}
```

原型属性 grade 和原型方法 run()为对象的所有实例 zhangsan、lisi 所共享，用户利用原型添加对象的新属性和新方法后，可通过对象实例来使用原型属性 grade 和原型方法 run()。

3.4 继　　承

继承是指一个对象（如对象 A）的属性和方法来自另一个对象（如对象 B）。此时称对象 A 的类为子类，定义对象 B 的类为父类。JavaScript 中常用的有两种继承方式，原型（prototype）实现继承和构造函数实现继承。

3.4.1　原型实现继承

原型实现继承中为子类指定父类的方法是将父类的实例对象赋值给子类的 prototype 属性，语法如下。

```
A. prototype = new B(…) ;
```

【例 3-5】下面的例子将创建一个 Student 类，它从 Person 继承了原型 prototype 中的所有属性和方法，子类 Student 类比父类多了一个 grade（年级）。

```
function Person(name,age){
    this.name=name;
    this.age=age;
}
Person.prototype.sayHello=function(){
    console.log("使用原型得到Name: "+this.name);
}
var per=new Person("马小倩",21);
per.sayHello();                          //输出：使用原型得到Name:马小倩
function Student(grade){                 //子类Student
    this.grade=grade;
}
Student.prototype=new Person("张海",21);
//将Person定义为Student的父类
Student.prototype.intr=function(){
    console.log("姓名"+this.name+",年级"+this.grade);
//可以访问父类中的name属性
}
var stu=new Student(5);                  //创建Student对象
stu.sayHello();//调用继承的sayHello()方法输出：使用原型得到Name:张海
stu.intr();                             //输出：姓名张海，年级 5
```

通过 Student 的 prototype 属性指向父类 Parent 的实例，使 Student 对象实例能通过原型链访问到父类所定义的属性、方法等，所以 Student 对象 intr()可以访问父类中的 name 属性。

操作符 instanceof 可用于识别正在处理的对象的类型。例如：

```
console.log(stu  instanceof  Person)//true
console.log(stu  instanceof  Student);//true
console.log(per  instanceof  Student);//false
console.log(per  instanceof  Person);//true
```

注意一个子类的实例 stu 既是子类的对象实例，也是父类的对象实例。显然，一个学生既是 Student，也是 Person。

原型实现继承时缺点是创建子类实例时，无法向父类构造函数传递参数。

3.4.2 构造函数实现继承

借用构造函数的方式继承是在子类构造函数中使用 call 调用父类构造函数，从而解决向父类构造函数传递参数问题，并实现继承父类。

【例 3-6】 改写例 3-5，使用构造函数实现继承的实例。

```
function Person(name,age){
   this.name=name;
   this.age=age;
}
Person.prototype.sayHello=function(){
        console.log("使用原型得到Name: "+this.name);
}
var per=new Person("马小倩",21);
per.sayHello(); //输出：使用原型得到Name:马小倩

function Student(name,age,grade){    //子类Student
   Person.call(this,name,age)        //核心处，使用call，在子类中给父类传参数
   this.grade=grade;
}
Student.prototype=new Person();       //将Person定义为Student的父类
Student.prototype.intr=function(){
    console.log("姓名 "+this.name+",年级 "+this.grade);
}
var stu=new Student("张海",21,5);     //创建Student对象
stu.sayHello();    //调用继承的sayHello()方法输出：使用原型得到Name:张海
stu.intr();        //输出：姓名张海，年级 5
```

通过 Person.call(this,name,age)可以实现 Student 继承 Person 的属性 name 和 age 并将其初始化。call 方法的第一个参数为继承类的 this 指针，第二和第三个参数为传给父类 Person 构造函数的参数。

3.4.3 重新定义继承的方法

如果子类重新定义继承的方法，则为原型对象定义与父类同名方法就可以了。例如，在上例中为 Student 重新定义 sayHello()方法：

```
Student.prototype.sayHello=function(){
   console.log("使用子类得到Name: "+this.name);
}
```

从而在 Student 类中重新定义来自父类 Person 的 sayHello()方法。Student 对象 stu 调用时会是子类自己的 sayHello()方法。

```
stu.sayHello();//调用自己的sayHello()方法输出：使用子类得到Name:张海
```

3.5　JavaScript 内置对象

JavaScript 脚本提供丰富的内置对象（内置类），包括同基本数据类型相关的对象（如 String、Boolean、Number）、允许创建用户自定义和组合类型的对象（如 Object、Array）和其他能简化 JavaScript 操作的对象（如 Math、Date、RegExp、Function）。了解这些内置对象是 JavaScript 编程和微信小游戏开发的基础和前提。

3.5.1　JavaScript 的内置对象框架

JavaScript 的内置对象（内置类）框架如图 3-3 所示。

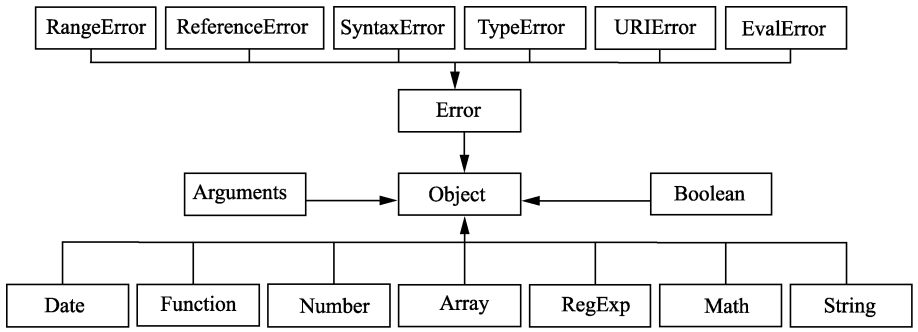

图 3-3　JavaScript 的内置对象框架

JavaScript 内置对象的基本功能如表 3-1 所示。

表 3-1　JavaScript 内置对象的基本功能

内置对象	基本功能
Arguments	用于存储传递给函数的参数
Array	用于定义数组对象
Boolean	布尔值的包装对象，用于将非布尔型的值转换成一个布尔值（True 或 False）
Date	用于定义日期对象
Error	错误对象，用于错误处理。它还派生出下面几个处理错误的子类。 EvalError，处理发生在 eval()中的错误。 SyntaxError，处理语法错误。 RangeError，处理数值超出范围的错误。 ReferenceError，处理引用不当错误。 TypeError，处理不是预期变量类型的错误。 URIError，处理发生在 encodeURI()或 decodeURI()中的错误

续表

内置对象	基本功能
Function	用于表示开发者定义的任何函数
Math	数学对象，用于数学计算
Number	原始数值的包装对象，可以自动地在原始数值和对象之间进行转换
RegExp	用于完成有关正则表达式的操作和功能
String	字符串对象，用于处理字符串

3.5.2 基类 Object

从图 3-3 中可以看到，所有的 JavaScript 对象都继承自 Object 类，后者为前者提供基本的属性（如 prototype 属性等）和方法（如 toString()方法等）。而前者也在这些属性和方法基础上进行扩展，以支持特定的某些操作。基类 Object 的属性和方法如表 3-2 所示。

表 3-2 基类 Object 的属性和方法

属性和方法	具体描述
prototype 属性	对该对象的对象原型的引用。原型是一个对象，其他对象可以通过它实现属性继承。也就是说，可以把原型理解成父类
constructor()方法	构造函数。构造函数是类的一个特殊函数。当创建类的对象实例时系统会自动调用构造函数，通过构造函数对类进行初始化操作
hasOwnProperty(proName)方法	检查对象是否有局部定义的(非继承的)、具有特定名字（proName）的属性
IsPrototypeOf(object)方法	检查对象是否是指定对象的原型
propertyIsEnumerable (proName)方法	返回 Boolean 值，指出所指定的属性（proName）是否为一个对象的一部分以及该属性是否是可列举的。如果 proName 存在于 Object 中且可以使用一个 for…in 循环穷举出来，则返回 true；否则返回 false
toLocaleString()方法	返回对象本地化字符串表示。例如，在应用于 Date 对象时，toLocaleString()方法可以根据本地时间把 Date 对象转换为字符串，并返回结果
toString()方法	返回对象的字符串表示
valueOf()方法	返回对象的原始值（如果存在）

3.5.3 Date 类

Date 类主要提供获取和设置日期和时间的方法，如 getYear()、getMonth()、getDate()等。Date 类的常用方法如表 3-3 所示。

可以使用下面 3 种方法创建 Date 对象。

表 3-3　Date 类的常用方法

方　法	具体描述
getDate	获得当前的日期
getDay	获得当前的天
getHours	获得当前的小时
getMinutes	获得当前的分钟
getMonth	获得当前的月份
getSeconds	获得当前的秒
getTime	获得当前的时间（以 ms 为单位）
getTimeZoneOffset	获得当前的时区偏移信息
getYear	获得当前的年份。请使用 getFullYear() 方法代替
getFullYear()	从 Date 对象以四位数字返回年份
setDate()	设置对象月中的某一天
setFullYear()	设置对象中的年份字段
setHours()	设置对象的小时字段
setMilliseconds()	设置对象的毫秒字段
setMinutes()	设置对象的分钟字段
setMonth()	设置对象的月份字段
setSeconds()	设置对象的秒字段
setTime()	使用毫秒的形式设置对象的各个字段
setYear()	推荐使用 setFullYear()
toDateString()	返回对象的日期部分的字符串表示
toGMTString()	推荐使用 toUTCString()
toLocaleDataString()	根据本地时间格式返回对象的日期部分的字符串表示
toLocaleString()	根据本地时间格式，将对象转换成一个字符串
toLocaleTimeString()	根据本地时间格式，返回对象的时间部分的字符串表示

1. 不带参数

```
var  today = new Date();
```

我们将取得当前的年份，并输出它：

```
var d = new Date()
console.log(d.getFullYear())
```

2. 创建一个指定日期的 Date 对象

```
var theDate = new Date(2017, 9, 1);
```

3. 创建一个指定时间的 Date 对象

```
var  theTime = new Date(2017, 9, 1, 10, 20,30,50);
```

【例 3-7】 计算求 1+2+3+…+100 000 之和所需要的运行时间（ms）。

```
//使用Date对象示例
var t1,t2,htime,i,sum=0;
t1 = new Date();                                          //记录循环前的时间
console.log("循环前的时间是:"+t1.toLocaleString()+":"+t1.
      getMilliseconds());
for(i=1;i<=100000;i++) sum+=i;                            //耗时的循环
t2 = new Date();                                          //记录循环后的时间
console.log("循环后的时间是:"+t2.toLocaleString()+":"+t2.
      getMilliseconds());
htime = t2.getTime() - t1.getTime();
console.log("执行100000次循环用时:"+ htime+"毫秒")
```

运行结果如下。

```
循环前的时间是:2020/2/1 下午12:18:11:408
循环后的时间是:2020/2/1 下午12:18:11:412
执行100000次循环用时:4毫秒
```

3.5.4 String 类

String 是 JavaScript 的字符串类，用于管理和操作字符串数据。可以使用下面两种方法创建 String 对象。

```
MyStr = new String("这是一个测试字符串");  //把参数作为MyStr对象的初始值
MyStr = "这是一个测试字符串";              //直接对String对象赋值字符串
```

String 类只有一个属性 length，用来返回字符串的长度。

【例 3-8】 计算 String 对象的长度。

```
//演示使用String对象的length属性
var MyStr;
MyStr = new String("这是一个测试字符串");
console.log("“" +MyStr+"”的长度为:" + MyStr. length);
```

运行结果如下。

```
“这是一个测试字符串”的长度为:9
```

String 类的常用方法如表 3-4 所示。

表 3-4　String 类的常用方法

方　法	具体描述
charAt (index)	用来返回字符串中指定位置的字符，参数 index 用于指定字符串中某个位置的数字，从 0 开始计数
slice(start,end)	用于返回字符串的片段。start：指定要返回的片断的起始索引，如果是负数，则从字符串的尾部开始算起的位置，–1 指字符串的最后一个字符，–2 指倒数第二个字符，以此类推。end：指定要返回的片断的结尾索引，如果是负数，则从字符串的尾部开始算起的位置
replace(substr,replace)	用于在字符串中用一些字符替换另一些字符，例如 str.replace("China", "Chinese")
concat (str)	用于返回一个 String 对象，该对象包含两个提供的字符串的连接，例如 console.log(str1.concat(str2))
substring (start,stop)	用于返回位于 String 对象中指定位置的子字符串。start：指定要提取子串的第一个字符的位置。stop：指定要提取子串的最后一个字符的位置
blink()	把 HTML<BLINK>标记放置在 String 对象中的文本两端，显示为闪动的文本
bold()	把 HTML <B>标记放置在 String 对象中的文本两端，显示为加粗的文本
italics()	把 HTML <I>标记放置在 String 对象中的文本两端，显示为斜体的文本
lastIndexOf (str)	返回 String 对象中子字符串最后出现的位置
match()	使用正则表达式对象对字符串进行查找，并将结果作为数组返回
search()	返回与正则表达式查找内容匹配的第一个子字符串的位置
small()	将 HTML 的<SMALL>标识添加到 String 对象中的文本两端
substr(start,length)	返回一个从指定位置开始的指定长度的子字符串
toUpperCase()	返回一个字符串，该字符串中的所有字母都被转换为大写字母
toLowerCase()	返回一个字符串，该字符串中的所有字母都被转换为小写字母
split(separator,howmany)	split()方法用于将一个字符串分隔为子字符串，然后将结果作为字符串数组返回。separator：指定分隔符。howmany：指定返回的数组的最大长度

【例 3-9】 演示 slice()方法的例子。

```
var str="Hello world!"
console.log(strslice(6, 11))
```

运行结果如下。

```
world
```

3.5.5　Array 类

Array 数组是在内存中保存一组数据。Array 类的常用方法如表 3-5 所示。

表 3-5 Array 数组的常用方法

方 法	具体描述
length 属性	数组包含的元素的个数
concat()	给数组添加元素(此操作原数组的值不变)
join()	把数组中所有元素转换成字符串连接起来,元素是通过指定的分隔符进行分隔的
pop()	删除并返回数组最后一个元素
push()	把一个元素添加到数组的尾部,返回值为数组的新长度
reverse()	在原数组上颠倒数组中元素的顺序
shift()	删除并返回数组的头部元素
slice()	返回数组的一个子数组，该方法不修改原数组
sort()	从原数组上对数组进行排序
splice()	插入和删除数组元素，该方法会改变原数组
toString()	把数组转换成一个字符串
unshift()	在数组头部插入一个元素，返回值为数组的新长度
length	数组包含的元素的个数
concat()	给数组添加元素(此操作原数组的值不变)

1. Array 数组的创建与使用

方法一：可以使用 new 关键字创建 Array 对象，方法如下。

```
Array对象 = new Array(数组大小)
```

例如，下面的语句可以创建一个由 3 个元素组成的数组 cars。

```
var cars=new Array(3);
```

通过下面的方法访问数组元素。

```
数组元素值 = 数组名[索引]
```

例如：

```
var cars=new Array(3);
cars[0]="Audi";
cars[1]="BMW";
cars[2]="Volvo";
```

方法二：在创建数组对象的时候给元素赋值。

```
var cars=new Array("Audi","BMW","Volvo");
```

方法三：直接赋值。

```
var cars=["Audi","BMW","Volvo"];
```

不过注意创建对象时用的是小括号“( )”，而直接赋值时用的是方括号“[ ]”。

2. 数组遍历

可以使用 for 语句遍历数组的所有索引，然后使用数组名[索引]方法访问每个数组元素。

【例 3-10】 使用 for 语句遍历数组。

```
var MyStr;
MyArr = new Array(3);
MyArr[0] = "中国";
MyArr[1] = "美国";
MyArr[2] = "日本";
for(var i=0;i< MyArr.length; i++)
   console.log(MyArr[i]);
```

运行结果如下。

```
中国
美国
日本
```

另外，for…in 循环也可用来遍历数组的每个元素，改写上例如下。

```
var MyStr;
MyArr = new Array(3);
MyArr[0] = "中国";
MyArr[1] = "美国";
MyArr[2] = "日本";
for(m in MyArr){        //m为数组的key
   console.log(MyArr[m]);}
```

运行结果同上。

【例 3-11】 给定任意一个字符串，使用 for…in 语句来统计字符出现的个数。

```
function charNum(str){
 var charObj=[];                          //空的Array数组
 for(i=0,len=str.length;i<len;i++){
      if(charObj[str[i]]){
         charObj[str[i]]++;
 }else{
         charObj[str[i]]=1;
     }
 }
 var strTem="";                          //临时变量
 for(value in charObj){
          strTem+='"'+value+'"的个数: '+charObj[value]+'\n';
 }
 return strTem;
}
console.log(charNum("Hello"));
```

运行结果如下。

```
"H"的个数：1 "e"的个数：1 "l"的个数：2 "o"的个数：1
```

3．数组排序

使用 Array 类的 sort ()方法可以对数组元素进行排序，sort ()方法返回排序后的数组。

语法如下。

```
arrayObject. sort(sortby)
```

其中，参数 sortby 可选，用于规定排序顺序，sortby 必须是函数。

如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点儿，是按照字符编码的顺序进行排序。

【例 3-12】 对数组排序的例子。

```
var arr = new Array(6);
arr[0] = "George";
arr[1] = "Johney";
arr[2] = "Thomas";
arr[3] = "James";
arr[4] = "Adrew";
arr[5] = "Martin";
console.log("排序前"+ arr + "n");
console.log("排序后"+ arr.sort());
```

运行结果如下。

```
排序前George,Johney,Thomas,James,Adrew,Martin
排序后Adrew,George,James,Johney,Martin,Thomas
```

数组元素为整数时，sort()方法并没有按数值大小真正排序，而是按字符编码顺序排序。下面举例说明。

```
var arr = new Array(6);
arr[0] = 10; arr[1] = 5; arr[2] = 40;
arr[3] = 25; arr[4] = 111; arr[5] = 1;
console.log(arr + "n")
console.log(arr.sort())
```

运行结果如下。

```
10,5,40,25,111,1
1,10,111,25,40,5
```

注意：上面的代码没有按照数值的大小对数字进行排序，而是按字符编码顺序排序。如果想按照其他标准进行排序，就需要提供排序比较函数（参数 sortby），该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 a 和 b，其返回值如下。

（1）若 a<b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。

（2）若 a=b，则返回 0。

（3）若 a>b，则返回一个大于 0 的值。

对例 3-12 增加一个排序比较函数 sortNumber(a, b)，代码如下。

```
function sortNumber(a, b)    //排序比较函数
{
   return a - b;
}
var arr = new Array(6) ;
arr[0] = 10; arr[1] = 5; arr[2] = 40;
arr[3] = 25; arr[4] = 111; arr[5] = 1;
console.log(arr + "n")
console.log(arr.sort(sortNumber))
```

运行结果如下。

```
10,5,40,25,111,1
1,5,10,25,40,111
```

4. 数组的操作

1）push()方法

往数组后面添加数组，并返回数组新长度。

```
var a = ["aa","bb","cc"];
console.log(a.push("dd"));    //输出4
console.log(a);               //输出aa,bb,cc,dd
```

而 unshift()方法可向数组的开头添加一个或更多元素，并返回新的长度。

2）pop()方法和 shift()方法

pop()方法删除数组最后一个元素，并返回该元素。而 shift()方法用于把数组的第一个元素从其中删除，并返回第一个元素的值。

```
var a = ["aa","bb","cc"];
console.log(a.pop()); //输出cc
console.log(a.shift ());//输出aa
```

3）slice()方法

可从已有的数组中返回选定的元素的一个新数组。语法如下。

```
arrayObject.slice(start,end)
```

返回一个新数组包含从 start 到 end（不包括 end 元素）的 arrayObject 中的元素。参数 start 是必需的。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。也就是说，–1 指最后一个元素，–2 指倒数第二个元素，以此类推。

end 可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果这个参数是负数，那么它规定的是从数组尾部开始算起的元素。

例如：

```
var a = ['a','b','c','d','e','f','g'];
console.log(a.slice(1,2));      //输出b
console.log(a.slice(2));        //输出 c,d,e,f,g
console.log(a.slice(-4));       //输出 d,e,f,g
console.log(a.slice(-6,-2));    //输出b,c,d,e
```

a.slice(1,2)返回从下标为 1 开始，到下标为 2 之间的元素，注意并不包括下标为 2 的元素，所以仅输出'b'。a.slice(2)只有一个参数，则默认到数组最后元素，所以输出 c, d, e, f', g。

a.slice(−4)中−4 是表示倒数第 4 个元素，所以返回倒数的 4 个元素。

console.log(a.slice(−6,−2))从倒数第 6 个开始，截取到倒数第 2 个前，则返回 b,c,d,e。

4）join()方法

用于把数组中的所有元素连接起来放入一个字符串，语法如下。

```
arrayObject.join(separator)
```

separator 指定要使用的分隔符。如果省略该参数，则使用逗号作为分隔符。

```
var arr = new Array(3);
arr[0] = "George";   arr[1] = "John";   arr[2] = "Thomas";
console.log(arr.join("."));//输出George.John.Thomas
```

5. 二维数组

若数组中的元素又是数组就称其为二维数组。创建二维数组的方法如下。

方法一：先创建一个一维数组，然后该一维数组的所有元素再创建一维数组。

```
var persons = new Array(3);//创建一个一维数组
persons[0] = new Array(2); //每个元素persons[0]又是一维数组
persons[1] = new Array(2); //每个元素persons[1]又是一维数组
persons[2] = new Array(2); //每个元素persons[2]又是一维数组
persons[0][0] = "zhangsan";
persons[0][1] = 25;
persons[1][0] = "lisi";
persons[1][1] = 22;
persons[2][0] = "wangwu";
persons[2][1] = 32;
```

方法二：先创建一个一维数组，然后该一维数组的所有元素直接赋值。

```
var persons = new Array(3);
persons[0] = ["zhangsan", 25];
persons[1] = ["lisi", 21];
persons[2] = ["wangwu", 32];
```

方法三：直接赋值。

```
var persons = [["zhangsan", 25], ["lisi", 21], ["wangwu", 32]];
```

二维数组或多维数组的长度是多少？测试下面的代码。