千锋教育™
WWW.QFEDU.COM

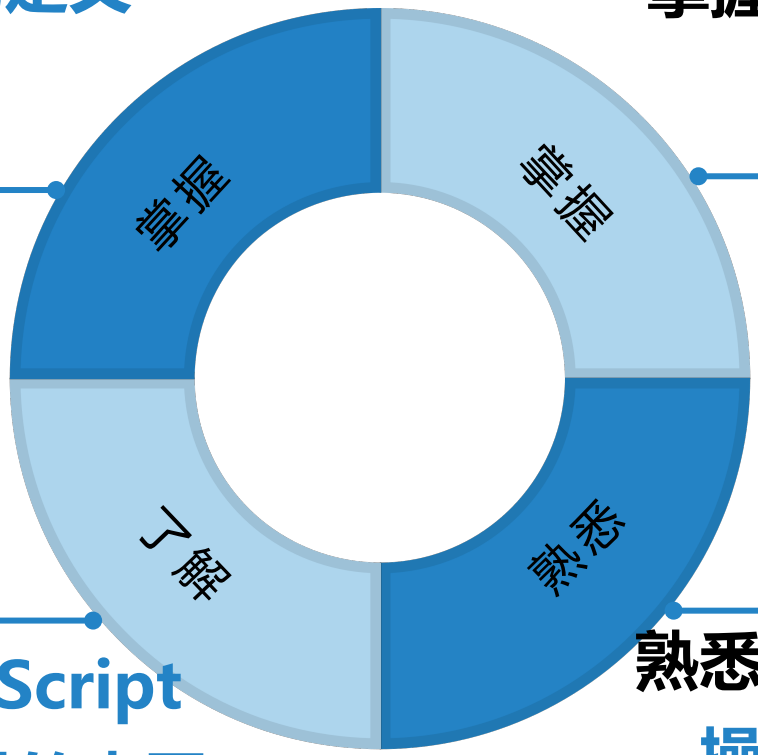
第4章 JavaScript函数

- 定义函数
- 操作函数
- 内置函数

目录

学习目标

1. 掌握函数的定义方式
2. 掌握函数的调用方式
3. 熟悉函数的基本操作行为，如传参、返回值、作用域等
4. 了解JavaScript中提供的内置函数方法



4.1 定义函数

返回目录

4.1 定义函数

千锋教育 WWW.QFEDU.COM

4.1.1 函数简介

- 程序中的函数与数学中的函数十分相似。如数学中的函数指给定一个数集A，对A应用对应法则f，记作f（A），得到另一数集B，即得到关系式B=f（A），这个关系式被称为函数关系式，简称函数。在程序中的函数也可以定义这样的方式，通过传入A值，得到对应的B值，例如，前面章节介绍的Number()方法，当传入Number(null)时，会返回0值。
- JavaScript中共有三种定义函数的方式，分别是函数声明方式、函数表达式定义法和创建对象定义法。其中，前两种最为常用。创建对象定义法即new Function()形式，由于在实际开发中很少涉及，因此本章不深入讲解，稍作了解即可。

4.1 定义函数

4.1.2 函数声明

- 利用函数声明方式定义函数，其语法格式如下：

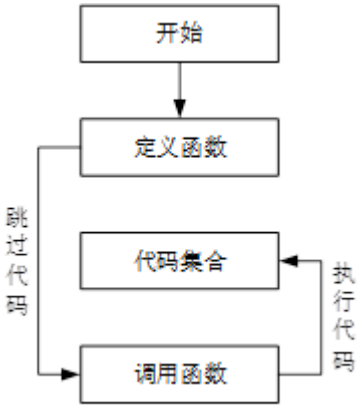
```
function 函数名(){
    代码集合;
}
```

- 其中，function为固定语法格式，其后的函数名为自定义字符，中间用空格隔开。小括号与大括号也为固定写法，在大括号中放置的是一组可以随时随地运行的代码集合。具体示例代码如下：

```
1  <script>
2      function foo(){
3          var sum = 0;
4          for( var i=1; i<=100; i++ ){
5            sum += i;
6          }
7            console.log( '从 1 加到 100 的和: '+sum );
8      }
9  </script>
```

4.1.2 函数声明

- 调用函数只需使用函数名加小括号即可。下面是函数调用流程图，如图所示。



- 函数的调用是可以重复操作的，即前面介绍过的函数调用时可重复的去执行这些代码集合。

4.1 定义函数

4.1.3 函数表达式

- 利用函数表达式方式定义函数，其语法格式如下：

```
var 函数名 = function (){
    代码集合;
};
```

- 用函数表达式的方式定义函数，是把一个函数赋值成一个变量，变量名即函数名。函数表达式的调用方式与函数声明的调用方式相同，也通过函数名加小括号的方式。具体示例代码如下：

```
1 <script>
2       var foo = function(){
3           console.log( 'Hello JS' );      // 执行
4       }
5       foo();
6 </script>
```

4.1.4 函数声明表达式与函数表达式方式的区别

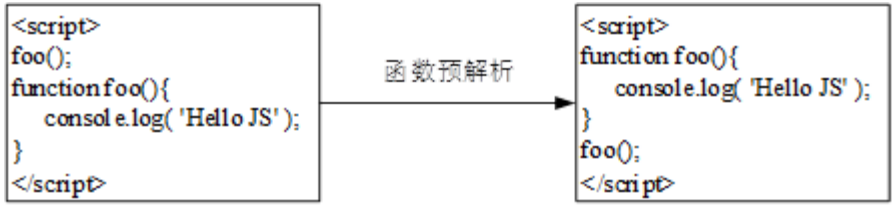
- 函数声明方式定义函数与函数表达式方式定义函数的区别主要有两点，具体如下：
- **1. 函数声明可以预解析**
- 定义函数和调用函数应遵循先定义后调用的原则。而函数声明的方式具备函数预解析的功能，因此写法上可以先写调用，再写定义。具体示例如下。

```
<script>
    foo();
    function foo(){
        console.log( 'Hello JS' );     //  执行
    }
</script>
```

- 运行结果，打印出Hello JS。这是因为无论在什么位置以函数声明的方式定义函数，都会被预先解析到<script>代码块的最开始位置，即函数预解析的特点，当调用函数时，函数已经提前定义。

4.1.4 函数声明表达式与函数表达式方式的区别

- 函数预解析过程示意图，如图所示。

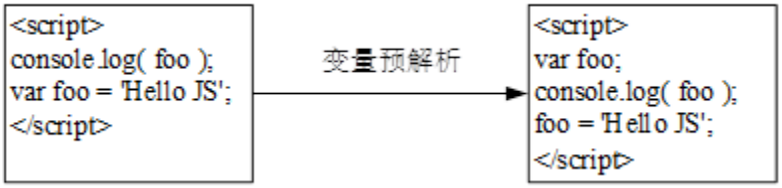


- 但这种预解析的方式并不适用于函数表达式，如果在定义函数前调用函数，则会提示报错，这是两种方式的第一个重要区别。此外，还需要了解另外一个重要概念——变量预解析，除函数有预解析外，变量同样具备预解析的特点，只是不容易被发现，具体示例代码如下：

```
1  <script>
2      console.log( foo );              // undefined
3      var foo = 'Hello JS';
4  </script>
```

4.1.4 函数声明表达式与函数表达式方式的区别

- foo变量会返回undefined未定义类型，但并不会报错。说明变量在程序中做了一些内部的处理，即变量预解析的作用，变量预解析过程示意图，如图所示。



- 函数表达式既属于定义函数，也属于定义变量，其变量值为函数。因此函数名返回undefined未定义，调用时会因找不到变量而报错。

4.1.4 函数声明表达式与函数表达式方式的区别

- **2.函数表达式可直接调用执行**
- 用函数表达式方式定义的函数，直接在定义函数后加一对小括号可以立即执行。这种方式省略了函数名调用的方式，在很多场合中会用到。本节只了解如何使用即可，后面章节中会具体分析其使用方式。具体示例代码如下：

```
<script>
    var foo = function(){
        console.log('Hello JS');        //  执行
    }();
</script>
```

- 函数声明的方式使用例中方式调用程序会报错，这是函数声明定义函数和函数表达式定义函数第二个比较大的区别。

4.2 操作函数

返回目录

4.2 操作函数

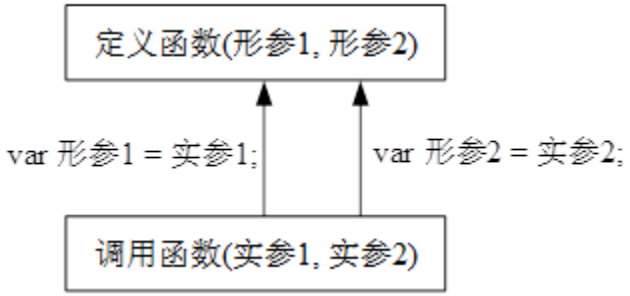
- 简单的定义和调用还不能够体现出函数的强大。在本小节中，将为大家介绍函数的一些常用操作，通过常用操作可以更好地理解和使用函数，学会操作函数会对JavaScript编程起到至关重要的作用。

4.2.1 函数传参

- 定义函数和调用函数时，函数名后面的小括号内都可以添加内容，添加的内容称为函数的参数。在定义函数中添加的参数称为形式参数，简称形参；而在调用函数中添加的参数称为实际参数，简称实参。其语法格式如下：

```
function foo(形参1，形参2){
  代码集合;
}
foo(实参1，实参2);
```

- 在函数中可以不添加参数，也可以添加多个参数。参数可以把一个变量拆分成变量名和变量值两部分，变量名作用在形参中，变量值作用在实参中。函数传递参数流程图，如图所示。



4.2 操作函数

4.2.1 函数传参

- 前面计算过1累加到100，如果要运算1累加到200，或是100累加到150，当不进行传参处理时，可能要写多行代码，如果使用函数传参的操作方式，就可以使程序相对简单许多。

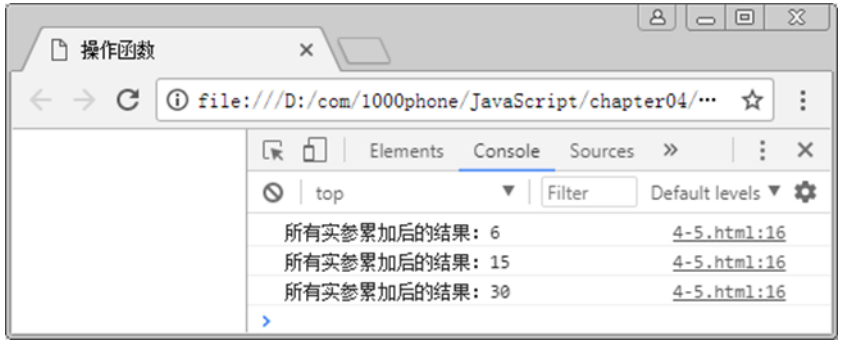
4.2.2 arguments

- 当参数过多时，参数操作起来可能会不方便。例如，实参数量由三个变到五个，形参数量也需从三个变到五个，而函数会变得极其复杂，并且不能复用函数。为此，函数中提供了arguments对象，用来表示实参的集合，且具备length属性，arguments.length值表示实参的个数。。接下来通过案例演示将所有的实参累加打印出结果，且能够复用函数，具体如例所示。

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>操作函数</title>
</head>
<body>
<script>
    function foo(){
        var result = 0;
        for( var i=0; i<arguments.length; i++ ){
            result += arguments[i];
        }
        console.log('所有实参累加后的结果：' + result );
    }
    foo(1,2,3);
    foo(1,2,3,4,5);
    foo(2,4,6,8,10);
</script>
</body>
</html>
```

4.2.2 arguments

- 在例中，当实参个数不确定时，需要把实参累加到一起，同时输出结果。首先定义一个函数foo，在函数内部利用arguments对象的特点，对其进行循环操作，每次循环可获得arguments对象中对应的每一项实参，并进行累加，最终输出图所示结果。



4.2.3 函数返回值

- 在调用函数时，并不是在函数体内得到函数执行结果，这就需要通过return语句实现，具体示例代码如下：

```
<script>
    function foo(){
        var result = 0;
        for( var i=0; i<arguments.length; i++ ){
            result += arguments[i];
        }
        return result;
    }
    var sum = foo(1,2,3);
    console.log( '所有实参累加后的结果：' + sum );    // 6
</script>
```

- 上述示例中，foo(1,2,3)调用后返回值为return的返回值6，然后将6赋值给sum变量，最后打印出sum值为6。

4.2.3 函数返回值

- 需要注意return语句的特点，即其后面的语句不执行，具体示例代码如下：

```
<script>
    function foo(a,b,c){
        console.log(a);            //   1
        return;
        console.log(b);            //   不执行
        console.log(c);            //   不执行
    }
    var bar = foo(1,2,3);
    console.log(bar);              //  undefined
</script>
```

- 可以发现，只打印出a和bar值，而b和c值并未打印，因为前面有return语句。另外，当return不返回任何值时，会得到undefined；默认不写return语句时，函数执行完也会得到undefined。利用return语句后的代码不执行的特点，可以对函数内部做一些判断操作，一旦满足某些条件，就不执行后面的代码。

4.2.4 函数作用域

- 作用域指作用范围，函数是具备作用域的。假设在foo函数中定义了bar函数，即bar函数只能在foo函数内进行调用，而不能在foo函数外调用，这就是函数作用域的特点。具体示例代码如下：

```
<script>
    function foo(){
        function bar(){
            console.log('函数作用域');
        }
        bar();          // √
    }
    foo();
    bar();              // ×
</script>
```

- 函数内能够调用函数外定义的函数称为作用域，而函数外不能够调用函数内定义的函数，同样，变量也具备作用域，与函数作用域类似，具体示例如下：

```
<script>
    var bar = 10;           //  全局变量
    function foo(){
        var baz = 20;       //  局部变量
        bar;                //  √
    }
    foo();
    baz;                    // ×
</script>
```

4.2.4 函数作用域

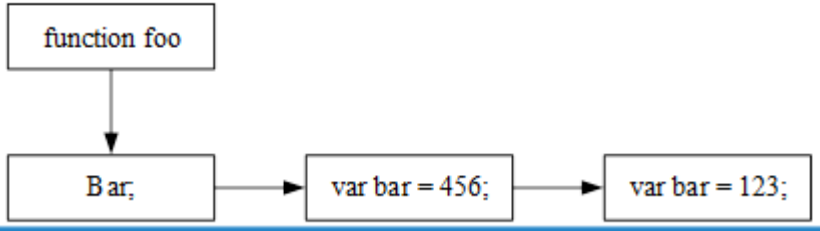
- 通常情况下，函数外定义的变量称为全局变量，而函数内定义的变量称为局部变量，变量作用域与函数作用域类似。
- 除变量作用域外，还存在变量作用域链。变量作用域链是指变量的查找过程，即每一段JavaScript代码（包含函数）都会有一个与之关联的作用域链。作用域链是一个对象列表或者链表，对象中定义这段代码“作用域中”的变量。
- 当JavaScript需要查找变量x的值时（这个过程称为变量解析），它会从链的第一个对象开始查找，如果这个对象有一个名为x的属性，则会直接使用这个属性的值，如果第一个对象中没有名为x的属性，则JavaScript会继续查找链上的下一个对象。如果第二个对象依然没有名为x的属性，则会继续查找下一个，以此类推。如果作用域链上没有任何一个对象含有属性x，则认为这段代码的作用域链上不存在x，最终抛出一个引用错误异常。

4.2.4 函数作用域

- 作用域链的开始位置为调用变量的位置，然后再一层层向外链接，下面通过示例代码理解作用域链，具体示例代码如下：

```
1  <script>
2      var bar = 123;
3      function foo(){
4          var bar = 456;
5          console.log( bar );        //  456
6      }
7      foo();
8  </script>
```

- 可以看到程序中有两个bar变量，当bar被调用时，将通过作用域链开始位置进行查找，先找到最近定义的变量位置，即第4行代码，作用域链停止查找，直接返回结果，即456。如果没有第4行代码，作用域链在开始位置找不到结果，就会向作用域的外层继续查找，直到找到想要的变量，第2行代码，返回123。bar的作用语域链，如图所示。



4.2.5 函数与事件

- 事件是在网页中触发某种行为，如单击、鼠标滑过、用户输入等，这些操作行为称为事件操作，JavaScript通过事件操作响应后续的处理方式，如单击后弹出提示框、鼠标滑过变换背景图、用户输入后显示输入是否正确等处理行为。
- 常见的事件有onclick单击事件(通过鼠标左键和键盘回车触发)、onmousedown鼠标按下事件、onmouseup鼠标抬起事件、onmouseover鼠标移入事件、onmouseout鼠标移出事件等。在后面章节会讲解事件如何操作，这里简单了解即可。
- 事件一般需要配合函数才能完成后续的操作。其语法格式如下：

```
元素.事件 = function(){
  代码集合;
};
```

- 其中，元素即HTML元素，如按钮或链接。事件即上面介绍的事件，如单击事件或鼠标移入事件。当事件被触发时，函数内的代码语句就会被执行。

4.2.5 函数与事件

- 事件的写法类似于函数表达式的写法，都是把等号右侧的函数赋值给等号左侧的表达式。函数表达式可以直接调用，事件函数能不能也直接调用呢？接下来通过案例演示，具体如例所示。

```
<body>
    <input id="btn" type="button" value="显示列表">
    <ul id="list" style="display:none;">
        <li>111</li>
        <li>111</li>
        <li>111</li>
    </ul>
</body>
<script>
    var btn = document.getElementById('btn');
    var list = document.getElementById('list');
    var onoff = true;
    btn.onclick = function(){
        if(onoff){
            list.style.display = 'block';
            btn.value = '隐藏列表';
        }
        else{
            list.style.display = 'none';
            btn.value = '显示列表';
      }
            onoff = !onoff;
    };
    btn.onclick();                  //  直接调用实行事件函数
</script>
```

4.2.6 实际运用

- 已经掌握了函数的基本操作，下面介绍在实际开发中如何灵活的运用函数。利用函数的特点，一般可以解决复用代码、简化操作和兼容处理三类问题。
- **1. 复用代码**
- 前面利用函数实现100到150累加，其实就是复用代码。下面实现一个复用代码的例子，即数值的阶乘计算（所有小于及等于该数的正整数的积），具体如例所示。

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>操作函数</title>
</head>
<body>
<script>
    function factorial(n){
       var result = 1;
       for( var i=2; i<=n; i++ ){
          result *= i;
       }
         return result;
    }
    console.log(factorial(4));        // 4 的阶乘
    console.log(factorial(5));        // 5 的阶乘
    console.log(factorial(6));        // 6 的阶乘
</script>
</body>
</html>
```

4.2.6 实际运用

- **2.简化操作**
- 可以利用函数对JavaScript语句进行简化操作，例如简化getElementById()方式获取元素。具体示例代码如下。

```
<script>
      function $(id){
        return document.getElementById(id);
      }
      $('btn').onclick = function(){
          $('list').style.display = 'block';
      };
</script>
```

4.2.6 实际运用

- **3.兼容处理**
- 前面介绍过获取最终样式的两种方法，即getComputedStyle()（标准方法）和currentStyle()（非标准方法），为了兼容老式浏览器，必须对这两个方法都进行操作。利用函数实现兼容处理会非常的方便。具体示例代码如下。

```
<script>
    function css(obj , attr){
        if(obj.currentStyle){
            return obj.currentStyle[attr];
        }
        else{
            return getComputedStyle(obj)[attr];
        }
    }
    css(elem1,'width');
    css(elem2,'color');
</script>
```

- 示例中，获取元素样式值时，并没有采用点（.）的方式，而是使用中括号（[]）的方式。（[]）属于获取属性的第二种方式，与(.)方式起到的作用一样。

4.3 内置函数

返回目录

4.3 内置函数

千锋教育 WWW.QFEDU.COM

4.3.1 弹窗模式

- 在JavaScript中一共有alert()警告框、confirm()确认框、prompt()对话框三种弹窗模式，下面进行详细讲解。
- **1. alert()警告框**
- 调用alert()方法时，会弹出一个警告框，alert()函数的参数为警告框的提示文字，当单击“关闭” 按钮时，会关闭警告框。
- **2. confirm()确认框**
- 调用confirm()方法时，会弹出一个确认框，confirm()函数的参数为确认框的提示文字。确认框中会存在“确定” 按钮和“取消” 按钮，当单击“确认” 按钮时，函数会返回true；当单击“取消” 按钮时，函数会返回false；当单击“关闭” 按钮时，会关闭确认框。

4.3.1 弹窗模式

- **3. prompt()对话框**
- 调用prompt ()方法时，会弹出一个对话框，prompt ()函数接收两个参数，第一个参数为提示输入的文本内容，第二个参数为输入框的默认输入内容。对话框中包含“确定” 按钮和“取消” 按钮，当单击“确定” 按钮时，函数会返回输入的内容，如果没有输入内容则返回空字符串；当单击“取消” 按钮时，函数会返回null；当单击“关闭” 按钮时，关闭对话框。

4.3.2 数字字符转为数字

- 在JavaScript中提供了两个将数字字符串转化为真正数字的内置函数，即parseInt()和parseFloat()。下面进行详细介绍。
- **1. parseInt()**
- parseInt()函数可解析字符串，并返回整数。parseInt()函数接收两个参数，第一个参数是要转换的字符串；第二个参数为可选值，表示以不同的进制进行解析。
- **2. parseFloat()**
- parseFloat()函数可解析字符串，并返回浮点数（带有小数）。parseFloat函数只接收一个参数，不存在进制问题。具体示例代码如下：

```
<script>
    var foo = '123.45';
    console.log( parseFloat(foo) );          // 123.45
    console.log( typeof parseFloat(foo) );   // number
</script>
```

4.3 内置函数

4.3.2 数字字符转为数字

- parseInt()函数和parseFloat()函数，都需要注意，字符串中的首个字符是否为数字。如果是数字，则对字符串进行解析，直到到达数字的末端为止，然后以数字返回该数字，而不是作为字符串。具体示例代码如下：

```
1  <script>
2      var foo = '100px';
3      var bar = '$123';
4      console.log( parseInt(foo) );         // 100
5      console.log( parseInt(bar) );         // NaN
6  </script>
```

4.3.3 eval

- 在JavaScript中，eval()函数可以把字符串当作JavaScript表达式一样去执行。其语法格式如下：

```
eval(字符串)
```

- eval()函数接收一个字符串参数。当字符串是JavaScript语句时，eval()函数会对其进行解析，得到对应的JavaScript语法。具体示例代码如下：

```
1  <script>
2      eval('function foo(){ console.log(123); }');
3      foo();         // 123
4  </script>
```

- 一般情况下，一定要慎用eval()函数，因为eval()函数能解析任何形式的字符串，可能会解析出存在安全问题的代码，导致页面受到不必要的攻击。

4.3.4 isNaN

- innerHTML方法是用来获取和设置指定标签内的内容。其内容包括文本、标签等信息。前面已经介绍过NaN和isNaN()方法，isNaN()方法是用来判断一个值是否是NaN值。
- 可以利用此内置函数完成一个小实例，即页面中有两个输入框，当输入框输入的不是数字时，提示请输入数字类型，当输入的两个值都为数值时，则弹出累加后的结果。注意，输入框中输入的内容为字符串类型。

4.3 内置函数

4.3.5 有限数值

- 在JavaScript中，isFinite()函数用来确定某个数是否为有限数值。isFinite()函数接收一个参数，可以是整数、浮点数。如果该参数为非数字、正无穷数和负无穷数，则返回false；否则，返回true。如果是字符串类型的数字，就会自动转化为数字型。在JavaScript中，Infinity表示“无穷”，- Infinity表示“负无穷”。具体示例代码如下。

```
1  <script>
2       console.log( isFinite(123) );            //  true
3       console.log( isFinite('456') );          //  true
4       console.log( isFinite(Infinity) );       //  false
5       console.log( isFinite(-Infinity) );      //  false
6  </script>
```

本章小结

- 通过本章的学习，大家能够掌握如何定义函数、调用函数及多函数相关的操作，如传参、argumnets、作用域、return返回值等函数常见操作。重点理解JavaScript提供的一些可以直接使用的内置函数。

千锋教育™
WWW.QFEDU.COM

THANK YOU