Day 05---函数

一、函数概述

1.1 函数的概念

> 函数就是把完成特定功能的一段代码封装起来。给该功能起一个名字（函数名）。
>
> 哪里需要实现该功能就在哪里调用该函数。
>
> 函数可以在任何时间任何地方调用。

- 函数是完成某一功能的代码段
- 函数是可重复执行的代码段
- 函数方便管理和维护 便于复用

1.2 函数的作用

- 使程序变得更简短而清晰
- 有利于程序维护
- 可以提高程序开发的效率
- 提高了代码的重用性(复用性）

二、自定义函数

2.1 函数的声明

> 函数声明也叫函数定义。
>
> 使用函数，必须要先定义。
>
> 语法:

```
function 函数名(形式参数1, 形式参数2, ...){
   //函数体
}
```

> 说明:

1. function 是定义函数用到的关键字，必须存在，不能省略。这个单词的所有字母必须小写。
2. 函数名是为了能让这个函数在别的地方调用。
3. 函数名的后面必须跟一对圆括号()。
4. 圆括号内根据需要可以声明形参，定义形参的时候只需要参数名，而不要var来声明。形参的个数不固定，根据需要，可以有多个形参，也可以一个也没有。(圆括号内的形参有时候我们也称之为形参列表)
5. 圆括号后面跟一对大括号{ },表示函数代码的开始和结束。圆括号内写我们要执行的一系列的代码，这一系列的

代码我们称之为函数体。

6. 函数体内可以根据需要决定是否添加return语句。return语句表示给方法的调用者返回一个值。 总体来说return语句有两个作用：1、给调用者返回值 2、结束函数(只要碰到return语句，不管代码执行到了什么地方，也不管进入了多少层循环，那么方法都会立即执行，并返回)。
7. return语句的语法是：return 返回值; 返回值可以省略，表示仅仅结束函数。
8. 如果省略return语句或者有return但是没有返回值，这个时候，返回的是undefined

> 注意:

- 如果仅仅声明了函数，而没有在别的任何地方使用这个函数，则这个函数永远不会执行。
- 使用函数，我们称之为函数调用。
- 永远记住这句话: 函数只有被调用才能被执行

> 函数声明实例：

```
/*
  声明一个函数
  功能：实现两个数的相加，并返回结果
  num1：第一个数
  num2：第二个数
*/
function add(num1, num2){ //add是方法名。  num1和num2是形式参数
  var sum = num1 + num2; //声明一个变量，来存储num1和num2的相加的值
  return sum; //使用return语句返回结果
}

//其中函数体的代码，也可以简化一行代码
function add(num1, num2){
  return num1 + num2;  // 直接返回num1 + num2这个表达式。则会自动计算，并返回计算的结果
}
```

2.2 函数的调用

> 函数只有被调用才能被执行。所以，如果要让函数内的代码执行，则必须先调用。
>
> 函数调用语法：

```
方法名(实参1，实参2);
```

> 说明：

1. 调用方法的时候，方法名是必须的。而且方法名是找到已定义的函数的唯一识别。
   函数调用时，方法名后面的一对圆括号不能省略。
2. 实参(实际参数)列表，对应着函数声明的形参列表部分。传递的实参，会被形参接受，然后就可以在函数内部使用了。
3. 可以用一个变量去存储方法执行完毕之后的返回值。

> 函数调用实例:

```
//利用方法名 add 去调用。   5 和 10是实参，会传递给方法声明的形参： num1和num2。
var sum = add(5, 10);  //重新定义一个变量sum来接受方法的返回值。
alert(sum);
//可以多次调用同一个函数，通过传入不同的参数来计算不同的值。我们定义的函数中的代码就完成了复用
alert(add(10, 20));
```

2.3 函数的命名规范

> 函数名是一个函数非常重要的特征，为了方便调用者调用函数，函数的命名必须遵循一定的规范。

1. 不能使用系统的关键字和保留字
2. 命名要有意义：见名知意。从方法名应该可以大致推测到这个方法的功能。不要起诸如a、b这些没有意义的方法名，
3. 业界多采用驼峰命名法来给函数命名。驼峰命名法：首字母小写，其余单词的首字母大写。例如：add、onCreate、doSomething。(xxxYyyZzz)
4. 函数的命名是严格区分大小写的。比如：add和Add是两个完全不同的函数名。
5. 注意：声明函数时，如果后定义的函数名与前面定义的函数名重复了，则后定义的会覆盖前定义的。

三、函数的参数

> 在大部分编程语言中，函数的参数都分为两种：
>
> 1、声明时的参数是形参(用于接收值)
>
> 2、调用方法的时候传递的参数是实参(用于把值传递给形参)
>
> 但是，对于JavaScript这门弱类型语言，对函数参数的处理方面与别的强类型语言有很大不同，且灵活了很多：

1. 形参声明不需要var。(因为所有的变量都是用var来声明，所以这个地方省略了没啥问题。添加var会出现语法错误)
2. 形参，在函数内部可以作为一个普通的局部变量使用。而且通常情况下，实参已经把值赋值给了形参。例如上面例子中的num1和num2在方法内部就可以作为一个普通的局部变量使用。
3. 在调用函数的时候，实参的个数可以形参的个数一致，也可以不一致。实参可以比实参的个数多，也可以比实参的个数少。
4. 形参和实参匹配的时候总是按照顺序匹配。
5. 函数对传入的实参，既不做类型的检查，也不做个数的检查。如果需要这些检查，需要开发者自行实现代码完成

> 示例代码:

```
//  声明一个函数，形参的个数2个
function doSomething(num1, num2){
  alert(num1 + ":" + num2); //  num1和num2，在方法的内部可以作为普通的局部变量使用。
}

//调用函数：把 5 传递给num1， 把 "a" 传递给num2
doSomething(5, "a");     // 弹出：5:a

//调用函数：把 "a" 传递给num1， 把 “b” 传递给num2 ."c"没有形参接受。
doSomething("a", "b", "c"); //虽然比形参的个数多，但是仍然可以正常调用。

//调用函数：把 "a"传递给num1.
doSomething("a"); //虽然比形参的个数少，但是仍然可以正常调用
```

1、在通过实参给形参传递参数的时候，如果实参的个数比形参少，则接收不到值的形参的初始化值为 undefined 的。
2、传递的实参比形参多的时候，多余的实参没有形参接受，正常情况下无法访问到传递过来的多余的实参。
3、其实多余的实参并没有丢失，函数帮我们保存在了一个变量(对象)中。这个变量的名字就是 arguments。
4、arguments不需要开发者手动创建，在调用函数的时候，会自动创建，并把传递过来的所有实参的值都保存在这个变量中。
5、可以暂时把arguments当成一个数组来理解，虽然他实际并不是一个数组。(其实是个对象，每个参数都是他的一个属性值)。

> arguments使用代码简单实例:
>
> 关于数组的详细使用，明天再讲。

```
function doSomething (num1, num2) {
  alert(num1 === arguments[0]);  //true
  alert(num2 === arguments[1]); // true
  /*
    只要方法被调用，则一定会自动创建一个arguments对象，这个对象会存储传过来的所有的实参。
  */
  for (var i = 0; i < arguments.length; i++) {
    alert(arguments[i]);
  }
}
doSomething("a", "b", "c", "d");
```

四、变量的作用域

> 变量的作用域指的是，变量起作用的范围。也就是能访问到变量的有效范围。
>
> JavaScript的变量依据作用域的范围可以分为：
>
> - 全局变量
> - 局部变量

4.1 全局变量

> 定义在函数外部的变量都是全局变量。

> 全局变量的作用域是当前文档，也就是当前文档所有的JavaScript脚本都可以访问到这个变量。
>
> 下面的代码是书写在同一个HTML文档中的2个JavaScript脚本:

```
<script type="text/javascript">
    //定义了一个全局变量。那么这个变量在当前html页面的任何的JS脚本部分都可以访问到。
    var v = 20;
    alert(v); //弹出：20
</script>
<script type="text/javascript">
    //因为v是全局变量，所以这里仍然可以访问到。
    alert(v);  //弹出：20
</script>
```

> 再看下面一段代码：

```
<script type="text/javascript">
    alert(a);
    var a = 20;
</script>
```

> 运行这段代码并不会报错， alert(a); 这行代码弹出：undefined。

为什么在声明 a 之前可以访问变量 a 呢? 能访问 a 为什么输出是undefined而不是20呢？

声明提前！

- 所有的全局变量的声明都会提前到JavaScript的前端声明。也就是所有的全局变量都是先声明的，并且早于其他一切代码。
- 但是变量的赋值的位置并不会变，仍然在原位置赋值。

> 所以上面的代码等效下面的代码：

```
<script type="text/javascript">
    var a; //声明提前
    alert(a);
    a = 20; //赋值仍然在原来的位置
</script>
```

4.2 局部变量

> 在函数内声明的变量，叫局部变量！表示形参的变量也是局部变量！
>
> 局部变量的作用域是局部变量所在的整个函数的内部。 在函数的外部不能访问局部变量。

```
<script type="text/javascript">
    function f(){
        alert(v);  //    弹出：undefined
        var v = "abc";  // 声明局部变量。局部变量也会声明提前到函数的最顶端。
        alert(v);   //  弹出：abc
    }
    alert(v);  //报错。因为变量v没有定义。 方法 f 的外部是不能访问方法内部的局部变量 v 的。
 </script>
```

4.3 全局变量和局部变量的一些细节

> 看下面一段代码:

```
<script type="text/javascript">
    var m = 10;
    function f(){
        var m = 20;
        alert("方法内部：" + m);  //代码1
    }
    f();
    alert("方法外部：" + m); //代码2
</script>
```

在方法内部访问m，访问到的是哪个m呢？局部变量的m还是全局变量的m？

4.3.1 全局变量和局部变量重名问题

1. 在上面的代码中，当局部变量与全局变量重名时，局部变量的作用域会覆盖全局变量的作用域。也就是说在函数内部访问重名变量时，访问的是局部变量。所以 "代码1" 部分输出的是20。
2. 当函数返回离开局部变量的作用域后，又回到全局变量的作用域。所以代码2输出10。
3. 如何在函数访问同名的全局变量呢？通过：window.全局变量名

```
<script type="text/javascript">
    var m = 10;
    function f(){
        var m = 20;
        alert(window.m);  //访问同名的全局变量。其实这个时候相当于在访问window这个对象的属性。
    }
    f();
</script>
```

4.3.2 JavaScript中有没有块级作用域？

> 看下面一段代码:

```
<script type="text/javascript">
  var m = 5;
  if(m == 5){
    var n = 10;
  }
  alert(n); //代码1
</script>
```

> 代码1输出什么？ undefined还是10？还是报错？
>
> 输出10！

- JavaScript的作用域是按照函数来划分的
- JavaScript没有块级作用域

在上面的代码中，变量 n 虽然是在 if 语句内声明的，但是它仍然是全局变量，而不是局部变量。

只有定义在方法内部的变量才是局部变量

> 注意：

- 即使我们把变量的声明放在 if、for等块级语句内，也会进行声明提前的操作！

五、匿名函数

5.1 声明匿名函数

> 匿名函数是指没有函数名的函数。
>
> 看下面一段代码：

```
<script type="text/javascript">
    /*
    //这里定义了一个函数，而且没有函数名。这样写语法是错误的,如果允许这样定义，那么根本就没有办法调用。
    //所以，我们可以用一个变量来存储一下
    function(){

    }
    */
  // 声明了一个匿名函数，并把匿名函数赋值给变量f。 注意这个时候这个匿名函数并没有执行。
  var f = function(){
    alert()
  }
  //我们可以把变量 f 当做一个函数名来调用
  f();  //调用上面定义的匿名函数
</script>
```

> 说明：

1. 匿名函数除了没有函数名之外，其他与普通的函数没有任何区别。
2. 如果想在别的地方调用匿名函数，则应该声明一个变量，并把匿名函数赋值给这个变量
3. 可以把这个变量名做为函数名来调用。参数传递，方法返回值和普通的函数一样。

5.2 匿名函数的作用

1. 函数表达式可以存储在变量中，变量也可以作为一个函数使用。
2. 可以将匿名函数作为参数传递给其他函数

```
<script type="text/javascript">
  //声明一个函数，参数接受一个函数
  function fun1 (fun) {
    if(typeof fun == "function"){    //如果传递的是function类型，则调用这个函数
      fun()
    }
  }
  // 调用函数fun1，并传入一个匿名函数实参
  fun1(function () {
    alert("这个是匿名函数的代码")
  });
</script>
```

3. 可以通过匿名函数完成某些一次性的任务。

- 如果一个函数不需要重复执行，则可以定义一个匿名函数

5.3 匿名函数立即执行

> 有些场景，我们需要定义完函数之后立即执行，这个时候可以定义一个匿名函数来完成。

```
(function () {
    alert("匿名函数立即执行")
})();
```

> 说明

1. 需要把匿名函数用一对圆括号括起来，把匿名函数作为一个整体来对待
2. 最后再添加一对圆括号表示调用函数。这样定义的匿名函数就会立即执行
3. 当然，这个时候即使给这个函数加上方法名，也可以调用。不过这种情况为什么还要加方法名呢?

六、函数重载

6.1 JavaScript支持重载吗?

> 重载在通常的面向对象语言中这样定义：
>
> 1. 方法名相同
> 2. 参数列表不同
>
> 满足这两个条件的函数就构成重载。

> 在JavaScript中，如果两个函数的名字相同，那么后定义的函数会覆盖先定义的函数。
>
> 所以，JavaScript不能完成通常意义的重载！JavaScript不支持重载

```
<script type="text/javascript">
  // 定义一个add函数
  function add (num1, num2) {
    alert("两个参数的函数...");
    return num1 + num2;
  }
  //函数名和上面定义的函数名相同，所以会覆盖上面的函数
  function add (num1, num2, num3) {
    alert("三个参数的函数");
    return num1 + num2 + num3;
  }
  add(1, 2, 3);  //调用3个参数的函数      返回6
  add(1, 2);     //调用3个参数的函数       返回undefined
</script>
```

6.2 模拟重载

> 虽然JavaScript不支持重载，但是我们可以通过一定的手动模拟出重载

```
<script type="text/javascript">
  function add () {
    var sum = 0;
    for (var i = 0; i < arguments.length; i++) {
      sum += arguments[i];
    }
    return sum;
  }
  alert(add(2, 3));
  alert(add(5, 6, 10));
</script>
```

七、函数的递归调用

> 递归调用是指的，在函数的内部调用当前函数。即自己调用自己。
>
> 使用递归一定要满足下面条件，否则很容易出现死循环。

1. 一定要有结束条件。
2. 随着递归的深入，应该逐步靠近结束条件。(结束条件也好收敛)

> 递归案例：计算一个数的阶乘

```
<script type="text/javascript">
    function jieCheng (num) {
        if(num == 1){
            return 1;
        }
        //递归调用
        return jieCheng(num - 1) * num;
    }
    alert(jieCheng(6));
</script>
```