JavaScript基础第05天笔记

1 - 作用域

1.1 作用域概述

通常来说，一段程序代码中所用到的名字并不总是有效和可用的，而限定这个名字的可用性的代码范围就是这个名字的作用域。作用域的使用提高了程序逻辑的局部性，增强了程序的可靠性，减少了名字冲突。

JavaScript（es6前）中的作用域有两种：

- 全局作用域
- 局部作用域（函数作用域）

1.2 全局作用域

作用于所有代码执行的环境(整个 script 标签内部)或者一个独立的 js 文件。

1.3 局部作用域

作用于函数内的代码环境，就是局部作用域。 因为跟函数有关系，所以也称为函数作用域。

1.4 JS没有块级作用域

- 块作用域由 { } 包括。
- 在其他编程语言中（如 java、c#等），在 if 语句、循环语句中创建的变量，仅仅只能在本 if 语句、本循环语句中使用，如下面的Java代码：

  java有块级作用域：

  ```
  if(true){
    int num = 123;
    system.out.print(num);  // 123
  }
  system.out.print(num);    // 报错
  ```

  以上java代码会报错，是因为代码中 { } 即一块作用域，其中声明的变量 num，在 “{ }” 之外不能使用；

  而与之类似的JavaScript代码，则不会报错：

Js中没有块级作用域（在ES6之前）

```
if(true){
  var num = 123;
  console.log(123); //123
}
console.log(123);   //123
```

2 - 变量的作用域

在JavaScript中，根据作用域的不同，变量可以分为两种：

- 全局变量
- 局部变量

2.1 全局变量

在全局作用域下声明的变量叫做全局变量（在函数外部定义的变量）。

- 全局变量在代码的任何位置都可以使用
- 在全局作用域下 var 声明的变量 是全局变量
- 特殊情况下，在函数内不使用 var 声明的变量也是全局变量（不建议使用）

2.2 局部变量

在局部作用域下声明的变量叫做局部变量（在函数内部定义的变量）

- 局部变量只能在该函数内部使用
- 在函数内部 var 声明的变量是局部变量
- 函数的形参实际上就是局部变量

2.3 全局变量和局部变量的区别

- 全局变量：在任何一个地方都可以使用，只有在浏览器关闭时才会被销毁，因此比较占内存
- 局部变量：只在函数内部使用，当其所在的代码块被执行时，会被初始化；当代码块运行结束后，就会被销毁，因此更节省内存空间

3 - 作用域链

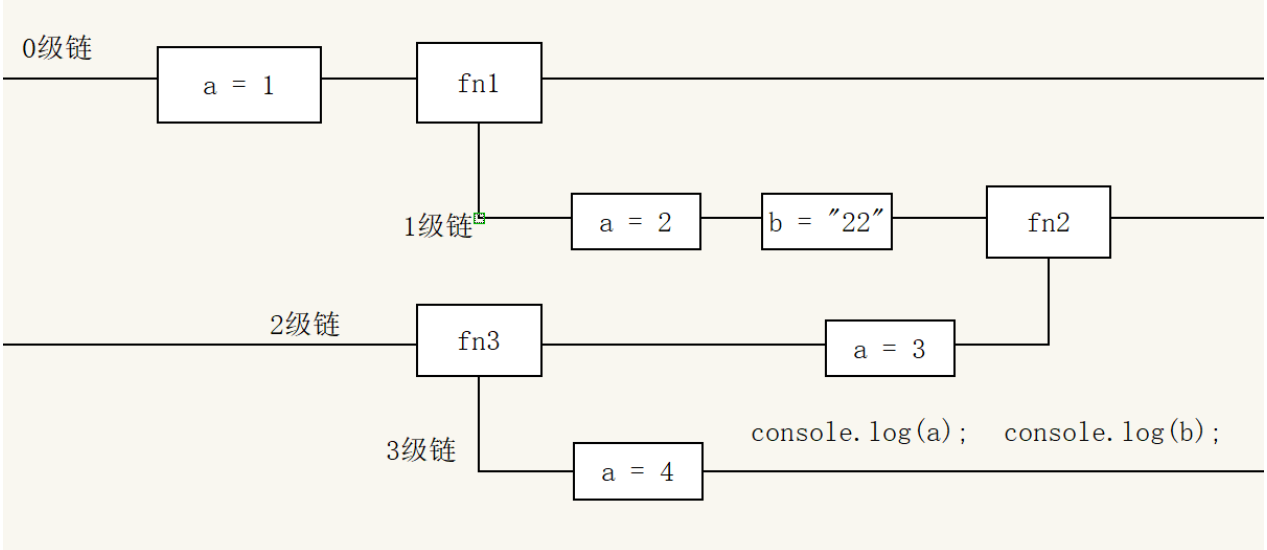
只要是代码都一个作用域中，写在函数内部的局部作用域，未写在任何函数内部即在全局作用域中；如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域；根据在**[内部函数可以访问外部函数变量]**的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链

案例分析1：

```
function f1() {
    var num = 123;
    function f2() {
        console.log( num );
    }
    f2();
}
var num = 456;
f1();
```

作用域链：采取就近原则的方式来查找变量最终的值。

```
var a = 1;
function fn1() {
    var a = 2;
    var b = '22';
    fn2();
    function fn2() {
        var a = 3;
        fn3();
        function fn3() {
            var a = 4;
            console.log(a); //a的值 ?
            console.log(b); //b的值 ?
        }
    }
}
fn1();
```



4 - 预解析

4.1 预解析的相关概念

JavaScript 代码是由浏览器中的 JavaScript 解析器来执行的。JavaScript 解析器在运行 JavaScript 代码的时候分为两步：预解析和代码执行。

- 预解析：在当前作用域下, JS 代码执行之前，浏览器会默认把带有 var 和 function 声明的变量在内存中进行提前声明或者定义。
- 代码执行： 从上到下执行JS语句。

  预解析会把变量和函数的声明在代码执行之前执行完成。

4.2 变量预解析

预解析也叫做变量、函数提升。
变量提升（变量预解析）： 变量的声明会被提升到当前作用域的最上面，变量的赋值不会提升。

```
console.log(num);  // 结果是多少?
var num = 10;      // ?
```

结果：undefined

注意：**变量提升只提升声明，不提升赋值**

4.3 函数预解析

函数提升： 函数的声明会被提升到当前作用域的最上面，但是不会调用函数。

```
fn();
function fn() {
    console.log('打印');
}
```

结果：控制台打印字符串 --- "打印"

注意：函数声明代表函数整体，所以函数提升后，函数名代表整个函数，但是函数并没有被调用!

4.4 函数表达式声明函数问题

函数表达式创建函数，会执行变量提升，此时接收函数的变量名无法正确的调用：

```
fn();
var  fn = function() {
    console.log('想不到吧');
}
```

结果：报错提示 ”fn is not a function"

解释：该段代码执行之前，会做变量声明提升，fn在提升之后的值是undefined；而fn调用是在fn被赋值为函数体之前，此时fn的值是undefined，所以无法正确调用

5 - 对象

5.1 对象的相关概念

- 什么是对象?

  在 JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等。

  对象是由属性和方法组成的。

  - 属性：事物的特征，在对象中用属性来表示（常用名词）
  - 方法：事物的行为，在对象中用方法来表示（常用动词）

- 为什么需要对象?

  保存一个值时，可以使用变量，保存多个值（一组值）时，可以使用数组。

  如果要保存一个人的完整信息呢?

  例如，将“张三疯”的个人的信息保存在数组中的方式为：

```
var arr = [‘张三疯’, ‘男', 128,154];
```

上述例子中用数组保存数据的缺点是：数据只能通过索引值访问，开发者需要清晰的清除所有的数据的排行才能准确地获取数据，而当数据量庞大时，不可能做到记忆所有数据的索引值。

为了让更好地存储一组数据，对象应运而生：对象中为每项数据设置了属性名称，可以访问数据更语义化，数据结构清晰，表意明显，方便开发者使用。

使用对象记录上组数据为：

```
var obj = {
    "name":"张三疯",
    "sex":"男",
    "age":128,
    "height":154
}
```

JS中的对象表达结构更清晰，更强大。

5.2 创建对象的三种方式

- 利用字面量创建对象

  使用对象字面量创建对象：

  就是花括号 { } 里面包含了表达这个具体事物（对象）的属性和方法；{ } 里面采取键值对的形式表示

  - 键：相当于属性名
  - 值：相当于属性值，可以是任意类型的值（数字类型、字符串类型、布尔类型，函数类型等）

    代码如下：

    ```
    var star = {
        name : 'pink',
        age : 18,
        sex : '男',
        sayHi : function(){
            alert('大家好啊~');
        }
    };
    ```

    上述代码中 star即是创建的对象。

- 对象的使用
  - 对象的属性
    - 对象中存储**具体数据**的 "键值对"中的 "键"称为对象的属性，即对象中存储具体数据的项
  - 对象的方法
    - 对象中存储**函数**的 "键值对"中的 "键"称为对象的方法，即对象中存储函数的项
  - 访问对象的属性
    - 对象里面的属性调用 : 对象.属性名 ， 这个小点 . 就理解为“ 的 ”
    - 对象里面属性的另一种调用方式 : 对象['属性名']，注意方括号里面的属性必须加引号

      示例代码如下：

```
console.log(star.name)      // 调用名字属性
console.log(star['name'])   // 调用名字属性
```

- 调用对象的方法
  - 对象里面的方法调用：对象.方法名()，注意这个方法名字后面一定加括号

    示例代码如下：

    ```
    star.sayHi();               // 调用 sayHi 方法,注意，一定不要忘记带后面的括号
    ```

- 变量、属性、函数、方法总结

  属性是对象的一部分，而变量不是对象的一部分，变量是单独存储数据的容器

  - 变量：单独声明赋值，单独存在
  - 属性：对象里面的变量称为属性，不需要声明，用来描述该对象的特征

> 方法是对象的一部分，函数不是对象的一部分，函数是单独封装操作的容器
>
> - 函数：单独存在的，通过“函数名()”的方式就可以调用
> - 方法：对象里面的函数称为方法，方法不需要声明，使用“对象.方法名()”的方式就可以调用，方法用来描述该对象的行为和功能。

- 利用 new Object 创建对象
  - 创建空对象

    ```
    var andy = new Obect();
    ```

    通过内置构造函数Object创建对象，此时andy变量已经保存了创建出来的空对象
  - 给空对象添加属性和方法
    - 通过对象操作属性和方法的方式，来为对象增加属性和方法

      示例代码如下：

      ```
      andy.name = 'pink';
      andy.age = 18;
      andy.sex = '男';
      andy.sayHi = function(){
          alert('大家好啊~');
      }
      ```

      注意：

      - Object()： 第一个字母大写
      - new Object()： 需要 new 关键字

        - 使用的格式：对象.属性 = 值;
- 利用构造函数创建对象
    - 构造函数
        - 构造函数：是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与 new 运算符一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。
        - 构造函数的封装格式：

```
function 构造函数名(形参1,形参2,形参3) {
      this.属性名1 = 参数1;
      this.属性名2 = 参数2;
      this.属性名3 = 参数3;
      this.方法名 = 函数体;
}
```

        - 构造函数的调用格式

```
var obj = new 构造函数名(实参1，实参2，实参3)
```

          以上代码中，obj即接收到构造函数创建出来的对象。
        - 注意事项
            1. 构造函数约定**首字母大写**。
            2. 函数内的属性和方法前面需要添加 **this** ，表示当前对象的属性和方法。
            3. 构造函数中**不需要 return 返回结果**。
            4. 当我们创建对象的时候，**必须用 new 来调用构造函数**。
        - 其他

          构造函数，如 Stars()，抽象了对象的公共部分，封装到了函数里面，它泛指某一大类（class）
          创建对象，如 new Stars()，特指某一个，通过 new 关键字创建对象的过程我们也称为对象实例化
- new关键字的作用
    1. 在构造函数代码开始执行之前，创建一个空对象；
    2. 修改this的指向，把this指向创建出来的空对象；
    3. 执行函数的代码
    4. 在函数完成之后，返回this---即创建出来的对象

5.3 遍历对象

```
for...in 语句用于对数组或者对象的属性进行循环操作。

其语法如下：
```

```
for (变量 in 对象名字) {
    // 在此执行代码
}
```

语法中的变量是自定义的，它需要符合命名规范，通常我们会将这个变量写为 `k` 或者 `key`。

```
for (var k in obj) {
    console.log(k);      // 这里的 k 是属性名
    console.log(obj[k]); // 这里的 obj[k] 是属性值
}
```