JavaScript基础

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.JavaScript作用域 ： 就是代码名字（变量）在某个范围内起作用和效果 目的是为了提高程序的可靠性更重要的是减少命名冲突
        // 2. js的作用域（es6）之前 ： 全局作用域  局部作用域
        // 3. 全局作用域： 整个script标签 或者是一个单独的js文件
        var num = 10;
        var num = 30;
        console.log(num);
        // 4. 局部作用域（函数作用域） 在函数内部就是局部作用域 这个代码的名字只在函数内部起效果和作用
        function fn() {
            // 局部作用域
            var num = 20;
            console.log(num);
        }
        fn();
```

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。

```
// 变量的作用域： 根据作用域的不同我们变量分为全局变量和局部变量
        // 1. 全局变量： 在全局作用域下的变量 在全局下都可以使用
        // 注意 如果在函数内部 没有声明直接赋值的变量也属于全局变量
        var num = 10; // num就是一个全局变量
        console.log(num);
        function fn() {
            console.log(num);
        }
```

```
        fn();
        // console.log(aru);

        // 2. 局部变量    在局部作用域下的变量    后者在函数内部的变量就是 局部变量
        // 注意： 函数的形参也可以看做是局部变量
        function fun(aru) {
            var num1 = 10; // num1就是局部变量 只能在函数内部使用
            num2 = 20;
        }
        fun();
        // console.log(num1);
        // console.log(num2);
        // 3. 从执行效率来看全局变量和局部变量
        // (1) 全局变量只有浏览器关闭的时候才会销毁，比较占内存资源
        // (2) 局部变量 当我们程序执行完毕就会销毁， 比较节约内存资源
```

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：

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）

```
// js中没有块级作用域  js的作用域： 全局作用域  局部作用域  现阶段我们js 没有 块级作用域
        // 我们js 也是在 es6 的时候新增的块级作用域
        // 块级作用域 {}   if {}  for {}
        // java
        // if(xx) {
        //     int num = 10;
        // }
        // 外面的是不能调用num的
        if (3 < 5) {
            var num = 10;
        }
        console.log(num);
```

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

```
        // 作用域链  ：  内部函数访问外部函数的变量，采取的是链式查找的方式来决定取那个值  这种结构我们称为作用域链   就近原则
        var num = 10;
        function fn() { // 外部函数
            var num = 20;
            function fun() { // 内部函数
                console.log(num);
            }
            fun();
        }
        fn();
```

> 作用域链：采取就近原则的方式来查找变量最终的值。

```
   // 案例1  :
        function f1() {

            var num = 123;
```

```
        function f2() {
            var num = 0;
            console.log(num); // 站在目标出发，一层一层的往外查找
        }
        f2();
    }
    var num = 456;
    f1();
    // 案例2
    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);
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('hello');
}
```

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

```
        console.log(num); // undefined
        var num = 10;
        // 相当于执行了以下代码
        // var num;
        // console.log(num);
        // num = 10;
        // 3问
        function fn() {
            console.log(11);
        }
        fn();


        fun(); // 报错
        var fun = function() {
                console.log(22);
            }
            // 函数表达式 调用必须写在函数表达式的下面
            // 相当于执行了以下代码
            // var fun;
```

```
            // fun();
            // fun = function() {
            //         console.log(22);

        //     }
        // 1. 我们js引擎运行js 分为两步：  预解析  代码执行
        // (1). 预解析 js引擎会把js 里面所有的 var  还有 function 提升到当前作用域的最前面
        // (2). 代码执行  按照代码书写的顺序从上往下执行
        // 2. 预解析分为 变量预解析（变量提升） 和 函数预解析（函数提升）
        // (1) 变量提升 就是把所有的变量声明提升到当前的作用域最前面  不提升赋值操作
        // (2) 函数提升 就是把所有的函数声明提升到当前作用域的最前面  不调用函数
```

```
// 预解析案例
        // 案例1
         var num = 10;
         fun();
         function fun() {
             console.log(num);
             var num = 20;
         }



         //相当于执行了以下操作
         var num;
         function fun() {
            var num;
            console.log(num);
             num = 20;
          }
         num = 10;
         fun();

        // // 案例2
        var num = 10;
        function fn() {
          console.log(num);
          var num = 20;
          console.log(num);
        }
        // fn();
        // // 相当于以下代码
         var num;
        function fn() {
            var num;
            console.log(num);
            num = 20;
            console.log(num);
         }
       num = 10;
        fn();
       //  案例3
```

```
 var a = 18;
 f1();
function f1() {
    var b = 9;
     console.log(a);
    console.log(b);
     var a = '123';
 }
// 相当于以下代码
// var a;

 function f1() {
    var b;
     var a;
     b = 9;
     console.log(a);
     console.log(b);
     a = '123';
 }
 a = 18;
 f1();
// 案例4
f1();
console.log(c);
console.log(b);
console.log(a);

function f1() {
    var a = b = c = 9;
    console.log(a);
    console.log(b);
    console.log(c);
}
// 以下代码
 function f1() {
    var a;
    a = b = c = 9;
     // 相当于 var  a  = 9; b = 9; c = 9; b 和 c 直接赋值 没有var 声明 当 全局变量看
     // 集体声明  var a = 9, b = 9, c = 9;
     console.log(a);
     console.log(b);
     console.log(c);
 }
 f1();
 console.log(c);
 console.log(b);
 console.log(a);
```

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”的个人的信息保存在数组中的方式为：

```
var person = ["张山", "男", 15, "小李", "女", 18];
```

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：

```
// 1.利用对象字面量创建对象 {}
   // var obj = {};  // 创建了一个空的对象
   var obj = {
           uname: '小李',
           age: 18,
           sex: '男',
           sayHi: function() {
               console.log('hi~');
           }
       }
       // (1) 里面的属性或者方法我们采取键值对的形式  键 属性名 : 值  属性值
       // (2) 多个属性或者方法中间用逗号隔开的
       // (3) 方法冒号后面跟的是一个匿名函数
       // 2. 使用对象
       // (1). 调用对象的属性 我们采取 对象名.属性名 . 我们理解为 的
   console.log(obj.uname);
   // (2). 调用属性还有一种方法 对象名['属性名']
   console.log(obj['age']);
   // (3) 调用对象的方法 sayHi   对象名.方法名() 千万别忘记添加小括号
   obj.sayHi();
```

JS中的对象表达结构更清晰，更强大。

```
// 变量、属性、函数、方法的区别
```

```
// 1.变量和属性的相同点 他们都是用来存储数据的
var num = 10;
var obj = {
    age: 18,
    fn: function() {
    }
}
function fn() {
}
console.log(obj.age);
// console.log(age);
// 变量 单独声明并赋值  使用的时候直接写变量名 单独存在
// 属性 在对象里面的不需要声明的 使用的时候必须是 对象.属性
// 2. 函数和方法的相同点 都是实现某种功能  做某件事
// 函数是单独声明 并且调用的 函数名() 单独存在的
// 方法 在对象里面 调用的时候 对象.方法()
```

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 : '小张',
        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

```
// 利用 new Object 创建对象
        var obj = new Object(); // 创建了一个空的对象
        obj.uname = '张三疯';
        obj.age = 18;
        obj.sex = '男';
        obj.sayHi = function() {
                console.log('hi~');

            }
            // (1) 我们是利用 等号 = 赋值的方法 添加对象的属性和方法
            // (2) 每个属性和方法之间用 分号结束
        console.log(obj.uname);
        console.log(obj['sex']);
        obj.sayHi();
```

```

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。

```
// 我们为什么需要使用构造函数
        // 就是因我们前面两种创建对象的方式一次只能创建一个对象
        var xz = {
            uname: '小张',
            age: 55,
            sing: function() {
                console.log('一生爱你');

            }
        }
        var xl = {
                uname: '小李',
                age: 58,
                sing: function() {
                    console.log('生活');

                }
            }
            // 因为我们一次创建一个对象，里面很多的属性和方法是大量相同的 我们只能复制
            // 因此我们可以利用函数的方法 重复这些相同的代码 我们就把这个函数称为 构造函
数
            // 又因为这个函数不一样，里面封装的不是普通代码，而是 对象
            // 构造函数 就是把我们对象里面一些相同的属性和方法抽象出来封装到函数里面
```

```
// 利用构造函数创建对象
        // 我们需要创建四大天王的对象  相同的属性： 名字 年龄 性别  相同的方法： 唱歌
        // 构造函数的语法格式
        // function 构造函数名() {
        //     this.属性 = 值;
        //     this.方法 = function() {}
        // }
        // new 构造函数名();
        function Star(uname, age, sex) {
            this.name = uname;
            this.age = age;
            this.sex = sex;
            this.sing = function(sang) {
                console.log(sang);

            }
        }
        var xz = new Star('小张', 18, '男'); // 调用函数返回的是一个对象
        // console.log(typeof xz);
        console.log(xz.name);
        console.log(xz['sex']);
        xz.sing('自由');
        var xl = new Star('小李', 19, '男');
        console.log(xl.name);
        console.log(xl.age);
        zxy.sing('一生有你')
```

```
// 1. 构造函数名字首字母要大写
        // 2. 我们构造函数不需要return 就可以返回结果
        // 3. 我们调用构造函数 必须使用 new
        // 4. 我们只要new Star() 调用函数就创建一个对象 ldh  {}
        // 5. 我们的属性和方法前面必须添加 this

    // 构造函数和对象
        // 1. 构造函数  明星 泛指的某一大类  它类似于 java 语言里面的  类(class)
        function Star(uname, age, sex) {
            this.name = uname;
            this.age = age;
            this.sex = sex;
            this.sing = function(sang) {
                console.log(sang);
            }
        }
        // 2. 对象 特指 是一个具体的事物 xz ==  {name: "小张", age: 18, sex: "男", sing: f}
        var xz = new Star('小张', 18, '男'); // 调用函数返回的是一个对象
        console.log(xz);
        // 3. 我们利用构造函数创建对象的过程我们也称为对象的实例化
```

- 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

```
// new关键字执行过程
      // 1. new 构造函数可以在内存中创建了一个空的对象
      // 2. this 就会指向刚才创建的空对象
      // 3. 执行构造函数里面的代码 给这个空对象添加属性和方法
      // 4. 返回这个对象
      function Star(uname, age, sex) {
          this.name = uname;
          this.age = age;
          this.sex = sex;
          this.sing = function(sang) {
              console.log(sang);
          }
      }
      var xz = new Star('小张', 18, '男');
```

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。

```
    // 遍历对象
        var obj = {
                name: '张小小',
                age: 18,
                sex: '男',
                fn: function() {}

            }
```

```
            // console.log(obj.name);
            // console.log(obj.age);
            // console.log(obj.sex);
            // for in 遍历我们的对象
            // for (变量 in 对象) {

        // }
        for (var k in obj) {
            console.log(k); // k 变量 输出  得到的是 属性名
            console.log(obj[k]); // obj[k] 得到是 属性值

        }
        // 我们使用 for in 里面的变量 我们喜欢写 k  或者  key
```

JavaScript基础

1 - 内置对象

1.1 内置对象

JavaScript 中的对象分为3种：**自定义对象 、内置对象、 浏览器对象** 前面两种对象是JS 基础 内容，属于ECMAScript； 第三个浏览器对象属于 JS 独有的， JS API 讲解内置对象就是指 JS 语言自带的一些对象，这些对象供开发者使用，并提供了一些常用的或是**最基本而必要的功能**（属性和方法），内置对象最大的优点就是帮助我们快速开发

JavaScript 提供了多个内置对象：Math、 Date 、Array、String等

1.2 查文档

查找文档：学习一个内置对象的使用，只要学会其常用成员的使用即可，我们可以通过查文档学习，可以通过MDN/W3C来查询。 Mozilla 开发者网络（MDN）提供了有关开放网络技术（Open Web）的信息，包括 HTML、CSS 和万维网及 HTML5 应用的 API。 MDN:https://developer.mozilla.org/zh-CN/

1.3 Math对象

Math 对象不是构造函数，它具有数学常数和函数的属性和方法。跟数学相关的运算（求绝对值，取整、最大值等）可以使用 Math 中的成员。

属性、方法名	功能
Math.PI	圆周率
Math.floor()	向下取整
Math.ceil()	向上取整
Math.round()	四舍五入版 就近取整　注意 -3.5　结果是 -3
Math.abs()	绝对值
Math.max()/Math.min()	求最大和最小值
Math.random()	获取范围在[0,1)内的随机值

注意：上面的方法使用时必须带括号

获取指定范围内的随机整数：

```
function getRandom(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

```
// Math数学对象 不是一个构造函数 ，所以我们不需要new 来调用 而是直接使用里面的属性和方法即可
        console.log(Math.PI); // 一个属性 圆周率
        console.log(Math.max(1, 99, 3)); // 99
        console.log(Math.max(-1, -10)); // -1
        console.log(Math.max(1, 99, 'green')); // NaN
        console.log(Math.max()); // -Infinity
```

```
    // 利用对象封装自己的数学对象　里面有 PI 最大值和最小值
        var myMath = {
            PI: 3.141592653,
            max: function() {
                var max = arguments[0];
                for (var i = 1; i < arguments.length; i++) {
                    if (arguments[i] > max) {
                        max = arguments[i];
                    }
                }
                return max;
            },
            min: function() {
                var min = arguments[0];
                for (var i = 1; i < arguments.length; i++) {
                    if (arguments[i] < min) {
                        min = arguments[i];
                    }
```

```
            }
            return min;
        }
    }
    console.log(myMath.PI);
    console.log(myMath.max(1, 5, 9));
    console.log(myMath.min(1, 5, 9));
```

```
    // 1.绝对值方法
    console.log(Math.abs(1)); // 1
    console.log(Math.abs(-1)); // 1
    console.log(Math.abs('-1')); // 隐式转换 会把字符串型 -1 转换为数字型
    console.log(Math.abs('red')); // NaN
    // 2.三个取整方法
    // (1) Math.floor()   地板 向下取整  往最小了取值
    console.log(Math.floor(1.1)); // 1
    console.log(Math.floor(1.9)); // 1
    // (2) Math.ceil()   ceil 天花板 向上取整  往最大了取值
    console.log(Math.ceil(1.1)); // 2
    console.log(Math.ceil(1.9)); // 2
    // (3) Math.round()   四舍五入  其他数字都是四舍五入，但是 .5 特殊 它往大了取
    console.log(Math.round(1.1)); // 1
    console.log(Math.round(1.5)); // 2
    console.log(Math.round(1.9)); // 2
    console.log(Math.round(-1.1)); // -1
    console.log(Math.round(-1.5)); // 这个结果是 -1
```

```
// 1.Math对象随机数方法   random() 返回一个随机的小数  0 =< x < 1
    // 2. 这个方法里面不跟参数
    // 3. 代码验证
    console.log(Math.random());
    // 4. 我们想要得到两个数之间的随机整数 并且 包含这2个整数
    // Math.floor(Math.random() * (max - min + 1)) + min;
    function getRandom(min, max) {
        return Math.floor(Math.random() * (max - min + 1)) + min;
    }
    console.log(getRandom(1, 10));
    // 5. 随机点名
    var arr = ['张三', '张三丰', '张三疯子', '李四', '李思思', 'pink老师'];
    // console.log(arr[0]);
    console.log(arr[getRandom(0, arr.length - 1)]);
```

```
// 猜数字游戏
    // 1.随机生成一个1~10 的整数  我们需要用到 Math.random() 方法。
    // 2.需要一直猜到正确为止，所以需要一直循环。
    // 3.while 循环更简单
```

```
// 4.核心算法：使用 if  else if 多分支语句来判断大于、小于、等于。
function getRandom(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
var random = getRandom(1, 10);
while (true) { // 死循环
    var num = prompt('你来猜？ 输入1~10之间的一个数字');
    if (num > random) {
        alert('你猜大了');
    } else if (num < random) {
        alert('你猜小了');
    } else {
        alert('你好帅哦，猜对了');
        break; // 退出整个循环结束程序
    }

}
// 要求用户猜 1~50之间的一个数字 但是只有 10次猜的机会
```

1.4 日期对象

Date 对象和 Math 对象不一样，Date是一个构造函数，所以使用时需要实例化后才能使用其中具体方法和属性。Date 实例用来处理日期和时间

使用Date实例化日期对象

获取当前时间必须实例化：

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

获取指定时间的日期对象

```
var future = new Date('2019/5/1');
```

注意：如果创建实例时并未传入参数，则得到的日期对象是当前时间对应的日期对象

使用Date实例的方法和属性

方法名	说明	代码
getFullYear()	获取当年	dObj.getFullYear()
getMonth()	获取当月（0-11）	dObj.getMonth()
getDate()	获取当天日期	dObj.getDate()
getDay()	获取星期几 （周日0 到周六6）	dObj.getDay()
getHours()	获取当前小时	dObj.getHours()
getMinutes()	获取当前分钟	dObj.getMinutes()
getSeconds()	获取当前秒钟	dObj.getSeconds()

通过Date实例获取总毫秒数

总毫秒数的含义

基于1970年1月1日（世界标准时间）起的毫秒数（1秒=1000毫秒）

获取总毫秒数

```
// 实例化Date对象
var now = new Date();
// 1. 用于获取对象的原始值
console.log(date.valueOf())
console.log(date.getTime())
// 2. 简单写可以这么做
var now = + new Date();
// 3. HTML5中提供的方法，有兼容性问题
var now = Date.now();
```

```
// Date() 日期对象  是一个构造函数 必须使用new 来调用创建我们的日期对象
        var arr = new Array(); // 创建一个数组对象
        var obj = new Object(); // 创建了一个对象实例
        // 1. 使用Date  如果没有参数 返回当前系统的当前时间
        var date = new Date();
        console.log(date);
        // 2. 参数常用的写法  数字型  2019, 10, 01  或者是 字符串型 '2019-10-1 8:8:8'
        var date1 = new Date("202";
        console.log(date1); // 返回的是 11月 不是 10月
        var date2 = new Date('2019-10-1 8:8:8');
        console.log(date2);
```

```
// 格式化日期 年月日
    var date = new Date();
    console.log(date.getFullYear()); // 返回当前日期的年  2019
    console.log(date.getMonth() + 1); // 月份 返回的月份小1个月    记得月份+1 呦
    console.log(date.getDate()); // 返回的是 几号
    console.log(date.getDay()); // 3  周一返回的是 1 周六返回的是 6 但是 周日返回的是 0
    // 我们写一个 2019年 5月 1日 星期三
    var year = date.getFullYear();
    var month = date.getMonth() + 1;
    var dates = date.getDate();
    var arr = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六'];
    var day = date.getDay();
    console.log('今天是：' + year + '年' + month + '月' + dates + '日 ' + arr[day]);
```

```
// 格式化日期 时分秒
    var date = new Date();
    console.log(date.getHours()); // 时
    console.log(date.getMinutes()); // 分
    console.log(date.getSeconds()); // 秒
    // 要求封装一个函数返回当前的时分秒 格式 08:08:08
    function getTimer() {
        var time = new Date();
        var h = time.getHours();
        h = h < 10 ? '0' + h : h;
        var m = time.getMinutes();
        m = m < 10 ? '0' + m : m;
        var s = time.getSeconds();
        s = s < 10 ? '0' + s : s;
        return h + ':' + m + ':' + s;
    }
    console.log(getTimer());
```

```
// 获得Date总的毫秒数(时间戳)  不是当前时间的毫秒数 而是距离1970年1月1号过了多少毫秒数
    // 1. 通过 valueOf()  getTime()
    var date = new Date();
    console.log(date.valueOf()); // 就是 我们现在时间 距离1970.1.1 总的毫秒数
    console.log(date.getTime());
    // 2. 简单的写法 (最常用的写法)
    var date1 = +new Date(); // +new Date()  返回的就是总的毫秒数
    console.log(date1);
    // 3. H5 新增的 获得总的毫秒数
    console.log(Date.now());
```

```
// 倒计时效果

    // 1.核心算法：输入的时间减去现在的时间就是剩余的时间，即倒计时 ，但是不能拿着时分秒相减，比如
```

```
05 分减去25分，结果会是负数的。
        // 2.用时间戳来做。用户输入时间总的毫秒数减去现在时间的总的毫秒数，得到的就是剩余时间的毫秒数。
        // 3.把剩余时间总的毫秒数转换为天、时、分、秒 （时间戳转换为时分秒）
        // 转换公式如下：
        //  d = parseInt(总秒数/ 60/60 /24);    //  计算天数
        //  h = parseInt(总秒数/ 60/60 %24)   //   计算小时
        //  m = parseInt(总秒数 /60 %60 );     //   计算分数
        //  s = parseInt(总秒数%60);            //   计算当前秒数
        function countDown(time) {

            var nowTime = +new Date(); // 返回的是当前时间总的毫秒数


            var inputTime = +new Date(time); // 返回的是用户输入时间总的毫秒数


            var times = (inputTime - nowTime) / 1000; // times是剩余时间总的秒数



            var d = parseInt(times / 60 / 60 / 24); // 天



            d = d < 10 ? '0' + d : d;

            var h = parseInt(times / 60 / 60 % 24); //时

            h = h < 10 ? '0' + h : h;


            var m = parseInt(times / 60 % 60); // 分

            m = m < 10 ? '0' + m : m;

            var s = parseInt(times % 60); // 当前的秒

            s = s < 10 ? '0' + s : s;


            return d + '天' + h + '时' + m + '分' + s + '秒';
        }
        console.log(countDown('2020-12-31 18:00:00'));
        var date = new Date();
        console.log(date);
```

1.5 数组对象

创建数组的两种方式（字面量方式，new Array()）

第一种方式：字面量方式

```
var arr = [1,"test",true];
```

第二种方式：new Array( )

```
var arr = new Array( );
```

注意：上面代码中arr创建出的是一个空数组，如果需要使用构造函数Array创建非空数组，可以在创建数组时传入参数参数传递规则如下：

如果只传入一个参数，则参数规定了数组的长度

如果传入了多个参数，则参数称为数组的元素

```
    // 创建数组的两种方式
        // 1. 利用数组字面量
        var arr = [1, 2, 3];
        console.log(arr[0]);
        // 2. 利用new Array()
        // var arr1 = new Array();  // 创建了一个空的数组
        // var arr1 = new Array(2);  // 这个2 表示 数组的长度为 2  里面有2个空的数组元素
        var arr1 = new Array(2, 3); // 等价于 [2,3]  这样写表示 里面有2个数组元素 是 2和3
        console.log(arr1);
```

检测是否为数组

instanceof 运算符

instanceof 可以判断一个对象是否是某个构造函数的实例

```
var arr = [1, 23];
var obj = {};
console.log(arr instanceof Array); // true、
console.log(obj instanceof Array); // false
```

Array.isArray()

Array.isArray()用于判断一个对象是否为数组，isArray() 是 HTML5 中提供的方法

```
var arr = [1, 23];
var obj = {};
console.log(Array.isArray(arr));   // true
console.log(Array.isArray(obj));   // false
```

```
    // 翻转数组
       function reverse(arr) {
           // if (arr instanceof Array) {
           if (Array.isArray(arr)) {
               var newArr = [];
               for (var i = arr.length - 1; i >= 0; i--) {
                   newArr[newArr.length] = arr[i];
               }
```

```
            return newArr;
        } else {
            return 'error 这个参数要求必须是数组格式 [1,2,3]'
        }
    }
    console.log(reverse([1, 2, 3]));
    console.log(reverse(1, 2, 3));
    // 检测是否为数组
    // (1) instanceof  运算符 它可以用来检测是否为数组
    var arr = [];
    var obj = {};
    console.log(arr instanceof Array);
    console.log(obj instanceof Array);
    // (2) Array.isArray(参数);  H5新增的方法  ie9以上版本支持
    console.log(Array.isArray(arr));
    console.log(Array.isArray(obj));
```

添加删除数组元素的方法

数组中有进行增加、删除元素的方法，部分方法如下表

方法名	说明	返回值
push(参数1....)	末尾添加一个或多个元素，注意修改原数组	并返回新的长度
pop()	删除数组最后一个元素，把数组长度减 1 无参数、修改原数组	返回它删除的元素的值
unshift(参数1...)	向数组的开头添加一个或更多元素，注意修改原数组	并返回新的长度
shift()	删除数组的第一个元素，数组长度减 1 无参数、修改原数组	并返回第一个元素的值

注意：push、unshift为增加元素方法；pop、shift为删除元素的方法

```
// 添加删除数组元素方法
    // 1. push() 在我们数组的末尾 添加一个或者多个数组元素   push  推
    var arr = [1, 2, 3];
    // arr.push(4, '45');
    console.log(arr.push(4, '85'));

    console.log(arr);
    // (1) push 是可以给数组追加新的元素
    // (2) push() 参数直接写 数组元素就可以了
    // (3) push完毕之后，返回的结果是 新数组的长度
    // (4) 原数组也会发生变化
    // 2. unshift 在我们数组的开头 添加一个或者多个数组元素
    console.log(arr.unshift('red', 'purple'));

    console.log(arr);
    // (1) unshift是可以给数组前面追加新的元素
```

```
        // (2) unshift() 参数直接写 数组元素就可以了
        // (3) unshift完毕之后，返回的结果是 新数组的长度
        // (4) 原数组也会发生变化
        // 3. pop() 它可以删除数组的最后一个元素
        console.log(arr.pop());
        console.log(arr);
        // (1) pop是可以删除数组的最后一个元素 记住一次只能删除一个元素
        // (2) pop() 没有参数
        // (3) pop完毕之后，返回的结果是 删除的那个元素
        // (4) 原数组也会发生变化
        // 4. shift() 它可以删除数组的第一个元素
        console.log(arr.shift());
        console.log(arr);
        // (1) shift是可以删除数组的第一个元素 记住一次只能删除一个元素
        // (2) shift() 没有参数
        // (3) shift完毕之后，返回的结果是 删除的那个元素
        // (4) 原数组也会发生变化
```

```
  // 有一个包含工资的数组[1500, 1200, 2000, 2100, 1800]，要求把数组中工资超过2000的删除，剩余的放到新数组里面
        var arr = [1500, 1200, 2000, 2100, 1800];
        var newArr = [];
        for (var i = 0; i < arr.length; i++) {
            if (arr[i] < 2000) {
                // newArr[newArr.length] = arr[i];
                newArr.push(arr[i]);
            }
        }
        console.log(newArr);
```

数组排序

数组中有对数组本身排序的方法，部分方法如下表

方法名	说明	是否修改原数组
reverse()	颠倒数组中元素的顺序,无参数	该方法会改变原来的数组　返回新数组
sort()	对数组的元素进行排序	该方法会改变原来的数组 返回新数组

注意：sort方法需要传入参数来设置升序、降序排序

如果传入“function(a,b){ return a-b;}”，则为升序

如果传入“function(a,b){ return b-a;}”，则为降序

```
// 数组排序
  // 1. 翻转数组
  var arr = ['pink', 'red', 'blue'];
  arr.reverse();
  console.log(arr);
  // 2. 数组排序（冒泡排序）
  var arr1 = [13, 4, 77, 1, 7];
  arr1.sort(function(a, b) {
      //  return a - b; 升序的顺序排列
      return b - a; // 降序的顺序排列
  });
  console.log(arr1);
```

数组索引方法

数组中有获取数组指定元素索引值的方法，部分方法如下表

方法名	说明	返回值
indexOf()	数组中查找给定元素的第一个索引	如果存在返回索引号 如果不存在，则返回-1。
lastIndexOf()	在数组中的最后一个的索引，	如果存在返回索引号 如果不存在，则返回-1。

```
// 返回数组元素索引号方法  indexOf(数组元素)  作用就是返回该数组元素的索引号 从前面开始查找
  // 它只返回第一个满足条件的索引号
  // 它如果在该数组里面找不到元素，则返回的是 -1
  // var arr = ['red', 'green', 'blue', 'pink', 'blue'];
  var arr = ['red', 'green', 'pink'];
  console.log(arr.indexOf('blue'));
  // 返回数组元素索引号方法  lastIndexOf(数组元素)  作用就是返回该数组元素的索引号 从后面开始查找
  var arr = ['red', 'green', 'blue', 'pink', 'blue'];
  console.log(arr.lastIndexOf('blue')); // 4
```

```
// 数组去重 ['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'] 要求去除数组中重复的元素。
  // 1.目标： 把旧数组里面不重复的元素选取出来放到新数组中， 重复的元素只保留一个， 放到新数组中去重。

  // 2.核心算法： 我们遍历旧数组， 然后拿着旧数组元素去查询新数组， 如果该元素在新数组里面没有出现过， 我们就添加， 否则不添加。

  // 3.我们怎么知道该元素没有存在？ 利用 新数组.indexOf(数组元素) 如果返回时 - 1 就说明 新数组里面没有改元素
  // 封装一个 去重的函数 unique 独一无二的
  function unique(arr) {
      var newArr = [];
      for (var i = 0; i < arr.length; i++) {
          if (newArr.indexOf(arr[i]) === -1) {
              newArr.push(arr[i]);
          }
      }

      return newArr;
```

```
        }
        // var demo = unique(['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'])
        var demo = unique(['blue', 'green', 'blue'])
        console.log(demo);
```

数组转换为字符串

数组中有把数组转化为字符串的方法，部分方法如下表

方法名	说明	返回值
toString()	把数组转换成字符串，逗号分隔每一项	返回一个字符串
join('分隔符')	方法用于把数组中的所有元素转换为一个字符串。	返回一个字符串

注意：join方法如果不传入参数，则按照“,”拼接元素

其他方法

数组中还有其他操作方法，同学们可以在课下自行查阅学习

方法名	说明	返回值
concat()	连接两个或多个数组 不影响原数组	返回一个新的数组
slice()	数组截取slice(begin, end)	返回被截取项目的新数组
splice()	数组删除splice(第几个开始,要删除个数)	返回被删除项目的新数组 注意，这个会影响原数组

```
// 数组转换为字符串
    // 1. toString() 将我们的数组转换为字符串
    var arr = [1, 2, 3];
    console.log(arr.toString()); // 1,2,3
    // 2. join(分隔符)
    var arr1 = ['green', 'blue', 'pink'];
    console.log(arr1.join()); // green,blue,pink
    console.log(arr1.join('-')); // green-blue-pink
    console.log(arr1.join('&')); // green&blue&pink
    var arr = ['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'];
    var aa = ["中国", "美国"];
    var bb = aa.concat(arr);
    console.log(bb);
```

1.6 字符串对象

基本包装类型

为了方便操作基本数据类型，JavaScript 还提供了三个特殊的引用类型：String、Number和 Boolean。

基本包装类型就是把简单数据类型包装成为复杂数据类型，这样基本数据类型就有了属性和方法。

```
// 下面代码有什么问题？
var str = 'andy';
console.log(str.length);
```

按道理基本数据类型是没有属性和方法的，而对象才有属性和方法，但上面代码却可以执行，这是因为

js 会把基本数据类型包装为复杂数据类型，其执行过程如下 ：

```
// 1. 生成临时变量，把简单类型包装为复杂数据类型
var temp = new String('andy');
// 2. 赋值给我们声明的字符变量
str = temp;
// 3. 销毁临时变量
temp = null;
```

```
        // 基本包装类型
        var str = 'hello';
        console.log(str.length);
        // 对象 才有 属性和方法    复杂数据类型才有 属性和方法
        // 简单数据类型为什么会有length 属性呢？
        // 基本包装类型：  就是把简单数据类型 包装成为了 复杂数据类型
        // (1) 把简单数据类型包装为复杂数据类型
        var temp = new String('hello');
        // (2) 把临时变量的值 给 str
        str = temp;
        // (3) 销毁这个临时变量
        temp = null;
```

字符串的不可变

指的是里面的值不可变，虽然看上去可以改变内容，但其实是地址变了，内存中新开辟了一个内存空间。

当重新给字符串变量赋值的时候，变量之前保存的字符串不会被修改，依然在内存中重新给字符串赋值，会重新在内存中开辟空间，这个特点就是字符串的不可变。 由于字符串的不可变，在**大量拼接字符串**的时候会有效率问题

根据字符返回位置

字符串通过基本包装类型可以调用部分方法来操作字符串，以下是返回指定字符的位置的方法：

方法名	说明
indexOf('要查找的字符', 开始的位置)	返回指定内容在元字符串中的位置， 如果找不到就返回 -1， 开始的位置是 index 索引号
lastIndexOf()	从后往前找，只找第一个匹配的

案例：查找字符串"abcoefoxyozzopp"中所有o出现的位置以及次数

1. 先查找第一个o出现的位置
2. 然后 只要indexOf 返回的结果不是 -1 就继续往后查找 . 因为indexOf 只能查找到第一个，所以后面的查找，利用第二个参数，当前索引加1，从而继续查找

```
// 字符串的不可变性
    var str = 'andy';
    console.log(str);
    str = 'red';
    console.log(str);
    // 因为我们字符串的不可变所以不要大量的拼接字符串
    var str = '';
    for (var i = 1; i <= 1000000000; i++) {
        str += i;
    }
    console.log(str);
```

根据位置返回字符

字符串通过基本包装类型可以调用部分方法来操作字符串，以下是根据位置返回指定位置上的字符：

方法名	说明	使用
charAt(index)	返回指定位置的字符(index 字符串的索引号)	str.charAt(0)
charCodeAt(index)	获取指定位置处字符的ASCII码 (index索引号)	str.charCodeAt(0)
str[index]	获取指定位置处字符	HTML5，IE8+支持 和charAt()等效

在上述方法中，charCodeAt方法返回的是指定位置上字符对应的ASCII码，ASCII码对照表如下：

ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位 / 低四位		ASCII控制字符 0000						ASCII控制字符 0001						ASCII打印字符 0010				ASCII打印字符 0011				ASCII打印字符 0100				ASCII打印字符 0101				ASCII打印字符 0110				ASCII打印字符 0111		
		0						1						2		3		4		5		6		7										
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl						
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p									
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q									
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r									
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s									
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t									
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u									
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v									
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w									
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x									
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y									
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z									
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{									
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|								
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}									
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~									
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	*Backspace 代码：DEL								

案例：判断一个字符串 'abcoefoxyozzopp' 中出现次数最多的字符，并统计其次数

1. 核心算法：利用 charAt( ) 遍历这个字符串
2. 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1

   . 遍历对象，得到最大值和该字符

   注意：在遍历的过程中，把字符串中的每个字符作为对象的属性存储在对象总，对应的属性值是该字符出现的次数

```
// 字符串对象  根据字符返回位置  str.indexOf('要查找的字符', [起始的位置])
        var str = '改革春风吹满地，春天来了';
        console.log(str.indexOf('春'));
        console.log(str.indexOf('春',3)); // 从索引号是 3的位置开始往后查找
```

```
// 查找字符串"abcoefoxyozzopp"中所有o出现的位置以及次数

        // 核心算法：先查找第一个o出现的位置
        // 然后 只要indexOf 返回的结果不是 -1 就继续往后查找
        // 因为indexOf 只能查找到第一个，所以后面的查找，一定是当前索引加1，从而继续查找
        var str = "oabcoefoxyozzopp";
        var index = str.indexOf('o');
        var num = 0;
        // console.log(index);
        while (index !== -1) {
            console.log(index);
            num++;
            index = str.indexOf('o', index);
        }
        console.log('o出现的次数是: ' + num);
        // 课后作业 ['red', 'blue', 'red', 'green', 'pink','red'], 求 red 出现的位置和次数
```

```
// 根据位置返回字符
        // 1. charAt(index) 根据位置返回字符
        var str = 'andy';
        console.log(str.charAt(3));
        // 遍历所有的字符
        for (var i = 0; i < str.length; i++) {
            console.log(str.charAt(i));
        }
        // 2. charCodeAt(index)  返回相应索引号的字符ASCII值 目的： 判断用户按下了那个键
        console.log(str.charCodeAt(0)); // 97
        // 3. str[index] H5 新增的
        console.log(str[0]); // a
```

```
/ 有一个对象 来判断是否有该属性 对象['属性名']
        var o = {
            age: 18
```

```
}
if (o['sex']) {
    console.log('里面有该属性');
} else {
    console.log('没有该属性');

}
//  判断一个字符串 'abcoefoxyozzopp' 中出现次数最多的字符，并统计其次数。
// o.a = 1
// o.b = 1
// o.c = 1
// o.o = 4
// 核心算法：利用 charAt( ) 遍历这个字符串
// 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1
// 遍历对象，得到最大值和该字符
var str = 'abcoefoxyozzopp';
var o = {};
for (var i = 0; i < str.length; i++) {
    var chars = str.charAt(i); // chars 是 字符串的每一个字符
    if (o[chars]) { // o[chars] 得到的是属性值
        o[chars]++;
    } else {
        o[chars] = 1;
    }
}
console.log(o);



// 2. 遍历对象
var max = 0;
var ch = '';
for (var k in o) {
    // k 得到是 属性名
    // o[k] 得到的是属性值
    if (o[k] > max) {
        max = o[k];
        ch = k;
    }
}
console.log(max);
console.log('最多的字符是' + ch);
```

字符串操作方法

字符串通过基本包装类型可以调用部分方法来操作字符串，以下是部分操作方法：

方法名	说明
concat(str1,str2,str3...)	concat() 方法用于连接两个或多个字符串。拼接字符串，等效于+，+更常用
substr(start,length)	从start位置开始（索引号），length 取的个数 重点记住这个
slice(start, end)	从start位置开始，截取到end位置，end取不到 (他们俩都是索引号)
substring(start, end)	从start位置开始，截取到end位置，end取不到 基本和slice 相同 但是不接受负值

replace()方法

replace() 方法用于在字符串中用一些字符替换另一些字符，其使用格式如下：

```
字符串.replace(被替换的字符串， 要替换为的字符串)；
```

split()方法

split()方法用于切分字符串，它可以将字符串切分为数组。在切分完毕之后，返回的是一个新数组。

其使用格式如下：

```
字符串.split("分割字符")
```

```
// 字符串操作方法
        // 1. concat('字符串1','字符串2'....)
        var str = 'andy';
        console.log(str.concat('red'));

        // 2. substr('截取的起始位置', '截取几个字符');
        var str1 = '中国人民共和国';
        console.log(str1.substr(2, 2)); // 第一个2 是索引号的2 从第几个开始  第二个2 是取几个字符
```

```
// 1. 替换字符 replace('被替换的字符', '替换为的字符')  它只会替换第一个字符
        var str = 'andyandy';
        console.log(str.replace('a', 'b'));
        // 有一个字符串 'abcoefoxyozzopp'  要求把里面所有的 o 替换为 *
        var str1 = 'abcoefoxyozzopp';
        while (str1.indexOf('o') !== -1) {
            str1 = str1.replace('o', '*');
        }
        console.log(str1);

        // 2. 字符转换为数组 split('分隔符')     前面我们学过 join 把数组转换为字符串
        var str2 = 'red, pink, blue';
        console.log(str2.split(','));
```

```
        var str3 = 'red&pink&blue';
        console.log(str3.split('&'));
```

2 - 简单数据类型和复杂数据类型

2.1 简单数据类型

简单类型（基本数据类型、值类型）：在存储时变量中存储的是值本身，包括string ，number，boolean，undefined，null

```
// 简单数据类型 null  返回的是一个空的对象  object
      var timer = null;
      console.log(typeof timer);
      // 如果有个变量我们以后打算存储为对象，暂时没想好放啥， 这个时候就给 null
      // 1. 简单数据类型 是存放在栈里面 里面直接开辟一个空间存放的是值
      // 2. 复杂数据类型 首先在栈里面存放地址 十六进制表示  然后这个地址指向堆里面的数据
```

2.2 复杂数据类型

复杂数据类型（引用类型）：在存储时变量中存储的仅仅是地址（引用），通过 new 关键字创建的对象（系统对象、自定义对象），如 Object、Array、Date等；

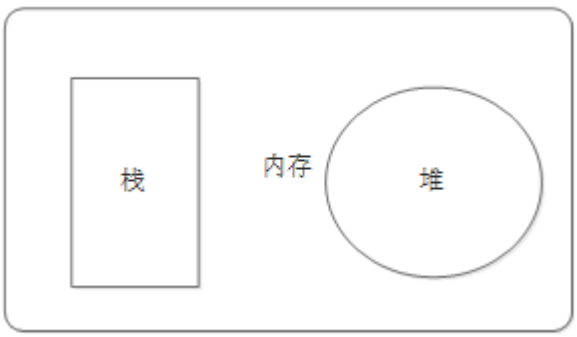
2.3 堆栈

- 堆栈空间分配区别：

1、栈（操作系统）：由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈；

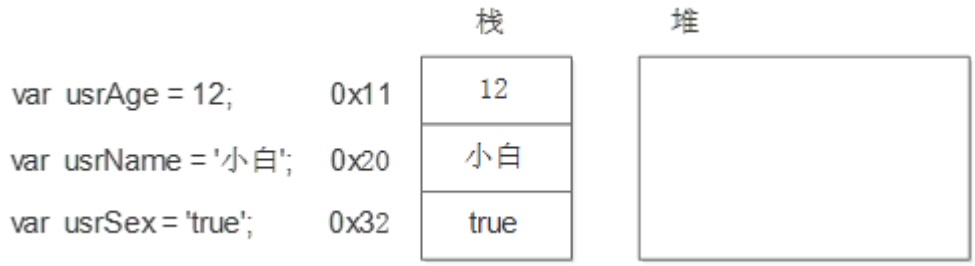
简单数据类型存放到栈里面

2、堆（操作系统）：存储复杂类型(对象)，一般由程序员分配释放，若程序员不释放，由垃圾回收机制回收。



- 简单数据类型的存储方式

值类型变量的数据直接存放在变量（栈空间）中



- 复杂数据类型的存储方式

  引用类型变量（栈空间）里存放的是地址，真正的对象实例存放在堆空间中

2.4 简单类型传参

函数的形参也可以看做是一个变量，当我们把一个值类型变量作为参数传给函数的形参时，其实是把变量在栈空间里的值复制了一份给形参，那么在方法内部对形参做任何修改，都不会影响到的外部变量。

```
// 简单数据类型传参
    function fn(a) {
        a++;
        console.log(a);
    }
    var x = 10;
    fn(x);
    console.log(x);
```

2.5 复杂数据类型传参

函数的形参也可以看做是一个变量，当我们把引用类型变量传给形参时，其实是把变量在栈空间里保存的堆地址复制给了形参，形参和实参其实保存的是同一个堆地址，所以操作的是同一个对象。

```
// 复杂数据类型传参
    function Person(name) {
        this.name = name;
    }

    function f1(x) { // x = p
        console.log(x.name); // 2. 这个输出什么 ?
        x.name = "张小龙";
        console.log(x.name); // 3. 这个输出什么 ?
    }
    var p = new Person("李小龙");
    console.log(p.name); // 1. 这个输出什么 ?
    f1(p);
    console.log(p.name); // 4. 这个输出什么 ?
```