运算符和表达式及数据类型转换

1、&&与 ||或 ！非

&& 整个表达式的值：如果&&前面为真就取后面的值作为整个表达式的值如果&&前面为假就取前面的值作为整个表达式的值,后面的表达式根本不执行;

|| 整个表达式的值：如果||前面为真就取前面的值作为整个表达式的值，后面的表达式根本不执行;如果||前面为假就取后面的值作为整个表达式的值;

！ 非真即假 非假即真 和它构成的表达式 最终都会是一个布尔值；

2、三目运算符 三元 问号冒号表达式

问号冒号表达式：先看第一个表达式问号前面的值是否为真(不是布尔要转化为布尔)？如果为真，则取冒号前面的值作为整个表达式的值，后面的不执行，如果为假,则取冒号后面的值作为整个表达式的值,前面的不执行

3、数据类型强制转换（显式）

1）其它类型值转数字：Number() 强制将一个其它类型数据转化为数字类型转不了就是NaN

转化字符串

如果字符串整体来看是一个数字，那么就转化为这个数字

如果字符串整体来看不是一个数字，那么就转化为NaN

如果字符串是一个特殊的空字符串或者空白字符串，那么转化为0

转化**boolean**

true会转化为1 false 会转化为0

转化**undefined**

undefined 会转化为NaN

转化**null**

null 会转化为0

2）其它类型值转字符串：String() 强制将一个其它类型数据转化为字符串类型

3）其它类型值转布尔值：Boolean() 强制将一个其它类型数据转化为boolean类型

转化数字的时候，除了0和NaN是false,其余都是true;

转化字符串的时候，除了空字符串是false,其余都是true

转化undefined和null都是false;

4、基本数据运算（隐式转换）

（1）判等：

判断是否同类型：是　　看是否相同

不是　转数字

当遇到null的时候;会有特殊情况发生,

特殊情况：空串和null不相等

特殊情况：false和null不相等

特殊情况: 0和null不相等

特殊情况：undefined 和 null 相等；

（2）在其余的运算和比较情况下：

同时存在 “+" 与 "字符串" :字符串拼接

字符串比较转Unicode码：null小于0

最后：　全部转数字

NaN:

所有的东西和NaN进行算术运算都是NaN

所有的东西和NaN进行比较大小都是false

所有的东西和NaN都不相等（包括自己）

NaN == NaN //false

=== !==

全等和不全等： 不会出现类型转换，他们在判等的时候，先判断类型是不是一样，如果类型一样再判断值是不是相同，如果都相同才全等；如果有一个不一样，那么就不全等;

1 == true; //true

1 === true； //false

流程控制语句

prompt();//由键盘输入（输入为字符串）

如果以后我们碰见if...else双分支语句，而且每个分支当中只有一条语句，就最好改成三元表达式（问号冒号表达式）；

1、switch....case分支语句（switch case break default）

switch 语句执行过程： 先求出小括号当中的值 接着会拿着这个值从上到下和所有的标号后面的值进行对比，如果对比成功，就执行对比成功这个标号下面的代码块；

注意：switch当中对比的时候，判等的时候使用的是全等

可在case中写表达式

流程语句练习 + 数组

1、break和continue关键字作用 *****

break:作用1：在switch语句当中是跳出switch 作用2 在循环当中，跳出离它最近的一层循环；

continue:作用： 结束本次循环，返回从下一次继续开始；

空格：

2、数组格式

```
//方法1
var arr = [1, 2, 3, 4, 5];
console.log(arr);
var arr1 = [];
console.log(arr1);
//方法2
var arr2 = new Array(1, 2, 3, 4, 5);
var arr3 = new Array();
var arr4 = new Array(4);
```

3、方法

```
//    在数组的末尾加一个数
arr[5] = 100;
arr[arr.length] = 200;
console.log(arr);
```

函数**********

1、概念，定义（表达式，字面量），作用

1）什么是函数：

具有某种特定功能的代码块~ //想象成工具或者工厂

函数其实本质也是一种数据，也是属于对象数据类型；

2）为什么要有函数

1、解决代码的冗余问题，形成代码复用；

2、可以把整个代码项目，通过函数模块化；

3、封装代码，让函数内部的代码对外部不可见

3）函数的定义（三要素）

功能（函数名，见名思意，看到了函数名就想到了这个函数的功能，函数名字最好使用动词）

参数：（）里面是函数的参数，定义函数的时候要考虑这个函数是否有参数；

返回值：函数最终都会有一个返回值；定义函数的时候也要考虑这个函数是否需要返回值。

4）函数的调用

默认return的是一个undefined.

5）****函数的调用过程（内存角度）**

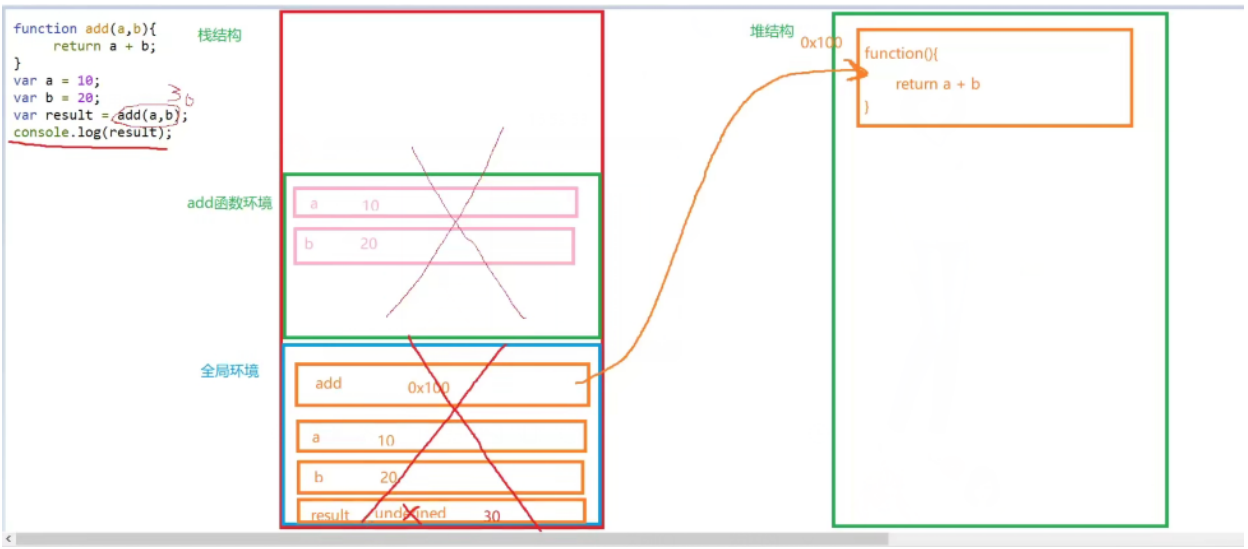
（1）内存结构

图1

（2）全局环境和局部环境

函数本身属于全局环境

（3）具体步骤



程序执行：在栈结构中开辟空间作为全局环境

在全局环境中开辟空间，起名叫add，去堆结构中开辟一块内存，将具体函数写入堆内存，将该内存的地址，赋值给add。

在全局环境中开辟空间，起名叫a，值为10.

在全局环境中开辟空间，起名叫b，值为20.

在全局环境中开辟空间，起名叫result，值为undefined。函数调用。在栈结构中开辟空间作为add函数环境，在函数环境中开辟空间，起名叫a，值为10.在函数环境中开辟空间，起名叫b，值为20.将a+b的值传给resul，30覆盖undefined。释放环境变量的内存。

程序执行完毕释放全部内存。

(4) 数组步骤

访问一个对象：

```
var arr = [1,2,3];
console.log(arr)  打印出来的是数组数据
var arr2 = arr;
    //arr2当中存储的是数组的地址，代表着arr和arr2同时指向了同一个数组；
arr[0] = 100;
 console.log(arr2);//[100,2,3];
```

(5) 程序开始执行到结束都做了什么：

****

1、程序一开始执行，碰见了全局环境，首先会创建全局环境并且进行压栈，全局代码执行的时候依赖的就是全局环境当中的东西；比如 全局变量（全局变量如果存的是基本数据类型，那么这个值是直接存在栈当中的，如果这个变量存的是对象类型（函数、数组），那么数据是要在堆结构当中开辟自己的空间专门存储的。然后把堆里面这块空间的地址存给栈当中的对应变量）；

2、当程序执行碰到了函数调用；函数是比较特殊，因为它也可以执行；函数执行的时候也要有自己的环境去依赖。因此函数执行也是创建自己的函数环境进行压栈

（函数环境一定是压在全局环境之上的），局部变量，是在函数环境当中存在的，

只有函数执行，局部变量才会出现。函数执行完成以后，函数环境要弹出栈（销毁

归还内存）,局部变量也就不复存在了。

3、当函数调用完成以后，会继续执行全局代码，一直到所有的代码都执行完成，

代表程序执行结束，程序结束的时候，我们的全局环境最后出栈。

2、全局变量和局部变量

1）全局和局部变量都带var

函数内部定义的局部变量只能在函数内部使用，函数外部看不到

函数外部全局变量，在函数内部是可以使用的（前提函数内部没有定义过这个相同的变量）；

2）碰到函数内部变量不带var（特殊情况）

1、先在函数内部去找，有没有写过var b;

2、如果找到var b，就是局部变量去处理；如果没有找到，那么去形参去找，看看有没有b这个变量;

3、如果形参找到了b，同样当作局部变量去处理（相当于内部写了var b）,如果没有找到，去函数外部全局作用域去找

4、如果全局找到了b，那么内部的b相当于是在操作全局变量；如果没有找到，那么相当于是在全局定义了一个全局变量；

3、案例:

面试题

```
var num = 10;
function fun() {
    var num = 20;
    fun2();
}
function fun2() {
    console.log(num);
}
fun();//答案是10
```

4、作用域链

作用域链和作用域不是一回事；

作用域描述是变量起作用的区域和范围

作用域链描述的程序在找变量的过程；

程序在查找变量的时候，先从自己的作用域去查找，如果找到就直接使用这个变量的值，如果没有找到，会继续往上一级作用域去找，同样也是找到就使用没有找到继续往上找；直到找到全局作用域，如果找到就使用，找不到就报错（引用错误，这个变量没有定义）；

作用域是抽象的，不存在；

作用域链是真正存在的东西，我们后面是可以看到的；

5、预解析（预解析，变量提升）

a) **预解析（全局和局部）**

1、程序在代码执行之前会先进行预解析：预解析会解析带var的变量和字面量定义的函数

2、解析函数优先级比解析变量要高：可以认为解析的时候分为两步，先去解析所有的函数，再去解析所有的带var变量；

3、解析过程当中，如果函数重名，会覆盖（后面的函数会把前面的覆盖掉）

如果变量重名，会忽略；

如果变量不带var，则不会解析

4、解析函数的时候，函数定义方式不同，解析过程也不大一样：

如果是字面量定义，整个函数都要被提升，而且是第一步解析function func(){};

如果是表达式定义，只会提升变量，当作变量提升第二步解析 var func = function(){};

b）字面量定义的函数和函数表达式定义的函数不同点：

1/预解析的时候不同：字面量定义的会全部提升 ，表达式定义的只会提升变量（当作变量提升对待）

2/打印这个函数本身的时候不同 字面定义的函数会带名字打印，表达式定义的不带名字；

3/除了上述2个不同点以外,都可以认为函数就是定义了一个变量 里面存了一个函数数据(本质上是函数数据的地址

```
3、function fn(){};//1
    var fn = function(){};//2   调用函数时打印2
```

c) **做题思路**

1、先把代码调整好（全局预解析,把该提升的提升好）；

2、画图 （全局环境）

3、执行全局代码

4、碰到函数调用先把函数内部的代码调整好（函数内部预解析，把该提升的提升好）；

5、画图（函数环境）

7、函数死，继续执行全局代码

8、全局死

6、 IIFE，回调函数 函数递归 arguments

1、IIFE:

Immediately Invoked Function Expression意为立即调用的函数表达式，也就是说，声明函数的同时立即调用这个函数。（也叫匿名函数自调用）

语法：

```
(function(){
    代码块;
})();
```

特点：

函数定义的时候同时执行

只执行一次

不会发生与解析（函数内部执行的时候会发生预解析）

作用：

防止外部命名空间污染

隐藏内部代码暴露接口

对项目的初始化

2、Arguments 函数实参伪数组

1、**不写形参**

1）伪数组：本质是一个对象，但可以跟数组一样使用，有下标有长度。

2)argument：存在于函数内部

3）函数可以不写形参

```
function add() {
    console.log(arguments);
    return arguments[0] + arguments[1];
}
console.log(add(10, 20));
```

2、通过argument使函数具有多个功能

```
function addOrSub(a, b, c) {
    if (arguments.length == 2) {
        return a - b;
    } else if (arguments.length == 3) {
        return a + b + c;
    }
}
console.log(addOrSub(20, 10));
```

3、回调函数：函数是我定义的　我没有调用　　最终执行了

事件

定时器

生命周期回调函数

4、函数递归调用

对象

一、 Object的实例对象

1.对象的概念

1）面向对象和面向过程

在js当中，可以说一切皆对象，js是一门面向对象的语言；

C就是面过程的语言

java　python js都是面向对象的；

大象装冰箱

2）什么是对象及作用

无序的名值对的集合（键值对的集合）就叫做对象；

- 如果存储一个简单的数据（一个数字，一个字符串）　直接var a = 10;
- 如果存储一堆的数据 此时我们想到数组　数组就是专门用来存储多个数据用的
- 如果我们想要执行一段代码，或者让这段代码有功能，此时我们需要函数
- 如果我想描述一个复杂的事物，比如说一个人，一台电脑（需要用到多个属性或者方法才能描述清楚），此时就要用到对象；

3）对象的创建方法

a)字面量创建:

对象是由属性和属性值组成的，其实就是我们说的键值；

属性也可以分为属性和方法

```
var obj = {
  name: "赵丽颖",
  age: 33,
  gender: "female",
  eat: function() {
    console.log("啥都吃");
  }
};

console.log(obj);
```

对象属性的本质是字符串，也必须是字符串。

b) new Object（构造函数定义）

```
var obj = new Object({
  name: "旺财",
  age: 2,
  color: "yellow",
  catagory: "金毛",
  run: function() {
    console.log("暖男");
  }
});
```

注意：无论你是通过字面量创建的对象还是通过构造函数创建的对象，本质上都是通过构造函数创建的对象

字面量创建的对象是构造函数创建对象的简写方式；

通过Object new出来的对象被称作是 Object的实例化对象，因为这个对象是通过Object实例化出来的；简称实例

只要是通过Object实例化出来的对象，都是比较高层次的对象，站的高度比较高，这个对象可以是任何的对象；

看里面的属性，如果里面的属性表示的是人那就是人呢，如果里面的属性表示的是狗，那就是狗；是一个通用的对象

c) 工厂函数模式：本质就是通过构造函数定义对象，只不过封装成了函数。

```
function createObject() {
  var obj = new Object();
  return obj;
}
```

无论你用的是上面的哪一种方式去创建的对象，这些对象都被称作是Object的实例对象

2.对象的操作及遍历（增删改查）

增删改查：点语法和[]语法操作

遍历　for in循环进行遍历对象

(1)点语法操作

```
//增
obj.gender = "female"; //无则增加
//改
obj.name = "颖宝"; //有则更改
```

```
//删
delete obj.age;
console.log(obj);
```

(2)[]语法操作

某些情况下点语法是没法去操作的，此时我们需要使用[]语法（【】内是一个值）

a、当我们的属性名是一个不符合标识符规范的名字要用[]。

任何的字符串都可以作为属性名，只不过不符合标识符规则的，操作的时候必须要用[]

[]语法其实任何字符串属性名都可以操作，符合规则的也可以使用[]。只不过符合规则的

我们常使用点语法

```
//改
obj.content-type = '嘿嘿';//假设操作的是不符合标识符规则的属性，那么没办法使用.
obj['content-type'] = '嘿嘿';
//增
obj['123'] = 345;//123这个字符串名字不符合标识符规则
//符合标识符规则的也可以使用[]
obj['name'] = '杨幂';
console.log(obj);
//[]是通用的
```

b、当我们要使用变量里面的值，作为对象属性名的时候，要用到[]

```
var a = 'haha';
obj[a] = 'heihei';//[内部如果是变量的话，不需要写''],如果写了引号就等价于  obj.a
                  //给对象内部添加了一个属性名字叫'haha'  值叫heihei
                  //obj['haha'] = 'heihei';
                  //obj.haha = 'heihei';

console.log(obj)
```

```
//删
delete obj['content-type'];
console.log(obj);
var b = 'name';
delete obj[b];//b是变量，不能加引号
console.log(obj);
```

(3)对象的查

```
//查
console.log(obj.gender);
//obj.gender 如果是右侧有等号，那么是在给这个属性赋值（可能是增也可能是改），没有等号就是读取这个属性的值
console.log(obj['123']);
//如果你读取一个不存在的属性的值，不会报错（它不是变量，变量的话会报错，属性会返回undefined）；
console.log(obj.aaa);//undefined
var c = 'eat';
console.log(obj[c]);//obj['eat'] ===== obj.eat;
console.log(obj.eat())//调用方法
```

(4)对象的遍历

对象的遍历要使用for in循环,遍历的是对象的属性（键）

for是专门用来遍历数组的

```
for (var key in obj1) {
  console.log(key);//遍历打印对象当中所有的属性
  console.log(obj1[key]);//遍历打印对象当中所有的属性值
  // console.log(key, obj1[key]);
}
```

3、数组、函数和对象的关系

对象就是对象，一切都是对象

（1）数组（数组是数组数组也是对象）

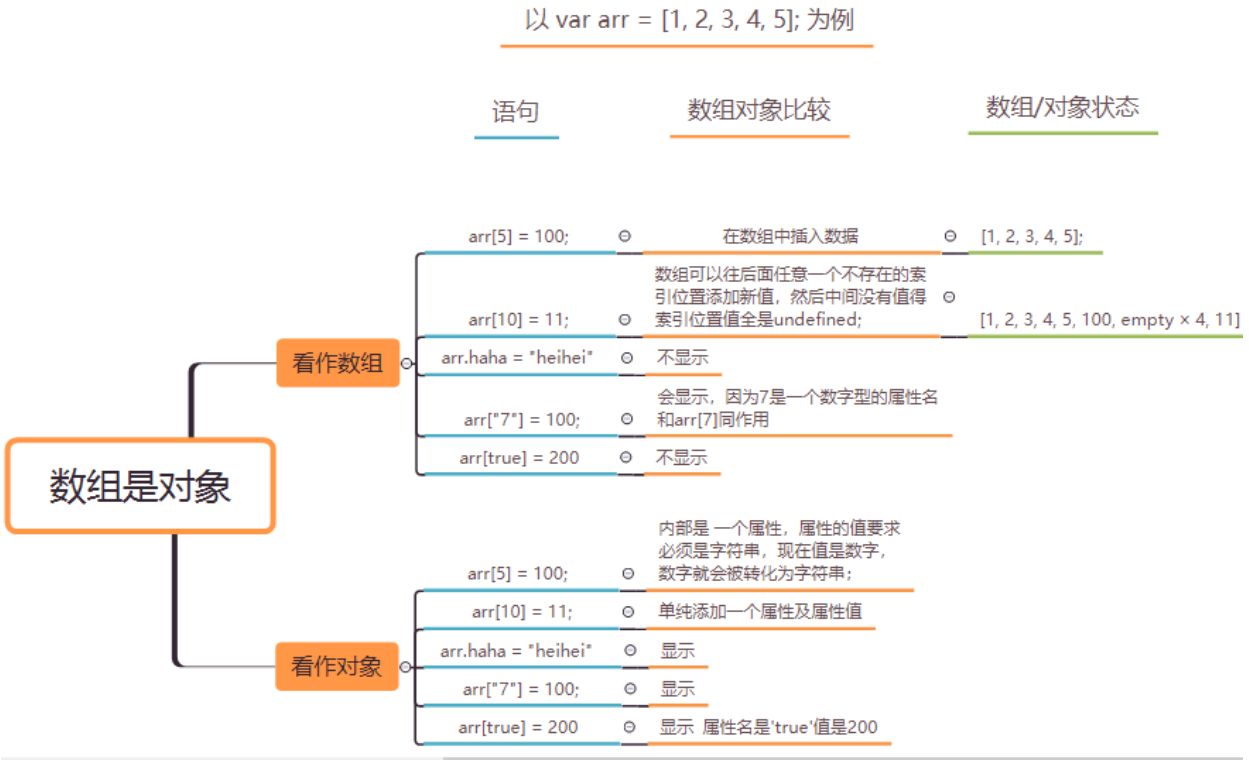
a、看作数组

```
arr[5] = 100;
// 当对象去看   【】内部是 一个属性，属性的值要求必须是字符串，现在值是数字，数字就会被转化为字符串;
arr[10] = 11;
console.log(arr);
//数组可以往后面任意一个不存在的索引位置添加新值，然后中间没有值得索引位置值全是undefined;

//当数组去用  只能拿到数组数字属性（包含索引）的值
for (var i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

b、看作对象

```
arr.haha = "heihei"; //只会在对象当中显示
arr["7"] = 100; //在数组当中和对象当中都会显示，因为7是一个数字型的属性名   和arr[7]
arr[true] = 200; //在对象当中会显示  属性名是'true'值是200

//当对象去用,所有的属性值都能拿到，length是一个特殊的属性，数组当中length属性不允许遍历访问;
for (var key in arr) {
  console.log(key, arr[key]);
}
```

c、比较



（2）函数（函数是函数函数也是对象）

console.dir(fn); 详细展示函数内部

A() A一定是函数，不是函数就报错（类型错误）；

A.B A一定是对象（不是对象也是要报错的，类型错误），B一定是A这个对象当中的属性

a、把函数当函数用,主要是为了执行函数当中的代码

```
function fn() {
  console.log("i love you~");
}
fn();
```

b、当对象用，是把函数当做一个对象用来存取值

```
fn.gender = "男";
for (var key in fn) {
  console.log(key, fn[key]);
}
```

二、 构造函数创建特定实例对象

1.构造函数的基本概念，使用，作用

构造函数：本质上也是一个函数，只不过通常我们把构造函数的名字写成大驼峰；

在js当中，没有类的概念（5版本）,构造函数可以理解为类；

任何的函数都可以是普通函数，也可以是构造函数，就看你在怎么用；

```
//相当于定义了一个人类，通过这个类new出来的都是人的对象，不可能是其他的
function Person(name, age, gender) {
  //如果我们要把函数当构造函数用，首字母一般都是大写（大驼峰写法）
  this.name = name;
  this.age = age;
  this.gender = gender;
  this.eat = function() {
    console.log("吃饭");
  };
}
//var result = Person('zhaoliying',33,'female');//把函数当做普通函数用
var p1 = new Person("zhaoliying", 33, "female"); //把函数当构造函数用
console.log(p1);
```

2.对**this**的讲解

this介绍

1、this是执行环境当中存在东西（程序执行后才有），在全局和函数内部都会存在this这个

东西

2、this的本质，它是系统内置变量，这个变量里面存储的是一个对象，this最终代表的是一

个对象

3、this通常情况都是在函数内部使用，函数外部this不常用；

4、this最终代表的是函数执行时候的执行者（执行环境对应的执行者）

5、window是一个对象，这个对象代表着浏览器窗口对象，当程序执行的时候，所有的一切

都是包含在window对象里面的；

6、我们把window称作是顶级对象；

this常见的情景

1、如果this在全局 永远是window

2、如果this出现在函数（调用的时候直接加（））当中 内部的this也是指向window，所有

的函数this都是window

3、如果this出现在方法当中 内部的this指向的是这个方法的对象；（方法与函数不同，方法

是存在于对象中，某个属性的值，是一个函数，那么成这个属性是一个方法）

4、如果this出现在构造函数（函数调用的时候加new和（））当中 内部的this指向的是准备

实例化出来的对象

5、如果函数调用使用了apply和call方法，那么这个函数内部this指向的对象是由我们指定的

6、如果this出现在事件回调函数当中 内部的this指向的是这个事件的事件源

注意:在js当中,函数也可以称作是window对象的方法,因此很多人会说函数就是方法,方法就是函数

window对象简介

浏览器窗口对象，代码执行的时候所有的一切都是包含在窗口对象下的.

函数的使用方式

首先我们要清楚无论是什么函数，本质上都是函数，函数在使用的时候有多重用法。

• 构造函数去用：如果当作构造函数去使用，需要在函数调用前面加上new才代表构造

函数使用，构造函数在使用的时候，this指向是准备实例化出来的那么对象，并且构造函数

如果没有return，它是可以返回实例化出来的那个对象的；如果return 是一个基本数据类型,

那么还是返回实例化对象,和return的值没关系，如return后面是一个对象数据,那么返回的就

是这个对象数据,不再是实例化对象了

• 普通函数去用，也就是说没有加new，那么此时函数相当于没有做实例化过程，仅仅

是给window对象添加了一些属性及属性值。如果没有return，返回的是undefined;

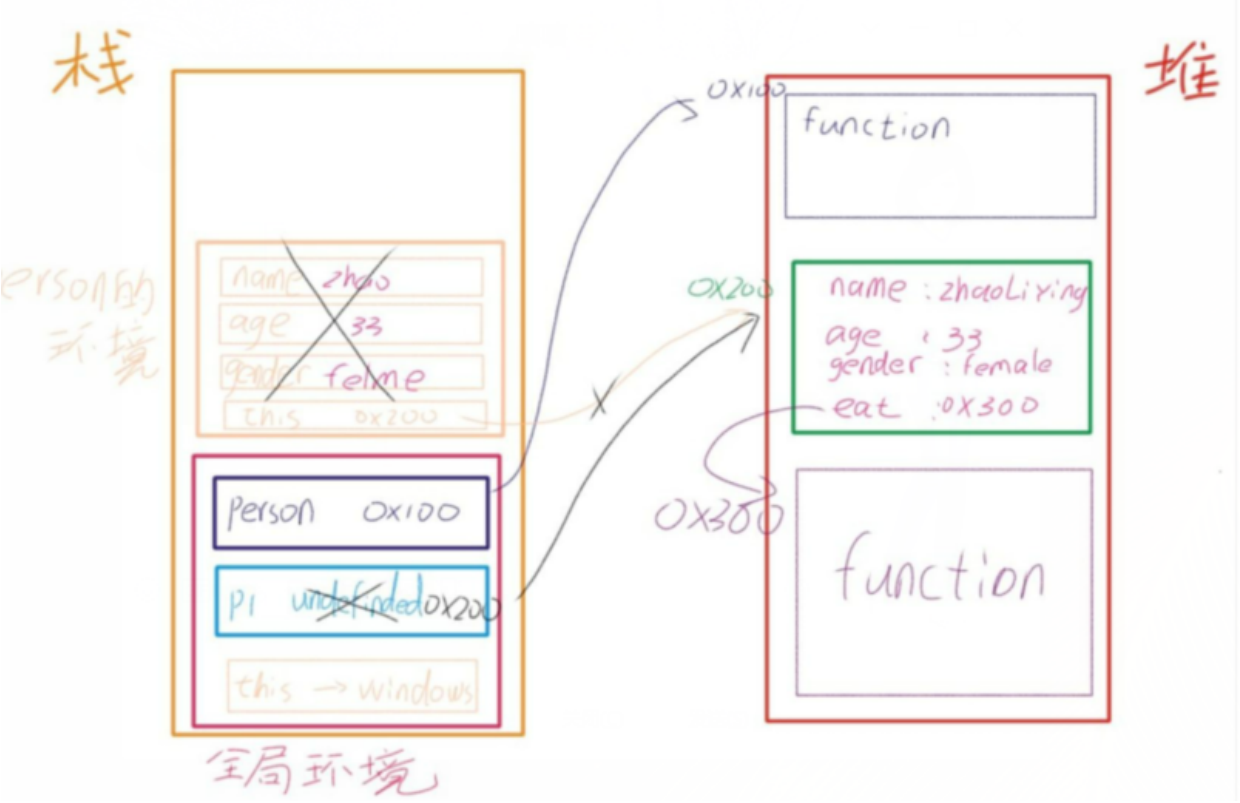
3.new关键字实例化对象的过程

1、开辟内存空间(堆)

2、this指向该内存（让函数内部的this）

3、执行函数代码

4、生成对象实例返回



4.原型对象和原型链 ******** **

原型对象

什么是原型对象: 一个函数对象在定义的时候，伴随它出现的另外一个对象就是原型对象，

原型对象是默认是object的实例对象

显示原型对象和隐式原型对象概念

函数对象身上的prototype属性值（显式）

实例化对象身上的__proto__属性值（隐式）

这两个属性的值是同一个对象，就是我们说的原型对象

原型对象的作用，

资源共享 节约内存

原型链

描述的是对象在查找属性或者方法的过程

实例化对象在找属性的时候，先从自身去找看有没有这个属性，如果有，直接使用这个属

性的值，如果没有，会继续顺着这个对象的隐式原型对象（__proto__）找到这个对象的原型对

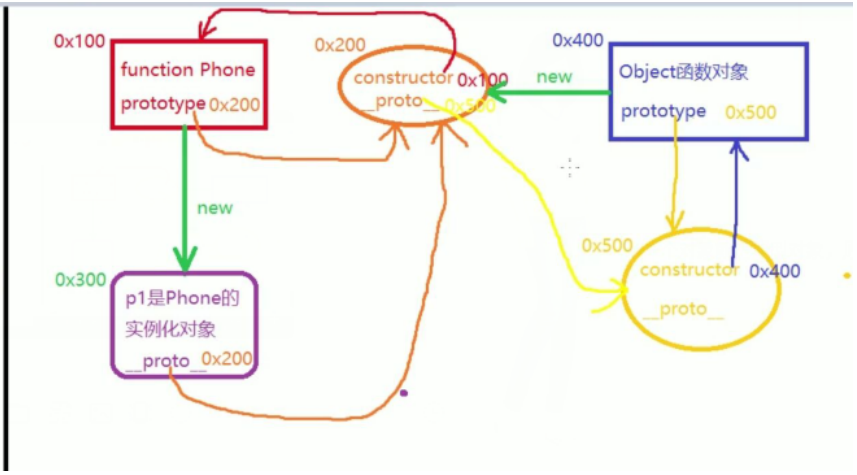
象（和它的构造函数的显式原型对象是同一个），看看原型对象是否存在这个属性，如果有就使

用原型对象当中的这个属性值，如果还没有，再去找原型对象的隐式原型对象（默认就是Object

显式原型对象），找到以后去看看有没有这个属性，如果有就使用这个属性值；如果没有就返回

undefined(代表已经找到顶了)；

```
functionPhone(name,brand,color,size){
    this.name = name;
    this.brand = brand;
    this.color = color;
    this.size = size;
}

Phone.prototype.playGame = function() {
    console.log('吃鸡~');
}

Object.prototype.playMusic = function(){
    console.log('听网易云音乐~');
}

var p1 = new
Phone('iphone6sp','iphone','gold',5.5);
p1.playGame();
p1.playMusic();
```



原型链和原型是两码事：类似于作用域和作用域链（查找的变量的过程，最终找不到要报错）

原型链：描述的是对象在查找属性的过程

对象先从自身内部查找，如果有就使用，没有就通过自身的隐式原型对象去找，如果有就用，没有就找自身的隐式原型对象的隐式原型对象去找

找到就用,没有找到继续找,找到Object这个构造函数的显式原型对象为止,找到就用,找不到返回undefined(不会报错)；

5. apply和call

任何函数对象都有apply和call方法（存在于prototype中）

apply和call可以使用第一个参数传对象，让函数或者方法的执行者（this）指向这个对象；

函数或者方法.apply(对象，【函数的参数】)；

函数或者方法.call(对象，函数的参数1，函数的参数2)；

```
//      apply
d1.run.apply(c1, [600, 'haha']);
//      call
d1.run.call(c1, 600, 'haha');
```

call和apply干了两件事：

1、调用的时候先把this指向改为指定的某个对象

2、然后再去执行使用的方法

call和apply可以让一个对象执行另外一个对象的方法；

6. instanceof

typeof 应用的场景

typeof 值　返回的是数据类型的小写字符串形式 （仅可区分5种）

数字　'number'

字符串　‘string’

布尔　'boolean'

undefined　'undefined'

null　'object'

数组　'object'

函数　'function'

对象　'object'

instanceof 应用的场景

instanceof 用来判断一个对象是哪个构造函数的实例用的　A(对象数据) instanceof B（构造函数）

专门用来解决判定数组和对象的时候使用

全等（===）于可以用来判定null和undefined的时候使用；因为他们两个都是数据类型，但是这数据类型当中只有一个值；

a === null

a === undefined

通过typeof instanceof以及 === 可以让我们判定js当中所有的数据类型

```
//判定b是不是一个数组
//判断b里面的值是不是Array的一个实例
console.log(b instanceof Array);    //new Array()
//判定b是不是一个对象，任何的对象数据都是对象
console.log(b instanceof Object);
```

三、 值类型，引用数据类型，堆， 栈

1.值类型和引用数据类型的概念

值类型都有啥: 其实就是我们所说的基本数据类型

引用类型都有啥：引用数据类型就是我们所说的对象数据类型、复杂（复合）

2.堆和栈的概念（见图1:函数章）

3.堆和栈的图解

4.值类型和引用数据类型与堆栈的关系

堆空间的释放是靠垃圾回收机制进行的

当程序函数或者整个程序执行完成后，栈里面所有的东西都被释放销毁，堆当中的数据可能还在，只是没有任何的变量指向（引用），那么堆当中的数据就会变成垃圾对象。回收

机制会在适当的时候将垃圾对象清理回收；

如果我们在程序当中需要去删除对象，那么就将这个对象的变量赋值为null，代表这个对象引用被改变，这个对象也就成了垃圾对象，其实删除对象就是让堆当中的对象数据成为垃圾对象；

5、面试题

第三题

```
var num1 = 55;
var num2 = 66;
function f1(num, num1) {
    num = 100;
    num1 = 100;
    num2 = 100;//未定义改全局
    console.log(num);
    console.log(num1);
    console.log(num2);
}
f1(num1, num2);
console.log(num1);
console.log(num2);
console.log(num)；//找不到，报错
```

第四题

```
function f1(arr){
    for(var i = 0; i < arr.length; i++){
        arr[i] += 2
    }
    console.log(arr);
}
var arr;
 arr = [1,2];
f1(arr);//传入的是地址
console.log(arr);
```

第五题

两个对象是同一个对象，不同的操作有什么不同

```
 var a = [1,2];
var b = a;
```

```
a[0] = 20;
//如果a = [20,2]；//代表把新的地址给a
console.log(b);
```

（1）20 （2）1

```
var a = [];//new Array()
var b = [];//new Array()
console.log(a == b);
//对象数据判等，看地址
```

第六题

```
function Person(name, age, salary) {
    this.name = name;
    this.age = age;
    this.salary = salary;
}
function f1(pp) {
    pp.name = "ls";
    pp = new Person("aa", 18, 10);
}
var p = new Person("zs", 18, 1000);
console.log(p.name);
f1(p);
console.log(p.name);
```

四、 内置对象JSON

1.什么是json,json作用

json是一种数据格式；现在我们大多数都是通过json数据格式进行前后端数据交互的，json本质上是一个字符串，简称json串

前端往后台传数据的时候，要传json格式的数据json串

在前端json串的格式原形就是对象或者对象的数组；所以我们要先把数据存储为对象或者对象的数组，然后转化为json串进行传递

```
var result = JSON.stringify(obj); //把对象和对象的数组转化为json串用的
```

ajax，ajax主要就是向后端发请求，此时就可以把result这个json串通过ajax传递到后端去;

如果后端给我们传递过来的数据也是json串

此时我们在前端拿到json串之后，要做逆运算，把json串再转化为我们前端认识的对象或者对象的数组，然后从中取数据

```
result = JSON.parse(result); //解析的意思，它会把json串再转化为我们的对象或者对象的数组
```

五、 Math工具对象

1.Math对象方式的使用

round（四舍五入取整）, floor（下取整）,ceil（上取整）， random, max min PI pow（a,b）(a的b次幂) abs （绝对值） sin

//负数的下代表小的，四舍五入有误差

当有效位数确定以后，其后面多余数字应该舍去，只保留有效数字最末一位，这种修约（舍入）的规则是“四舍六入五成双”，里面的“四”指<=4舍去，“六”指>=6进位，“五”要根据后面的数字来定，5后面有数时，舍五入一，5后无有效数字时，需要分两种情况：（1）5前为奇数，舍五入一（2）5前为偶数，舍五不进（0算偶数）

2.案例：

随机数点名

随机生成验证码

六、 Date日期对象

1.Date对象的方法

```
var date = new Date();
console.log(date);
console.log(date.getFullYear());
console.log(date.getMonth());
console.log(date.getDate());
console.log(date.getHours());
console.log(date.getMinutes());
console.log(date.getSeconds());
console.log(date.getYear());//已经废弃
console.log(date.toLocaleTimeString());
console.log(date.toLocaleDateString());
console.log(date.getTime());//1970 年 1 月 1 日之间的毫秒数
```

```
//一开始拿到一个毫秒数
var timeStart = Date.now();
//Date是一个函数，这个函数是和日期时间相关的一个函数
//Date直接当普通函数用
var a = Date();
console.log(a);
//Date当构造函数用
var date = new Date();
console.log(date);
//       这两种函数用法，返回的都是中国标准时间，它是一个对象
console.log(Date.now()); //这个方法返回的是从1970年1月1日0时0分0秒到现在的毫秒数
//函数是函数  函数也是对象
//函数当函数用，有两种角色，一种普通函数，一种是构造函数
//函数当对象用，就是函数.属性(或者方法名)
//原型对象是为了实例化对象服务的，而不是给构造函数对象本身用
//构造函数对象本身的方法和实例化对象的方法没关系，不能互相去直接使用
console.log(date.getFullYear()); //只拿年，并且是全写的年
console.log(date.getMonth()); //拿到月  月份是用0 - 11表示的，因此拿到都得+1
console.log(date.getDate()); //拿到日
console.log(date.getDay()); //拿的是星期
console.log(date.getHours()); //小时
console.log(date.getMinutes()); //分钟
console.log(date.getSeconds()); //秒
console.log(date.toLocaleTimeString());
console.log(date.toLocaleDateString());
console.log(date.getTime()); //1970 年 1 月 1 日之间的毫秒数
```

（Data 的.now()函数不能给data使用，因为.now 是存在于Data中，若data中找不到只会去Data中的原型量中寻找，并不会去Data中寻找或者Data中的隐式原型链_photo_中寻找）

2.案例：

1. 格式化日期　2019年8月24日 下午4：43：

2. 封装函数实现格式化日期

```
//我现在想要拿到这样的一个日期时间字符串  '现在是2020年2月25日 上午10:27:3'
function getDateTimeString() {
  var date = new Date();

  var year = date.getFullYear();
  var month = date.getMonth() + 1;
  var day = date.getDate();
  var time = date.toLocaleTimeString();

  return '现在是' + year + '年' + month + '月' + day + '日 ' + time;
}

console.log(getDateTimeString());
```

七、 包装对象

基本数据类型也可以使用.调用方法

```
var a = 10; //基本数据类型10
console.log(a);
console.log(a instanceof Object);
console.log(a.toString()); //在这行执行的时候，其实是把a变成了包装对象，这行执行完之后，a再立马变回去成为数字基本值
//1、a = new Number(a);//先变成包装对象
//2、a.toString其实调用的是包装对象的方法
//3、调用完成之后，立马执行a = 10;

//A.B  A一定是对象，B一定是这个对象的属性
//为了让基本数据类型也可以和对象一样使用，因此添加了包装对象

//下面这个b其实就是数字对象（包装对象），包装对象有三种（数字  字符串  布尔值）
var b = new Number(10); //Number不带new就是普通函数调用，是为了转化数字用的，带了new就是构造函数是为了实例化对象的;
console.log(b);
console.log(b instanceof Object);
console.log(b.toString()); //把对象当中的基本值（PrimitiveValue）转化为字符串
console.log(b.valueOf()); //获取对象当中基本值（PrimitiveValue）;

//基本数据类型的valueOf和toString方法,都是在包装对象的原型对象中
```

1、对象的比较运算

（1）对象与对象

1、判等：如果两遍都是对象，那么判断的是地址是否一样

2、比较运算：比较和运算对象没法进行，因此，首先对象会转化成基本数据然后再进行；

对象数据转化基本数据的时候遵循规则：

a、数组可以调用valueOf(Object的原型中)和toString（数组自身原型当中）

b、函数也可以调用valueOf(Object的原型中)和toString（函数对象自身原型当中）

c、对象也可以调用valueOf(Object的原型中)和toString（Object原型中）

a、先尝试调用valueOf方法，取基本值，但是很可惜，除了包装对象以外，其它的对象是没有基本值；返回自身；

b、如果没有基本值，那么会接着调用toSting方法（自身原型中），把对象转化为字符串，字符串就是一个基本数据；

c、接着再按照基本数据的类型转化进行比较和运算；

转化字符串规则

数组最终调用toString返回的就是去掉[]，中间的变为字符串；

函数最终调用toString返回的就是函数本身，是一个字符串；

对象最终调用toString返回的是固定的字符串[object Object]

（2）对象与基本数据类型

对象和基本数据运算比较判等：对象都要去转基本值

```
var a = {};
var obj1 = [1, 2, 3];
var obj2 = function () {};
var obj3 = {
  n: 3,
  m: 4
};
a[obj1] = 5; //a['1,2,3'] = 5
a[obj2] = 6; //a['function(){}'] = 6;
a.username = '赵丽颖';
a[obj3] = 4; //a['[object Object]'] = 4;

console.log(a);
```

```
03-笔记.html:25
▶ {1,2,3: 5, function () {}: 6, username: "赵丽颖", [object Object]: 4}
> |
```

ES5/ES6 String/Array方法

一、 string方法 ES5

JavaScript核心参考索引 JavaScript核心参考说明

String（字符串对象）

String	对字符串的支持
String.charAt( )	返回字符串中的第n个字符
String.charCodeAt( )	返回字符串中的第n个字符的代码
String.concat( )	连接字符串
String.fromCharCode( )	从字符编码创建一个字符串
String.indexOf( )	检索字符串
String.lastIndexOf( )	从后向前检索一个字符串
String.length	字符串的长度
String.localeCompare( )	用本地特定的顺序来比较两个字符串
String.match( )	找到一个或多个正则表达式的匹配
String.replace( )	替换一个与正则表达式匹配的子串
String.search( )	检索与正则表达式相匹配的子串
String.slice( )	抽取一个子串
String.split( )	将字符串分割成字符串数组
String.substr( )	抽取一个子串
String.substring( )	返回字符串的一个子串
String.toLocaleLowerCase( )	把字符串转换小写
String.toLocaleUpperCase( )	将字符串转换成大写
String.toLowerCase( )	将字符串转换成小写
String.toString( )	返回字符串
String.toUpperCase( )	将字符串转换成大写
String.valueOf( )	返回字符串

```
console.log(str.charAt(8)); //index     ****************
//功能 ：找出指定索引处的字符
//参数：指定的索引
//返回值：返回这个指定索引的字符
console.log(str.charCodeAt(9)); //0  48 1 49
//功能 ：找出指定索引处的字符的Unicode码
//参数：指定的索引
//返回值：返回这个指定索引的字符的Unicode码
console.log(str.concat()); //拼接
//功能 ：将原串和指定的字符串进行拼接
//参数：指定的字符串，如果没有参数，相当于复制一个字符串
//返回值：返回拼接好的字符串（这个字符串是一个新串，原串是没有改变的）
console.log(String.fromCharCode(50)); //unicode编码
//功能 ：将指定的Unicode码转化为对应字符
//参数：指定的Unicode码
//返回值：返回指定Unicode码的字符
console.log(str.indexOf('23', 3)) //默认从0位置开始查找'2'，第二个参数可以指定从哪开始；****************
//功能 ：从原串当中找指定的子串的索引
//参数：可以是一个 也可以是2个   如果是一个，代表指定的子串 ，  如果是两个，后面要多加一个索引位置（代表从哪开始查找）
//如果只有一个参数，默认从左边0位置开始查找，如果是两个参数，默认从指定的那个索引开始查找
//返回值：返回指定的子串索引位置；如果指定的子串有多个只能返回一个；如果子串不存在返回-1；
//从左往右查找
console.log(str.lastIndexOf('23', 8)); //求字串位置,默认从末尾开始
//功能 ：从原串当中找指定的子串的索引
//参数：可以是一个 也可以是2个   如果是一个，代表指定的子串 ，  如果是两个，后面要多加一个索引位置（代表从哪开始查找）
//如果只有一个参数，默认从最右边位置开始查找，如果是两个参数，默认从指定的那个索引开始查找
//返回值：返回指定的子串索引位置；如果指定的子串有多个只能返回一个；如果子串不存在返回-1；
//从右往左查找
```

```
str = '12A345678aB';
console.log(str.localeCompare('12345678')); //比较大小     两个字符串直接可以使用条件运算符
//功能 ：比较指定的字符串和原串的大小
//参数：指定的字符串
//返回值：返回1，代表原串大   -1代表原串小   0 代表相等
//下面三个全部都是截取字符串（从原串里面截取子串）
console.log(str.slice(-6, -1)); //抽出字串  ********************
//功能 ：从原串里面截取指定索引之间的子串
//参数：指定的索引，开始位置和结束位置，但是结束位置的字符不包含   （参数可以是负数，-1代表最后一个），参数也可以不写（相当于截取所
//注意：传递的参数，前面的一定是左边的，后面的参数一定是右边的
//返回值：返回截取到的子串
console.log(str.substr(-5, 5)); //后面的参数是长度
//功能 ：从原串里面截取指定起始索引和指定长度的子串
//参数：前面的参数，代表的是起始索引（也可以是负数），后面的参数代表的是长度，参数也可以不写（相当于截取所有）
//返回值：返回截取到的子串
console.log(str.substring(4, 2)); //后面的参数不允许是负数
//功能 ：从原串里面截取指定起始索引和结束索引的子串（和slice功能类似）
//参数：不能是负数，这两个参数没有固定的谁在先谁在后，它会自动的去判断谁在左边谁在右边，参数也可以不写（相当于截取所有）
//返回值：返回截取到的子串
console.log(str.split('')); //*************************
//      //功能 ：以指定的字符为间隔，将字符串转化为数组（转化后指定的那个子串不存在了）
//参数：指定的子串，也可以不写；如果不写，那么以整个字符串为元素生成数组
//如果写的是空串，那么每个字符都会作为数组的元素生成数组
//返回值：返回生成的数组
console.log(str.toLocaleLowerCase());
//      //与 toLowerCase() 不同的是，toLocaleLowerCase() 方法按照本地方式把字符串转换为小写。
//只有几种语言（如土耳其语）具有地方特有的大小写映射，所有该方法的返回值通常与 toLowerCase() 一样
//功能 ：把字符串转化为小写，可以转化具有地方特色的字符串
//参数：无
//返回值：返回转化成小写的字符串
```

```
console.log(str.toLocaleUpperCase());
//功能 ：把字符串转化为大写，可以转化具有地方特色的字符串
//参数：无
//返回值：返回转化成大写的字符串
//下面两个和上面两个都是转大小写，只不过下面两个不关注具有地方特色的字符串
//      console.log(str.toLowerCase());     ****************************
//      console.log(str.toUpperCase());
console.log(str.valueOf());
//功能 ：从字符串的包装对象当中取基本值，取出来的就是字符串本身；（只有包装对象才能取出基本值，非包装对象的对象是取不出值得，还要进
//参数：无
//返回值：返回取出的基本值，就是字符串本身
console.log(str.toString());
//功能 ：把取出的基本值转化为字符串
//参数：无
//返回值：返回转化后的字符串，就是字符串本身
//以后等正则讲的时候去说
//      console.log(str.replace('1','**'));//替换字串
//      console.log(str.search())
//      console.log(str.match(/1/g));//找到匹配项返回数组
```

二、 string方法 ES6

1. includes(str) : 判断是否包含指定的字符串
2. startsWith(str) : 判断是否以指定字符串开头
3. endsWith(str) : 判断是否以指定字符串结尾
4. repeat(count) : 重复指定次数

三、 Array方法 ES5


```
arr.sort(function(a,b){
        return b - a;
    })
```

语法：array.sort(fun)；参数fun可选。规定排序顺序。必须是函数。

注：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，是按照字符编码的顺序进行排序。

如果想按照其他规则进行排序，就需要提供比较函数，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：

若 a 小于 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。若 a 等于b，则返回 0。若 a 大于 b，则返回一个大于 0 的值。

简单点就是：比较函数两个参数a和b，返回a-b 升序，返回b-a 降序，原数组发生改变

四、 Array方法 ES5 + ES6

ES5

1. Array.prototype.indexOf(value) : 得到值在数组中的第一个下标
2. Array.prototype.lastIndexOf(value) : 得到值在数组中的最后一个下标
3. Array.prototype.forEach(function(item, index){}) : 遍历数组

```
arr = [1,23];
arr.forEach(function(item,index){
    console.log(item);
})
```

4. Array.prototype.map(function(item, index){}) : 遍历数组返回一个新的数组，返回加工之后的值

map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

map() 方法按照原始数组元素顺序依次处理元素。

注意： map() 不会对空数组进行检测。

注意： map() 不会改变原始数组。

```
console.log(arr.map(function(item,index){
    return item * 2;
}))
```

5. Array.prototype.filter(function(item, index){}) : 遍历过滤出一个新的子数组， 返回条件为true的值

```
console.log(arr.filter(function(item,index){
    return item >= 2;
}))
```

ES6

1. Array.from(v) : 将伪数组对象或可遍历对象转换为真数组

```
var arr1 = '12345';
console.log(Array.from(arr1));
```

2. Array.of(v1, v2, v3) : 将一系列值转换成数组

```
console.log(Array.of(1,2,3));  //console.log(new Array(1,2,3));
```

3. find(function(value, index, arr){return true}) : 找出第一个满足条件返回true的元素

```
console.log(arr.find(function(item,index){
      return item > 10;
}));
```

4. findIndex(function(value, index, arr){return true}) : 找出第一个满足条件返回true的元素下标

```
console.log(arr.findIndex(function(item,index){
        return item > 10;
}));
```

所有的方法都要注意三要素；