**\*Javascript简称：**JS

\* **JS分三个部分**

\* 1 ECMAScript标准--基础的语法

\* 2 DOM Document Object Model 文档对象模型

\* 3 BOM Browser Object Model 浏览器对象模型

\* **什么是JS**

\* 是一门脚本语言

\* 是一门解释性语言

\* 是一门弱类型语言

\* 是一门基于对象的语言

\* 是一门动态类型的语言

\*

\* **动态页面**：页面由html+css+JS

\* 向服务器发送请求，服务器那边没有页面，是动态的生成，返回给客户端

\*

\* **js最初的目的**：解决用户和服务器之间的交互问题

\* js做特效，游戏，移动端，服务器

\*

\* **什么是对象**？

\* 1 对象是单个事物的抽象；

\* 2 对象是一个容器，封装了属性（property）和方法（method)，对象可以看成是一坨无序属性 的集合

\*

\* **编程思想**：

\* **面向过程**：所有的事情都是亲力亲为，注重的是过程

\* **面向对象**：提出需求，找对象，对象解决，注重的是结果

\* js不是一门面向对象的语言，是基于对象的语言，js来**模拟面向对象**

\* **面向对象的特性**：封装，继承，多态，（抽象性）

\* **封装**：就是包装，把一些重用的内容进行包装，在需要的时候，直接使用

\* 把一个值，存放在一个变量中，把一些重用的代码放在函数中，把好多相同功能的函数放在一个对象中，

把好多功能的对象，放在一个文件中，把一些相同的内容放在一个对象中

\***继承**：类与类之间的关系，js中没有类的概念，js中有构造函数的概念，是可以有继承的，是基于原型

\*

\* **多态**：同一种行为，针对不同的对象，产生不同的效果

\*

\* **OOP**：面向对象编程

**创建对象三个方式：**

1 字面量的方式 ------Object

2 调用系统的构造函数 ------Object

3 自定义构造函数方式

**工厂模式和自定义构造函数的区别：**

**\* 共同点**：都是函数，都可以创建对象，都可以传入参数

**\* 工厂模式**：函数名都是小写

有new

有返回值

new之后的对象是当前的对象

直接调用函数就可以创建对象



**\* 自定义构造函数：**

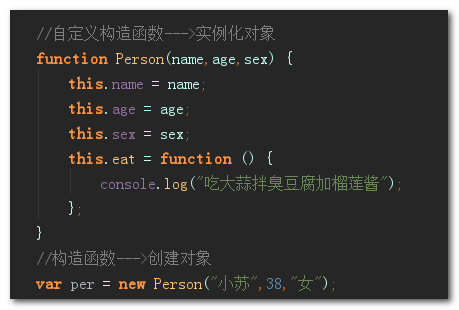
函数名是大写（首字母)

没有new

没有返回值

this是当前的对象

通过new的方式来创建对象



**自定义构造函数创建对象:4件事**

\* 1.在内存中申请一块空闲的空间,存储创建的对象

\* 2.this就是当前实例化的对象

\* 3.设置对象中的属性和方法(为对象添加属性和方法,为属性和方法赋值)

\* 4.把创建后的对象返回

\* 都是需要通过new的方式

**面向对象的思想**是 --- 抽象的过程 --->实例化的过程

**实例对象和构造函数之间的关系：**

\* 1.实例对象是通过构造函数来创建的---创建的过程叫实例化

\* 2.如何判断对象是不是这个数据类型？

\* 1）通过构造器的方式 实例对象.构造器 == 构造函数名字

*(dog.\_\_proto\_\_.constructor == Animal)*

\* 2）对象instanceof构造函数名字

**原型?（**实现数据共享,继承, 都是为了节省内存空间**）**

\* 实例对象中有\_\_proto\_\_这个属性,叫原型,也是一个对象,这个属性是给浏览器使用,不是标准的属性----->\_\_proto\_\_----->可以叫原型对象

\* 构造函数中有prototype这个属性,叫原型,也是一个对象,这个属性是给程序员使用,是标准的属性------>prototype--->可以叫原型对象

\*

\* 实例对象的\_\_proto\_\_和构造函数中的prototype相等--->true

\* 又因为实例对象是通过构造函数来创建的,构造函数中有原型对象prototype

\* 实例对象的\_\_proto\_\_指向了构造函数的原型对象prototype

\* 构造函数中的prototype里面的属性或者方法,可以直接通过实例对象调用

\* 正常的写法:实例对象.\_\_proto\_\_才能访问到构造函数中的prototype中的属性或者方法

\* per.\_\_proto\_\_.eat();//\_\_proto\_\_不是标准的属性

\* per.eat();

\* 原型就是属性,而这个属性也是一个对象

\* Person.prototype--->是属性

\* Person.prototype.属性或者Person.ptototype.方法()

**构造函数和实例对象和原型对象之间的关系**

\* 1.构造函数可以实例化对象

\* 2.构造函数中有一个属性叫prototype，是构造函数的原型对象

\* 3.构造函数的原型对象（prototype）中有一个constructor构造器，这个构造器指向的就是自己 所在的原型对象所在的构造函数

\* 4.实例对象的原型对象（\_\_proto\_\_）指向的是该构造函数创建的原型对象

\* 5.构造函数的原型对象（prototype）中的方法是可以被实例对象直接访问的

**什么样子的数据是需要写在原型中？**

需要共享的数据就可以写原型中

原型的作用之一：数据共享

属性需要共享，方法也需要共享

不需要共享的数据写在构造函数中，需要共享的数据写在原型中

**原型的写法:**

\* 构造函数.prototype.属性=值

\* 构造函数.prototype.方法=值---->函数.prototype,函数也是对象,所以,里面也有\_\_proto\_\_

\* 实例对象.prototype-------->实例对象中没有这个属性,只有\_\_proto\_\_(暂时的)

**简单的原型的写法**

\* 缺陷:--->新的知识点---->原型直接指向{}---->就是一个对象,没有构造器

\* 构造函数.prototype={

\* 切记:如果这这种写法,要把构造器加上

\*

\* };



**实例对象的方法，是可以相互调用的**

**原型中的方法，是可以相互访问的**

**原型链:**是一种关系,实例对象和原型对象之间的关系,关系是通过原型(\_\_proto\_\_)来联系的

divObj.\_\_proto\_\_---->HTMLDivElement.prototype的\_\_proto\_\_--->

HTMLElement.prototype的\_\_proto\_\_---->Element.prototype的\_\_proto\_\_---->

Node.prototype的\_\_proto\_\_---->EventTarget.prototype的\_\_proto\_\_---->Object.prototype没有\_\_proto\_\_,所以,Object.prototype中的\_\_proto\_\_是null

**原型指向可以改变**

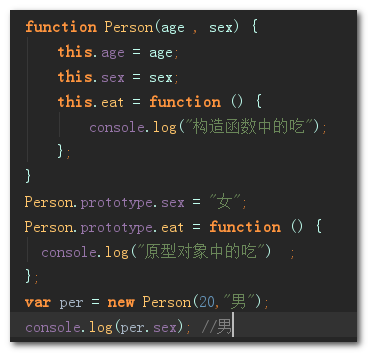
实例对象的原型\_\_proto\_\_指向的是该对象所在的构造函数的原型对象

构造函数的原型对象(prototype)指向如果改变了,实例对象的原型(\_\_proto\_\_)指向也会发生改变

**注意：**实例对象使用的属性或者方法，先在实例对象中找，找到了则直接使用，

找不到的话去实例对象的\_\_proto\_\_指向的原型对象prototype中找，

找到则使用，找不到则报错



**函数的自调用---自调用函数**

一次性的函数--声明的同时，直接调用了

(function () {

console.log("函数");

})();

//页面加载后，这个自调用函数的代码就执行完了

(function (形参) {

var num = 10 ;//局部变量

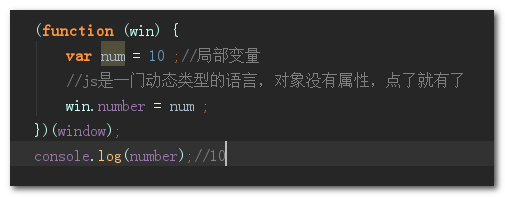
console.log("哈哈");

})(实参);

console.log(num);//未定义

**如何把局部变量变成全局变量**

把局部变量给window就可以了



**对象.bind(参数)**;---->改变this的指向

**函数中的this的指向**

\* 普通的函数中this是window

\* 构造函数中的this,构造函数一般都是创建实例对象使用的,是通过new关键字,构造函数也是函数

\* 构造函数中的this是实例对象

\* 方法中的this是实例对象

\* 原型中的方法中的this是实例对象

\* 定时器中的this是window

**对象是不是函数呢?（**不一定）

\*函数是对象,构造函数也是函数,也是对象

\* 对象中有\_\_proto\_\_

\* 函数中有prototype

\*

\* Math是对象,但不是函数

**继承的方法**

\* 原型继承:改变原型的指向

\* 借用构造函数继承:主要解决属性的问题，方法继承不了

\* 组合继承:原型继承+借用构造函数继承

既能解决属性问题,又能解决方法问题

\* 拷贝继承:就是把对象中需要共享的属性或者犯法,直接遍历的方式复制到另一个对象中

**apply和call的使用方法**

\* apply的使用语法

\* 函数名字.apply(对象,[参数1,参数2,...]);

\* 方法名字.apply(对象,[参数1,参数2,...]);

\* call的使用语法

\* 函数名字.call(对象,参数1,参数2,...);

\* 方法名字.call(对象,参数1,参数2,...);

\*

\* 作用:改变this的指向

\* 不同的地方:参数传递的方式是不一样的

\*

\* 只要是想使用别的对象的方法,并且希望这个方法是当前对象的,那么就可以使用apply或者是call的 方法改变this的指向

**bind的使用方法**

复制一个方法或者是函数,是在复制的同时改变了this的指向

\* 函数名字.bind(对象,参数1,参数2,...);---->返回值是复制之后的这个函数

\* 方法名字.bind(对象,参数1,参数2,...);---->返回值是复制之后的这个方法

**函数中的几个成员**

函数中有name属性-->函数名字，name属性只读，不能修改

函数中有一个arguments属性--->实参的个数

函数中有一个length属性-->形参的个数,函数定义的时候形参的个数

函数中有一个caller属性---->调用(f1函数在f2函数中调用的,所以,此时调用者就是f2)

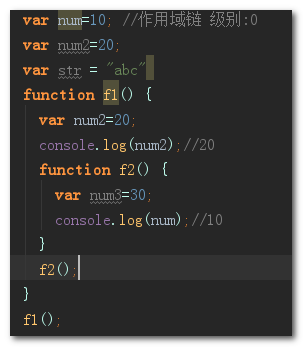
**高阶函数**

函数的使用的方式:函数作为参数使用，函数作为返回值使用

函数作为参数使用的时候,这个函数可以是匿名函数,也可以是命名函数

**作用域链**

变量的使用,从里向外,层层的搜索,搜索到了就可以直接使用了层层搜索,搜索到0级作用域的时候, 如果还是没有找到这个变量,结果就是报错。



**预解析**

就是在浏览器解析代码之前,把变量的声明和函数的声明提前(提升)到该作用域的最上面。

**闭包**

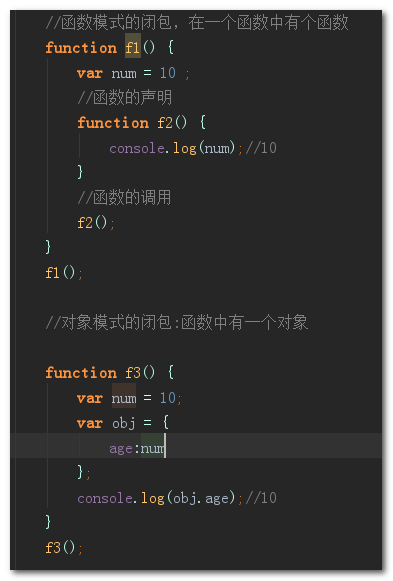
\* 概念：函数A中，有一个函数B，函数B中可以访问函数A中定义的变量或者是数据，此时形成闭包

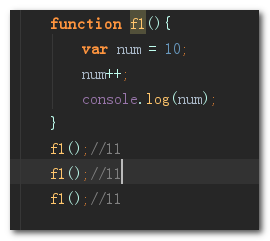
\* 作用：缓存数据，延长作用域链

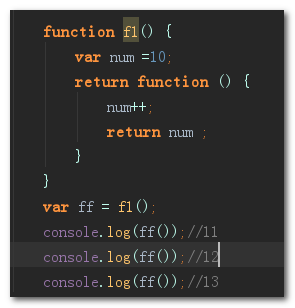
\* 优点和缺点：缓存数据

\* 模式：函数模式的闭包，对象模式的闭包

总结:如果想要缓存数据,就把这个数据放在外层的函数和里层的函数的中间位置

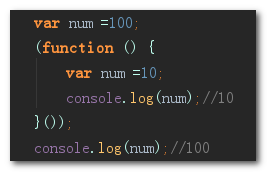






**沙箱:**

环境,黑盒,在一个虚拟的环境中模拟真实世界,做实验,实验结果和真实世界的结果是一样,但是不 会影响真实世界



**递归**

函数中调用自己，此时就是递归，递归一定要有结束条件;

一般应用在遍历上，递归轻易不要用,效率很低,



/\*

\* 上述代码执行过程:

\* 代码执行getSum(5)--->进入函数,此时的x是5,执行的是5+getSum(4),此时代码等待

\* 此时5+getSum(4),代码先不进行计算,先执行getSum(4),进入函数,执行的是4+getSum(3),等 待, 先执行的是getSum(3),进入函数,执行3+getSum(2),等待,先执行getSum(2),进入函数,执行 2+getSum(1);等待, 先执行getSum(1),执行的是x==1的判断,return 1,所以,

\* 此时getSum(1)的结果是1,开始向外走出去

\* 2+getSum(1) 此时的结果是:2+1

\* 执行:

\* getSum(2)---->2+1

\* 3+getSum(2) 此时的结果是3+2+1

\* 4+getSum(3) 此时的结果是4+3+2+1

\* 5+getSum(4) 此时的结果是5+4+3+2+1

\* 结果:15

\* \*/

**浅拷贝**：

拷贝就是复制，就相当于把一个对象的所有内容，复制一份给另一个对象，直接复制，

或者说，就是把一个对象的地址给了另一个对象，他们指向相同，两个对象之间有共同的属 性或者方法，都可以使用

**深拷贝**:

拷贝还是复制,深:把一个对象中所有的属性或者方法,一个一个的找到.

并且在另一个对象中开辟相应的空间,一个一个的存储到另一个对象中

**正则表达式**

\*概念：也叫规则表达式，按照一定的规则组成的一个表达式，这个表达式的作用主要是匹配字符串的

\* "我的电话:10086,他的电话:10010,你的电话:10000" 正则表达式,把这个字符串中的所有的数字找到

\* 正则表达式的作用:匹配字符串的

\* 在大多数编程语言中都可以使用

\* 正则表达式的组成：是由元字符或者是限定符组成的一个式子

**元字符**

\* . 表示的是：除了\n以外的任意的一个字符 “gjskda362”

\*

\* [] 表示的是：范围 ，[0-9] 表示的是0到9之间的任意的一个数字, "789" [0-9]

\* [1-7] 表示的是1到7之间的任意的一个数字 [100-200]错误的，要写成[1][0-9][0-9]

\* [a-z] 表示的是:所有的小写的字母中的任意的一个

\* [A-Z] 表示的是:所有的大写的字母中的任意的一个

\* [a-zA-Z] 表示的是:所有的字母的任意的一个

\* [0-9a-zA-Z] 表示的是: 所有的数字或者是字母中的一个

\* [] 另一个含义：把正则表达式中的元字符的意义干掉 [.]就是一个 .

\* | 或者 [0-9]|[a-z] 表示的是要么是一个数字,要么是一个小写的字母

\* () 分组 提升优先级 [0-9]|([a-z])|[A-Z]

\* ([0-9])([1-5])([a-z]) 三组, 从最左边开始计算

\* (()(()))

\*

\*

\* 都是元字符,但是也可以叫限定符,下面的这些（限定符:限定下面表达式出现的次数）

\* \* 表示的是:前面的表达式出现了0次到多次

\* [a-z][0-9]\* 小写字母中的任意一个 后面是要么是没有数字的,要么是多个数字的

\* "fdsfs3223323" [a-z][0-9]\* //可以匹配

\*

\* + 表示的是:前面的表达式出现了1次到多次

\* [a-z][9]+ 小写字母一个后面最少一个9,或者多个9

\* "fesfewww9fefds" //可以匹配

\*

\* ? 表示的是:前面的表达式出现了0次到1次,最少是0次,最多1次 ,另一个含义:阻止贪婪模式

\* [4][a-z]? "1231234ij" //可以匹配

\*

\* 限定符:限定前面的表达式出现的次数

\* { } 更加的明确前面的表达式出现的次数

\* {0,} 表示的是前面的表达式出现了0次到多次,和 \*一样的

\* {1,} 表示的是前面的表达式出现了1次到多次,和 +一样的

\* {0,1} 表示的是前面的表达式出现了0次到1次,和 ?一样的

\* {5,10} 表示的是前面的表达式出现了5次到10次

\* {4} 前面的表达式出现了4次

\* {,10} 错误的========不能这么写

\*

\* ^ 表示的是以什么开始,或者是取非(取反) ^[0-9] 以数字开头

\* ^[a-z] 以小写字母开始

\* [^0-9] 取反,非数字

\* [^a-z] 非小写字母

\* [^0-9a-zA-Z\_] \_不算特殊符号

\*

\* $ 表示的是以什么结束 [0-9][a-z]$ 必须以小写字母结束

\* ^[0-9][a-z] ^$相当于是严格模式 "3f2432e" "4f"

\*

\* \d 数字中的任意一个,相当于[0-9]

\* \D 非数字中的一个

\* \s 空白符中的一个

\* \S 非空白符

\* \w 非特殊符号

\* \W 特殊符号

\*

\* \b 单词的边界

\* "what are you no sha lei"

**创建正则表达式对象**（两种）

1.通过构造函数创建对象

2.字面量的方式创建对象

**正则表达式的作用：**

匹配字符串

**邮箱的正则表达式**

[0-9a-zA-Z\_.-]+[@]+[0-9a-zA-Z\_.-]+([.][a-zA-Z]+){1,2}

**中文的正则表达式**

[\u4e00-\u9fa5] [一-龥]

**正则表达式中:**

g 表示的是全局模式匹配

i 表示的是忽略大小写

**伪数组和数组的区别**

真数组的长度是可变的

伪数组的长度不可变

真数组可以使用数组中的方法

伪数组不可以使用数组中的方法