# 本章内容

本章内容：共7小节，28个知识点

第1节：javascript简介

第2节：javascript语法基础第3节：面向对象编程

第4节：本地以及内置对象第5节：DOM标准

第6节：事件监听第7节：BOM对象s第8节：ajax请求

# 本章目标

掌握javascript的基本语法，掌握基本数据类型与对象数据类型。 掌握js的流程控制语法与js语法特性；

了解js运行流程与数据存储机制；

熟练使用js的本地以及内置对象，熟练调用常用api接口； 了解DOM标准以及htmlDOM标准，掌握常用的dom接口；

了解事件流，掌握事件绑定机制，掌握事件对象常用的api接口； 了解BOM标准，掌握常用的BOM接口；

掌握ajax的技术组成，掌握JSON数据格式以及数据传输方式；

# javascript简介

**javascript简介**

javascript的发展历程

Netscape（网景）公司在开发出LiveScript脚本语言，当时的目的是在浏览器中使用该脚本操 纵页面上的元素完成页面的特殊效果。

LiveScript后期由NetScape和Sun公司共同完成，而且LiveScript脚本语言遵照的一个基本原 则就是尽可能的像java，后命名为javascript。

微软在IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 Jscript。

在ECMA（欧洲计算机制造商协会）的协调下，由Netscape、Sun、微软、Borland组成的工 作组确定统一标准：ECMA-262，规范了脚本语言的定义及使用方式，并将遵循该规范的脚本语言称为ECMAScript 。

javascript的使用场景

嵌入动态文本于HTML页面，对浏览器事件做出响应。读写HTML元素。

在数据被提交到服务器之前验证数据。检测访客的浏览器信息。

控制cookies，包括创建和修改等。基于Node.js技术进行服务器端编程。

Javascript的基本特点

脚本语言：JavaScript是一种解释型的脚本语言,C、C++等语言先编译后执行,而JavaScript可以直接执行。

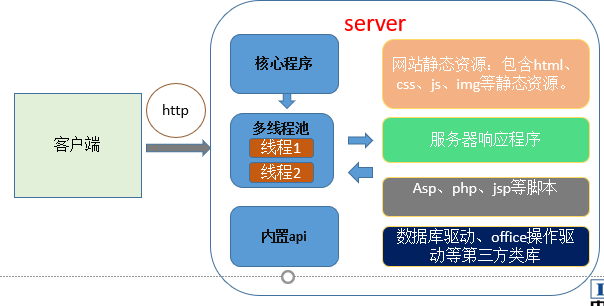
基于对象：JavaScript是一种基于对象的脚本语言,它不仅可以创建对象,也能使用现有的对 象。（后面章节介绍）

简单:JavaScript语言中采用的是弱类型的变量类型,对使用的数据类型未做出严格的要求,是基 于Java基本语句和控制的脚本语言,其设计简单紧凑。

跨平台：JavaScript只依赖于浏览器而与操作系统无关，目前JavaScript已被大多数的浏览器所支持。

## w eb开发执行流程

web服务器软件：一般指网站服务器，第三方计算机的程序，可解析http请求，并调用服务器程序完成http响应。



浏览器端js执行基础组成：

Js核心语法：包含js的数据类型、函数封装、内置对象与本地对象。

事件：事件是使用者对浏览器当前显示界面的操作，例如点击、鼠标移动、敲击键盘都是事 件。

DOM：文档对象模型（Document Object Model，简称DOM），现在可以简单的理解为浏览器供应商提供的操作页面元素的方式。Javascript并不能直接操作页面元素（例如拖拽等功 能），但可以调用DOM完成。

BOM： BOM(Browser Object Mode) 是指浏览器对象模型，是浏览器供应商提供的操作页面窗口的方式（例如打开新窗口、全屏显示等）。

## j s的引入方式

javascript语言如果直接编写在html文件上，浏览器将会将他的内容当作html代码执行，而不是javascript代码。常见的引入javascript的方式有三种：

内嵌javascript脚本

引入外部的javascript文件

在事件中编写javascript(不推荐)。模块化引入（详见VUE章节）

在script标签中加入js程序（内嵌javascript脚本）



内嵌js脚本方式中script标签可用属性说明：

type:声明脚本的语言类型。所有浏览器默认都是javascript(可以缺省) charset：声明脚本的编码类型，很多浏览器都忽略此属性，而使用meta标签定义的编 码类型解析js代码。（不建议使用此属性）

引入外部的javascript文件

<script src="./myscripty.js" type="application/javascript"></script>

在事件中编写javascript(不推荐)。

<input type="button" value="点我" onclick="alert('helloworld')">

alert('helloworld') 是按钮点击事件要执行的js脚本

## 入 门程序

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title></title>

<script type="text/javascript"> document.write("javascript 我来了")

</script>

</head>

<body>

</body>

</html>

document.write():是DOM接口，可以改变html文档内的内容。javascript 我来了：这几个字符是javascript的变量。

## 本 节小节

js实际上标准的名称是ECMASCRIPT、他的前身是javascript

Js的主要场景是在网页中访问以及操作网页的元素，在nodejs中也可以调用封装的接口做服务器端开发。

引入js的方式有三种，模块化引入方式是现在前端模块化开发的主要模式。

# 变 量与数据类型

## 基 本语法规则

注释：编写在程序中作为程序的说明，以便于以后的参考、修改。在运行程序时不做处理。具备如 下特点：

被注释的程序块不会执行.

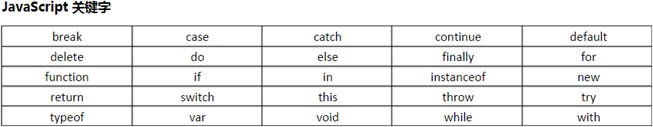
注释可以提高代码的可读性，因此添加注释是编程的好习惯。

javascript的注释虽然不执行，但也会下载到客户端，因此在正式上线之前应使用其他工具去除掉全部的注释。

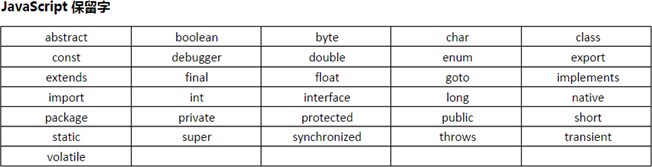
编写方式



关键字：目前在javascript语言中已经具有语法含义的英文词语，每个关键字都具有不同的使用场景和规则。



保留字：预计未来扩展javascript功能时会拓展的关键字。



标识符就是函数、变量的名字，由开发者自己起名。

var m;//变量

function fun(){} //函数

命名规则：

不能使用javascript的关键字、保留字。

必须以字母、下划线（\_）或美元符（$）开始。

后续的字符可以是字母、数字、下划线或美元符（数字是不允许作为首字符出现的）。

## 变 量

变量的概念：变量是存储信息的容器，变量可以根据变量名进行访问以及赋值。

var关键字：声明变量的关键字，如果在函数外部不使用var关键字，变量对应的内存地址将不会被浏览器回收，除非当前页面被关闭。

声明变量的方式

var name="tjetc",age=56,job="CEO"; //声明多个变量

//声明变量

//声明变量并赋值

var x;

var x=2;

修改变量的方式，var只在声明的时候被使用一次，修改变量的值直接使用变量的名字进行赋值。

//修改变量的值

//相当于定义了一个重名变量

var x=2; x=3;

var x=3;

变量重名规则：程序中要保证定义的多个变量不重名，如果重名并不像其他高级语言那样无法通过 编译，而是采用如下规则

如果两次变量的声明都对应赋值表达式，则后执行的变量会覆盖先执行的变量。

<script type="text/javascript"> var m = 3;

var m= 5;

//弹出框函数alert(m); //显示5

</script>

如果第一次变量的声明对应赋值表达式，第二次变量声明不对应赋值表达式，则第二次变量声 明无效。

<script type="text/javascript"> var m = 3;

var m;

//弹出框函数alert(m); //显示3

</script>

## 数 据类型简介

数据类型分类：

基本数据类型对象类型

基本数据类型

js引擎默认支持的数据类型，除了string类型外其余类型都占用固定内存空间。基本数据类性的分类

undefined （未被初始化）

null类型 （null即是一个不存在的对象的占位符）

boolean类型number类型string类型

<script type="text/javascript"> var m;

//弹出框函数

alert(m); //显示undefined var k=null;

//弹出框函数alert(k); //显示null

</script>

对象数据类型

本质上是一组无序的名值对，占用内存大小可以变化。

var c ={aa=11,bb=22}

对象类型分类

内置类型：如数组、日期类型。浏览器内置对象，可以直接使用。用户自定类型：开发者自定义的对象。

对象类型、基本类型之间的区别

定义方式不同赋值方式不同

内存使用方式空间

## t ypeof简介

typeof：获取目标变量的数据类型，以字符串形式返回，typeof是系统内置函数

typeof的局限性：typeof只能区别出5个基本类型和对象类型，但无法识别对象类型的继承关系， 以及对象类型赋值方式。

<script type="text/javascript"> var aa = 1;

var bb ={

a:1,

c:{mm:22}

}

alert("aa的数据类型为："+typeof(aa)); alert("bb的数据类型为："+typeof(bb));

</script>

## u ndefined类型

undefined类型：是变量的默认赋值，表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法：

变量被声明了，但没有赋值时，就等于undefined。

调用函数时，应该提供的参数没有提供，该参数等于undefined。函数没有返回值时，默认返回undefined。

var x;

alert(x);//此处运行的结果为undefined类型

## n ull类型

null类型：表示"没有对象"，即该处不应该有值。

var m=null;

alert(m); //此处运行的结果为null

null类型的特点：

undefined是null的衍生物，早期的null代表对象类型的数据为空。null可以参与运算，在参与运算时js引擎会给null默认值。

null转为数值时为0。undefined与null的区别：

含义不同：null代表不应该有值，undefined应该有值，但还没赋值。

null转为数值时为0；undefined转为数值时为NaN。

/\* Number 对象是原始数值的包装对象 \*/ alert(Number(undefined)); //结果为NaN alert(Number(null)); //结果为0

## b oolean类型

boolean类型:只有两个取值范围，代表真、假。取值范围:true、false。

程序示范

var a = true; var b = false;

## n umber类型

number类型:数字类型，可以表示整数和浮点数。

var a = 11; var b = 11.5;

number支持多进制赋值但最终还是以二进制存储

var a = 11;//默认为十进制

var b = 010;//以0开头代表8进制，如果080则扔以10进制解析var c = 0xAF;//以0x开头代表16进制

number类型的浮点数表示方式

var b=0.1;

var c=.1;//可以省略0，但是不建议这么使用

number类型的范围

-(2^53-1)= -9007199254740991

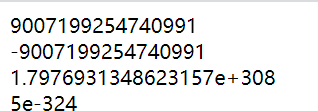
2^53-1 = 9007199254740991

number类型浮点数范围

最大值：0 (1.7976931348623157e+308)

最小精度值：5e-324

Number对象的常量可以获取number类型的范围



//<br/>换行document.write(Number.MAX\_SAFE\_INTEGER+"<br/>"); document.write(Number.MIN\_SAFE\_INTEGER+"<br/>"); document.write(Number.MAX\_VALUE+"<br/>"); document.write(Number.MIN\_VALUE+"<br/>");

number类型转换

通过内置函数

paraseInt('字符',进位数); paraseFloat('字符',进制数);

从字符串的第一个有效数字开始到第一个无效字符结束的内容进行数字转换。 对于无有效数字则返回NaN类型(Not A Number);

//按照10进制转换alert(parseInt("20",10));//20

//按照16进制转换alert(parseInt("a",16))//10 alert(parseInt("abc20",10));//NaN

alert(parseInt("20abc",10));//20 alert(parseFloat("3.254",10));//3.254

通过内置Number对象

var a = Number("11");//不能识别有效数字，不建议使用

number类型其他取值

NaN：非数字。当执行parseInt无有效数字时返回Infinity：无穷。当超过浮点类型最大范围时

## S tring类型

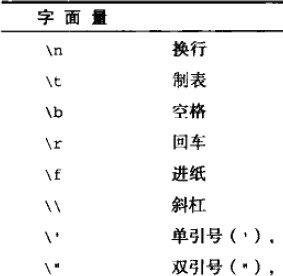
string类型：字符串类型，可表示单个字符或多个字符的组合,采取可变内存的方式存储。编写方式

//字符串可以使用双引号也可以使用单引号var x = "111";

var y = 'ddd';

var c = "";//代表空字符

转义字符：用一些普通字符的组合来代替一些特殊字符



## 对 象类型

对象类型：或称引用类型，除了基本类型之外的都是引用类型或对象类型。

var aa = {

b:"33", c:22

};

alert(aa.b);

alert(aa.c);

对象类型本质上是多个基础数据类型的数据封装。

与基础数据类型相比对象类型在内存存储结构、访问方式、声明方式都不同。

## 本 节总结

注释：编写在程序中作为程序的说明，以便于以后的参考、修改。在运行程序时不做处理

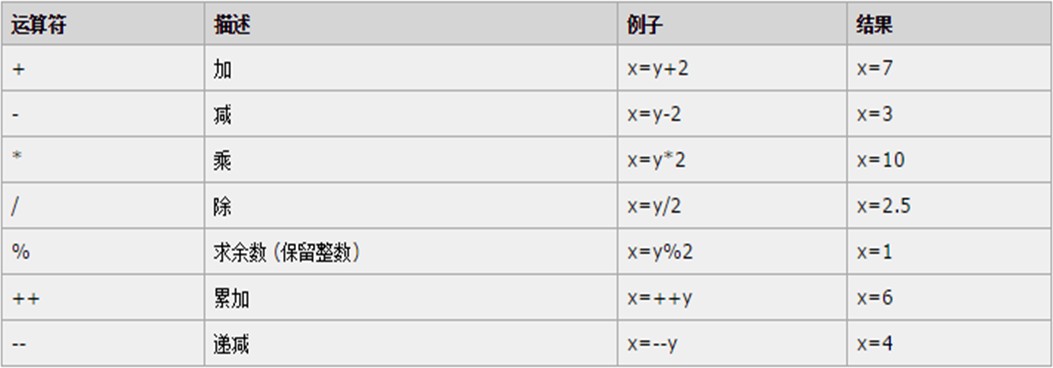
关键字目前在javascript语言中已经具有语法含义的英文词语，每个关键字都具有不同的使用场景和规则。保留字：预计未来扩展javascript功能时会拓展的关键字。

基本数据类性包含：undefined、null类型、boolean类型、number类型、string类型

# 运 算符

## 运 算符类型

算数运算符：用于进行算术运算(arithmetic operators) 符号，就是用来处理四则运算的符号，一般用于数据的计算以及字符串的拼接。



赋值运算符用于给 JavaScript 变量赋值。



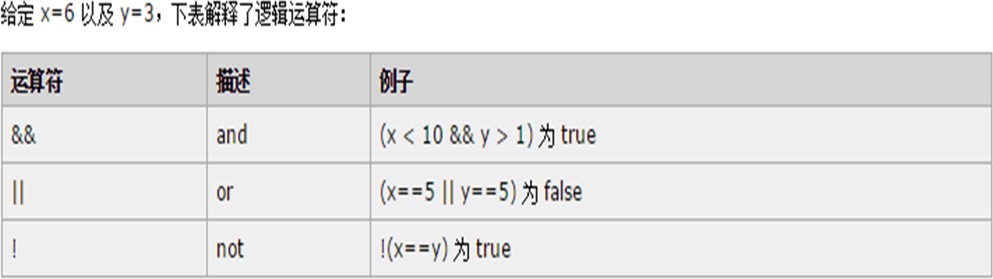
比较运算符：<、>、>=、<=等操作符，用于对比数据的大小或编码。

如果操作符两侧为number类型的数据，返回比较结果的boolean值。

如果一个操作数为字符另一个为number类型，则将字符转换为数字在做比较，返回比较结果的boolean值。

如果两个操作数都为字符，则比较首字母编码值，若相等比较第二个字母的编码值，以此类 推。

逻辑运算符：逻辑运算符用于连接boolean类型的值或值为boolean类型的表达式，用于逻辑判断。

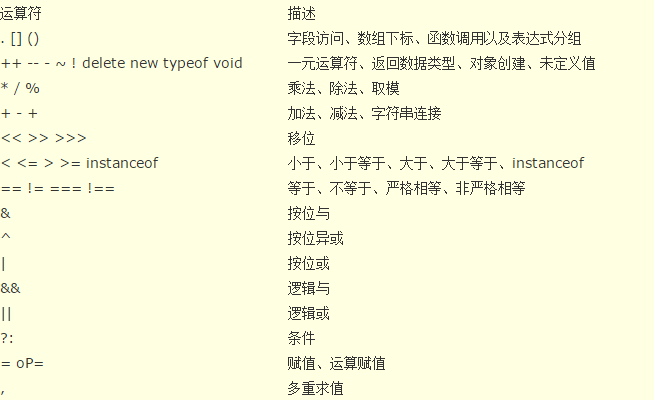


条件运算符：条件运算符是一个三目运算符，用于连接三个变量或表达式，最终返回一个计算结果 值。语法格式与java相同。

var a = 10; var b = 8;

alert("ab的最大值为:"+(a>b?a:b));

## 运 算符的优先级别



### 有相同优先级的运算符按从左至右的顺序求值，但注意，这些顺序不需要特殊记忆，编程时对于优先 级别不确定的，用（）来调节。

**本 节总结**

算术运算符的作用是完成基本的数学运算的符号，比如数字的处理使用到算术运算符号。 赋值运算符它的作用是将一个表达式的值赋给一个变量。

比较运算符用来比较数据的大小等使用的运算符。

逻辑运算符用于连接boolean类型的表达式，计算最终逻辑值。

# 流 程控制语句

## 判 断语句

if语句语法规范：

if(条件表达式){

//只有条件为true，执行代码

}

在javascript中对于条件表达式会默认使用Boolean类进行转换，例如if(undefined)会返回否。

var a;

if(!a) //if(a)会返回false，因为a为undefined

只有当指定条件为 true 时，使用该语句来执行代码。

如果省略{}，则如果条件为true，则if语句下的第一行有效程序将会被执行，但为了程序的可读性和健壮性，要求必须加{}

//不建议如下写法if(条件表达式)

//执行的一行代码

if...else if. else 语句 - 使用该语句来选择多个代码块之一来执行

if(条件1){

//代码逻辑1

}else if(条件2){

//代码逻辑2

}else{

//代码逻辑3

}

## s witch语句

switch基本语法

switch(值){

case:值1{

代码块1 break;

}

case:值2{

代码块2 break;

}

......

default:{

所有的条件都不满足执行此代码块

}

}

## d o-while语句

ｄo-while 语句(后测试循环) :退出条件在执行循环内部的代码之后计算。这意味着在计算表达式之前，至少会执行循环主体一次。不常用，都可以使用while语句、for语句替换。

基本语法：

do{

执行的代码块

}while(条件)

## w hile语句

基本语法

while(条件){

执行的代码块

}

var i =1; while(i<10){

if(i>8){

break;

}

document.write(i+"<br/>"); i++;

}

## f or语句

基本语法

for(赋值表达式;判断表达式;程序语句){

代码块

}

赋值表达式可以声明一个变量，此变量在循环体内生效且只在for语句中第一次执行。执行后执行for语句中的内容。

执行程序语句

执行判断表达式，如果为true继续从判断表达式执行，如果为false循环体结束

for(var i=0;i<10;i++){ if(i%2==0){

continue;

}

document.write(i+"<br/>");

}

continue:用于声明跳出本次循环，忽略continue后的程序执行。break：用于声明跳出循环体。

continue和break在所有的循环都适用

## 本 节总结

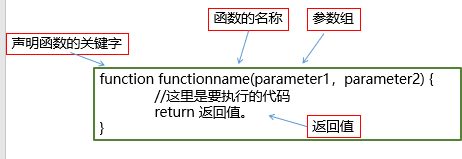
在js中也通过if…else关键字来进行判断的流程控制。

通常使用while、for语句对数组、对象类型的数据进行循环访问。在循环中通过continue、break关键字来控制循环的执行。

# 函 数

## 函 数的声明方式

函数：一段包含应用程序的程序块，此程序块可以被其他程序通过函数名调用，目的是提高代码公 用性、可维护性。

函数定义语法（一种方式）：

调用方式：functionname(实际参数值);

函数体内的程序在函数定义时并不执行，只有在调用函数并传递给有效的参数后才能执行函数 体内的程序。

function:关键字，在定义函数时使用function来声明当前程序块为函数。function在因为使用 场景不同，作用也不同。

functionname：函数的名称，在执行函数时，需要使用函数名调用函数中的程序块。命名规则如下：

符合javascript标识符的命名规则。

首字母小写，多个单词组合时除首单词外，其余单词的首字母大写。可以巧妙的使用缩写，例如rowNum表示行数。

parameter：形式参数，在定义函数时，该参数并没有实际的数值，因此称为形式参数，形式参数为 0-多个，用逗号分开。函数体内可将形参作为变量使用。

### 重名规则如果在一个网页中出现多个同名函数（不区分参数）,则最后定义的函数将覆盖之前的函数

代码示例

//有参函数

function test(a,b){ return "两个参数的函数";

}

//无参函数function test(){

return "没有参数的函数";

}

//上述的函数名称相同，后面的函数名称会覆盖前面的函数var result = test(1,2);

alert(result);//执行结果：没有参数的函数

js中调用函数时，实际传递的参数叫实参，实参的数量可以与形参的数量不同，缺省的实参在函数 体内为undefined。

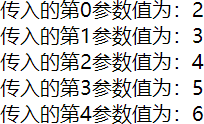
function fun1(param1,param2){ document.write(param1+"<br/>"); document.write(param2+"<br/>");

}

//调用函数fun1(1);



在函数体内可以使用arguments[i]来获取第i个位置的实际参数值。



//arguments对象是比较特别的一个对象，实际上是当前函数的一个内置属性function fun2(){

var len=arguments.length; for(var i=0;i<len;i++){

document.write("传入的第"+i+"参数值为："+arguments[i]+"<br/>");

}

}

//调用函数fun2(2,3,4,5,6);

利用arguments和typeof实现函数的重载

return : 返回，函数执行后返回给调用程序的值，一般为执行结果，

例如如下程序计算出1+2+。。N的值。

function sum(n){ var sum = 0;

for(var i=1;i<=n;i++){ sum+=i;//等价 sum=sum+i;

}

return sum;

}

//控制台输出console.log(sum(5));

定义函数的方式

普通函数方式：

function functionName(paramter) {}

匿名函数：（实际是匿名函数赋值给了window对象的onload属性）

window.onload = function (paramter){}

函数表达式：（实际是匿名函数赋值给了变量f）

var f = function(paramter) {}

Function对象：不建议使用，但可以帮助我们理解js的核心var b = new Function('a','b','return a+b');

等价 function b(a,b){return a+b}

## F unction对象简介

函数的本质：函数是Function的一个实例对象，函数名是对象实例的变量名称，函数不存在重载。

var b = new Function("a","b","return a+b");

console.log(b instanceof Function); //true 说明b是Function实例

console.log(b.constructor===Function); //true

1、对于string,number等基础类型，==和===是有区别的

1. 不同类型间比较，==之比较“转化成同一类型后的值”看“值”是否相等，===如果类型不同，其结果就是不等
2. 同类型比较，直接进行“值”比较，两者结果一样

2、基础类型与高级类型，==和===是有区别的

1. 对于==，将高级转化为基础类型，进行“值”比较
2. 因为类型不同，===结果为false

函数的预编译：浏览器在客户端加载时将会检索全局函数，并将其使用new Functon方式进行初始化，因此如下程序可以运行。

test1(); function test1(){

console.log("test1执行了");

}

## 本 节总结

函数：一段包含应用程序的程序块，此程序块可以被其他程序通过函数名调用，目的是提高代码公 用性、可维护性。

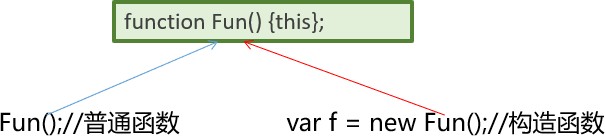
函数的本质：函数是Function的一个实例对象，函数名是对象实例的变量名称，函数不存在重载。函数的预编译：浏览器在客户端加载时将会检索全局函数，并将其使用new Functon方式进行初始化，因此如下程序可以运行。

# 面 向对象编程

## 构 造函数

构造函数：构造函数的声明方式与普通函数完全一致，但调用方式不同，返回的结果及内存结构也 不同，执行构造函数将会返回一个该构造函数对应的对象。

构造函数的定义(与普通函数定义完全一致，但使用构造函数一般替代类，因此要符合java的规范)



function hello(){

console.log("我是普通函数hello");

}

//普通函数与构造函数定义一样function Person(){

//this指的是对象实例本身this.name="zhangsan"; console.log("Person的构造函数执行了");

}

//构造函数使用（适用new关键字，跟java一样） var p = new Person();

//普通函数使用hello();

构造函数初始化时的执行过程创建了一个空对象obj。

空对象的原型链（ proto ）指向了函数的原型对象prototype对象。

修改函数对象的this指针为新创建的对象引用。执行函数体。

function hello(){

console.log("我是普通函数hello");

}

//普通函数与构造函数定义一样function Person(){

//this指的是对象实例本身this.name="zhangsan"; console.log("Person的构造函数执行了");

}

//构造函数使用（适用new关键字，跟java一样） var p = new Person();

//普通函数使用hello();

//Person.prototype为Person函数的原型对象(prototype) console.log("Person.prototype 如 下 ：") console.log(Person.prototype);

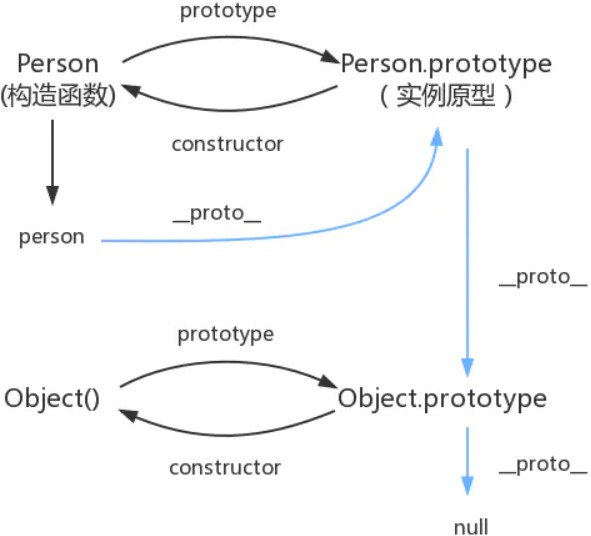
//p为实例化Person后的对象，p. proto 为p对象原形链console.log("p.\_\_proto\_\_为 如 下 ：") console.log(p.\_\_proto\_\_);

//Person.prototype与p. proto 指的是同一个对象

//再看下 Person. proto 是什么console.log(Person.\_\_proto\_\_);



原形链图



## 原 型链与原型对象

原型对象(prototype)：是**函数对象**（当且仅当函数对象，如：new Function()，function fun1()

{}）的一个内部属性，值是一个普通的对象，用户可以访问、操作该对象。

原型链( proto )：**每个对象**都具备的不可访问的内部属性（指针）（chrome等浏览器可以访问，但无法操作）,指针指向构造函数对应的原型对象。

原型链的特点：

原型链的终点是Object对象的prototype属性，该属性的 proto 为null，所有连接构成原型链。（稍后详解）

对象可以调用原型链上的所有属性以及函数。

console.log(Object.prototype. proto );



原型链的作用

使用原型链可以模拟类的功能。使用原型链可以实现继承关系。

原型对象(prototype)的创建

在预编译或执行匿名函数时，由js引擎调用new Function()创建函数对象，以及函数对象的prototype对象。只有函数对象才有prototype属性。

函数对象的本质就是一个变量，函数名是变量名，值为函数对象。原型链指针( proto ) 的特点

不可访问性：在IE等浏览器中**proto**属性不可访问。不可操作性：不能修改原型链指针的指向。



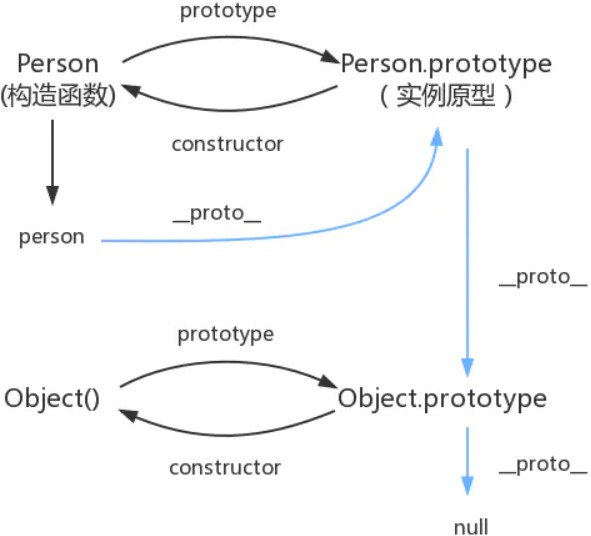
Function. proto ="abc"; console.log(Function. proto );

多元性：**所有的对象类型**（包含Function）都包含原型链指针。

连接性：对象的原型链指针默认指向构造函数的prototype对象，但prototype对象的

proto 也会作为原型链的一部分，直到连接到Object的prototype对象。

通用性：在原型链上所有prototype对象中的属性和函数，对象都可以访问和调用。



## O bject对象介绍

Object函数：对象函数，是由Function实现的，原型链的最终端，是所有对象类型的父类。Object

函数本质：Object就是用function实现的特殊构造函数。

创建对象方式：

字面方式



var obj={

name:"zhangsan", sayHello:function(){

console.log("我是张三");

}

}

//调用对象方法obj.sayHello();

构造函数方式使用new方式



var obj2 = new Object();

//可以直接给对象动态添加属性obj2.name="李四";

//可以直接给对象动态添加函数（匿名函数赋值给sayHello） obj2.sayHello=function(){

console.log("我是李四");

}

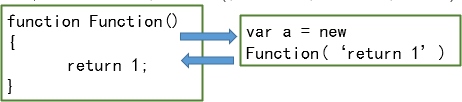
//调用对象方法obj2.sayHello();

Object原型函数：在Object对象的prototype对象里封装的函数，这些函数可以被任何对象调用。Obj.hasOwnProperty(‘field’)：判断某对象是否含有特定的自身属性。Obj2.isPrototypeOf（obj1）：判断一个对象是否存在于另一个对象的原型链上。obj.propertyIsEnumerable(‘field’)：判断一个对象的某一个属性是否是枚举类型的。obj.toLocaleString()： 将 一 个 对 象 转 换 为 本 地 字 符 串 。 obj.toString()：将一个对象转换为字符串。

obj.valueOf()：返回对象的值，一般由js引擎、Function、Object级别函数调用，请不要覆盖、调用。

console.log(obj2.toString()); console.log(obj.hasOwnProperty("name"));

## F unction对象介绍

Function函数：函数对象，任何一个函数的定义最终都是生成了一个Function对象，Function是自 省的（自己调用自己，创建自己的）。

函数对象与函数体的区别

函数对象是无序属性的集合，不调用函数也会经过预编译，加载到内存中。

函数体是程序块，在调用时，由函数对象执行apply方法进行执行，执行时才会加载到内存， 执行后会弹栈。

函数的全局属性与函数说明

arguments:参数数组，在函数调用后才会赋值，在函数体内也可以访问。 已经废弃

length：函数声明时的参数数量。apply(this,arguments):执行函数体内容。

this：在函数体的中this指针。arguments：传递给函数体的参数、数组。

call(this,paramter1，paramter2…):执行函数体内容。

this：在函数体的中this指针。paramter：传递给函数体的参数。



var me = {

name:" 张 三 ", sayHello:function(age){

console.log("大家好，我是"+this.name+"我今年"+age+"岁");

}

}

var someone={name:"某人"}; me.sayHello(24);

/\* sayHello中的this指向发生变化 \*/ me.sayHello.apply(someone,[30]); me.sayHello.call(someone,35);

## 对 象的创建模式

构造原型组合模式：使用构造函数和原型模式组合完成对象创建。构造函数完成基本属性的赋值， 原型完成对象函数的绑定。

function Person(name,age){ this.name=name; this.age=age;

}

/\* 原 型 对 象 设 置 sayHello 函 数 \*/ Person.prototype.sayHello=function(){

alert("你好，我是"+this.name);

}

var p = new Person("张三",20); p.sayHello();

## 寄 生组合继承模式

继承：继承是面向对象重要的特征。继承是指子类对象拥有父类对象的属性与方法，同时子类对象 可以扩展、覆盖父类对象的属性和方法。

混合继承方式：通过构造函数、原型链共同完成继承。使用最多的方案

/\* 定义Person构造函数 \*/ function Person(name,age){

this.name=name; this.age=age;

}

/\* Person的原型对象设置sayHello函数 \*/ Person.prototype.sayHello = function(){

alert("我是"+this.name+",大家好");

}

/\* 定义Student构造函数 \*/

function Student(name,age,schoolName){

this.schoolName = schoolName;

/\* 调用Person构造函数，使用call方法，改变this指向的对象，下面的代码，this指是Student

\*/

Person.call(this,name,age);

}

/\* 原 形 链 继 承 \*/ Student.prototype=new Person();

/\* 原 型 对 象 设 置 goSchool 函 数 \*/ Student.prototype.goSchool=function(){

alert("我去"+this.schoolName+"上学");

}

/\* 实例化Student对象 \*/

var s = new Student("张三",18,"清华");

/\* 调用方法 \*/ s.sayHello();

s.goSchool();

对象在访问属性或方法时，先检查自己的实例，如果存在就直接使用。如果不存在那么就去原 型对象上去找，存在就直接使用，如果没有就顺着原型链一直往上查找，找到即使用，找不到 就重复该过程直到原型链的顶端，如果还没有找到相应的属性或方法，就返回undefined，报错。

## 闭 包

闭包（ closure ）：闭包的目的是读取其他函数内部变量，由于正常函数执行后都会进行弹栈，从而不再保存临时变量，因此闭包会使函数的临时变量永久保存在内存中。

通过内部函数才能访问函数的变量，因此所有闭包的实现方式都是通过函数内部创建内部函数 的方式实现。

闭包会导致函数执行过程中内存不会释放，要注意使用场景与频繁程度。

function greet(sth){

/\* 匿名函数赋值给fun \*/

var fun = function(name){

/\* 这里使用了sth变量，就是闭包\*/ console.log(sth+","+name);

}

/\* 返 回 \*/ return fun;

}

/\* 调用 \*/

var g = greet("你好"); g("haha");

## 本 节总结

构造函数：函数的另外一种执行方式，执行后创建对象，并创建原型对象。

原型对象(prototype)：是函数对象（当且仅当函数对象）的一个内部属性，值是一个普通的对象，用户可以访问、操作该对象。

原型链( proto )：每个对象都具备的不可访问的内部属性（指针）（chrom等浏览器可以

访问，但无法操作）,指针指向构造函数对应的原型对象。

Function函数：函数对象。

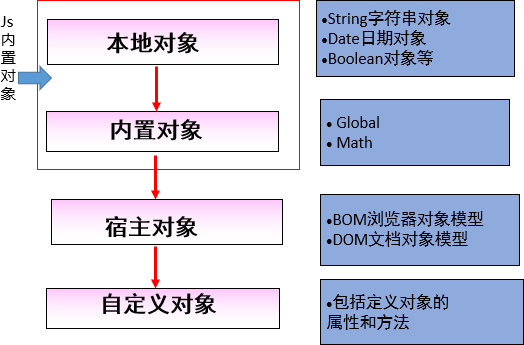
Object函数：所有创建对象的祖辈对象，也是由Function对象实现的。

构造原型组合模式：使用构造函数和原型模式组合完成对象创建。构造函数完成基本属性的赋 值，原型完成对象函数的绑定

混合继承方式：通过构造函数、原型链共同完成继承

# 本 地以及内置对象

## 对 象类型概述



宿主对象了解（之后再DOM、BOM章节详解）

宿主：寄居在主人家。Javascript需要插入或运行在浏览器内，因此可以称js宿主在浏览器提 供的环境内，浏览器是js的宿主环境。

宿主对象：由 ECMAScript 实现的宿主环境提供的对象。通俗的讲就是浏览器依据w3c制定的规范（DOM、htmlDOM）开发出来的对外接口，而这些接口是js对象（当然也要符合ECMAscript标准）

宿主对象分类（简单理解）

DOM：用于操作网页内元素的接口。BOM：用于操作浏览器窗口的接口。

Js内置对象：“独立于宿主环境的 ECMAScript 对象”。

js内置对象就是浏览器开发商依据标准（ECMA-262）使用原生javascript开发的对象

（function）

js内置对象与浏览器、网页上的元素无关。

js内置对象包含Object、Function、Array、String、Boolean、Number、Date、RexExp。 js内置对象在网页加载前就已经可以使用。

Js内置对象分类：

本地对象：javascript开发的引用类型，如String、Function、Boolean等等，这些引用类型都是function。

内置对象：已经被实例化的对象，如Math，内置对象都是Object。

Math对象：该对象用来执行常见的数学运算。Math对象在网页加载时已经由js引擎创建Math的实例对象。

获 取 Math 对 象 的 方 式 ： window.Math;//window可以省略Math对象的常用方法



console.log(Math.random()); console.log(Math.min(3,5));//3 console.log(Math.ceil(3.31));//4

## 基 本类型的封装类型

基本类型封装类型：js的内置对象，提供了对字符、数字、布尔三个基本类型的封装。封装类型中 的属性和函数可以协助完成对基本类型数据的操作，例如截取字符串等。

String：字符类型的封装类型。Number：数值类型的封装类型。Boolean：布尔类型的封装类型。

初始化方式：

var s = new String("aa");

String 类型常用api



var s = new String("abcdefg"); console.log(s.length);//7 console.log(s.charAt(2));//c console.log(s.concat("kkk")); //abcdefgkkk console.log(s.substring(2,4));//cd 左闭右开console.log(s.substr(2,5)); //cdefg console.log(s.toUpperCase());//ABCDEFG

## 数 组对象Array

Js数组与其他语言（例如java）的数组的相同点： 功能相同，都是存储介质。

操作相同，都是需要循环遍历数组的内容，同时都提供了修改数组数据的方式。

Js数组与其他语言（例如java）的数组的区别

Js数组对象是由js编写的本地对象，而其他语言中的数组都是独立的数据类型。Js数组的长度可变化。

Js数组存储的数据可以为不同数据类型。

Js数组由于是对象，因此可以为数组对象设定属性。数组的初始化

数组初始化后可以设定长度，若没有赋值，则数组中的元素为undefined 如果访问数组的索引号大于数组长度，则出现程序异常。

通过数组的length属性可以访问数组的实际长度。数组的内容可以为任意数据类型。

/\* 初始化数组长度为0 \*/ var array = new Array();

console.log(array.length);//0

//存入数据array.push("你好"); array.push(1); array.push(true);

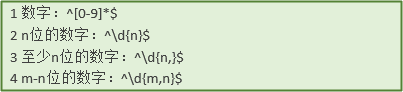
/\* 打印数组 \*/

console.log(array);// ["你好", 1, true] console.log(array[1]);//1

## 正 则表达式

正则表达式：正则表达式使用一种有规则的字符串来检验目标字符串是否匹配。

用于检查目标字符串串是否包含某个小字符串、或将匹配的子串做替换或提取子字符串的位 置。

正则表达式也是一串特殊的字符，转换为某种算法，根据这个算法来匹配文本。 正则表达式与js无关，但可以在js中调用。

正则表达式在web前端方向的使用场景：

数据验证：测试表单数据输入是否合法。

验证Email地址合法性

验证网址、电话号码、出生年月等输入信息的正确性。验证用户名、密码的注册信息安全性。

替换文本：使用正则表达式来识别字符串中的特定字符，并且删除或替换特定字符，

删除目标字符串中的空格、tab等。对特殊字符进行字符转换。

提取子字符串：使用正则表达式在目标字符串中提取子字符串，网址中获取域名。

## R egExp对象

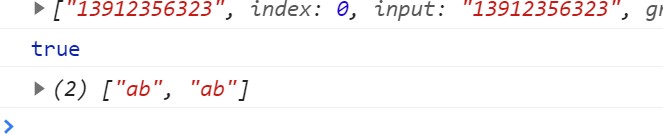
RegExp对象：用于执行或预编译正则表达式的对象。



flag说明：flag为字符类型，可以联合使用，取值如下。i：对大小写不敏感的匹配。

g:对全局字符进行匹配。m：对多行数据进行匹配。

RegExp对象的内置函数



var rex = /^1[3,5,7]\d{9}$/ var str = "13912356323";

console.log(rex.exec(str)); var rex2 = /ab/;

var str2 ="ab,abb,bb"; console.log(rex2.test(str2)); var rex3 = /ab/g;

/\* g表示全局匹配. 不会因为找到第一个匹配而返回 \*/

console.log(str2.match(rex3));

## 日 期对象Date

构造函数：日期的构造函数有四种

无参构造函数：返回的日期对象为当前系统时间。

参数为number类型的构造函数:number为格林威治时间及当前与1970年1月1日0点的毫秒 数。可以为负值。（时间戳）



参数为字符的构造函数



多参数构造函数：参数顺序为年、月（0-11）、日、时、分、秒、毫秒

静态方法：静态方法不需要实例化就可以直接使用日期来调用，Date有两个主要的静态方法。

Number Date.now（）:返回系统当前时间的格林威治时间，类型为number.

var d = Date.now(); console.log(d);//1621953082462

Number Date.parse(‘2020/12/01 16:07:32’)

参数：代表时间的字符串，格式为yyyy/mm/dd hh:mm:ss。返回参数所代表时间的格林威治时间，类型为number。

console.log(Date.parse('2020/12/01 16:07:32'));//1606810052000

实例方法：Date的实例对象有三类方法。

以set开头的方法是为实例对象设定时间，

以get开头的方法为获取当前实例对象的时间，以下列出常用的实例方法。

其他方法：包含将日期对象转换为字符串，以及将字符串转换为时间的方法。

get方法

set方法



其他方法

/\* 获取当前日期Date对象 \*/ var d1 = new Date();

console.log(d1.getDate());//25 console.log(d1.getMonth());//4 console.log(d1.getFullYear());//2021 console.log(d1.getYear());//121

/\* 设 置 年 \*/ d1.setFullYear("2020")

/\* 设 置 月 \*/ d1.setMonth("8")

console.log(d1.toDateString()); console.log(d1.toString());

## 本 节总结

js包含本地类型的对象和内置对象，本地对象是未实例化的对象，内置对象是已经实例化的对象。Math对象用来执行常见的数学运算。

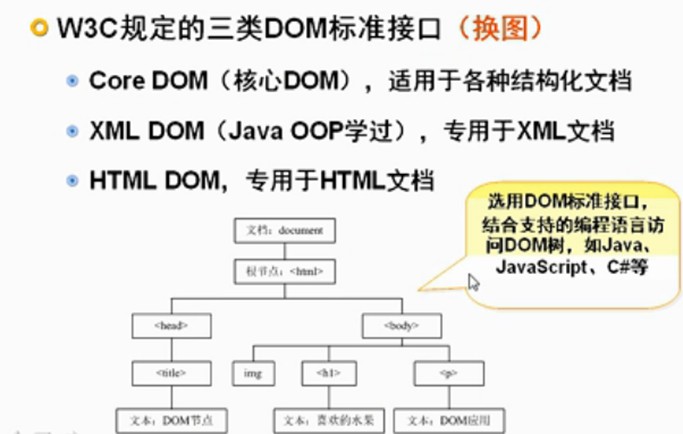
String数据类型为基本类型的封装类型，包含了字符截取字符转换等方法js的数组的初始化方式为var array2 = [1,2,'字符'];

正则表达式使用一种有规则的字符串来检验目标字符串是否匹配

Date对象提供用于处理日期和时间的属性及方法。

# D OM编程

## D OM标准

结构化文档解析的三个标准：

三个标准的意义以及关系

CORE DOM接口中只声明了一般结构化文档的访问方式，包含对文档元素的增删改查等接口，html与xml也都是结构化的文档。

XML DOM接口中声明了用于操作XML元素的方式，比CORE DOM接口声明的接口更明确。HTML DOM接口中声明了操作HTML元素的方式，CORE DOM接口只明确了一个类标签的操作方式，但HTML DOM确声明了每一个HTML标签的操作接口。

不同语言对三个标准的实现

接口的实现与标准无关，但是接口的调用方式与返回方式与语言类型相关。

语言一般只实现部分接口（例如js中实现了DOM接口和HTMLDOM接口用来操作文档） 在网页中的DOM、HTML DOM

遵照DOM接口、HTML DOM接口开发的操作网页（文档）元素的一组程序实现。接口都是由浏览器实现的，也只能在浏览器中运行。

浏览器在加载时网页时将每一个网页元素都封装成了一个javascript对象，这个对象封装了在

DOM、HTMLDOM规定的属性以及方法（只有IE有些区别）。DOM与HTMLDOM的区别

在DOM中每一个元素都是节点，都拥有相同的属性和方法。例如body、div。而HTML DOM

为不同类型的网页元素封装了不同的方法和属性。

DOM接口在浏览器中兼容，而HTML DOM确不完全兼容。与CSS类似，浏览器都扩充了私有的HTML DOM。

在网页开发中的DOM版本

DOM1：W3C组织确定网页开发的标准由DOM核心与DOM HTML两个模块组成。DOM2：DOM2通过对象接口增加了对鼠标和用户界面事件（DHTML长期支持鼠标与用户界面事件）、范围、遍历（重复执行DOM文档）和层叠样式表（CSS）的支持。 DOM3：包含一个名为“DOM载入与保存”的新模块，DOM核心扩展后可支持XML1.0的所有内 容，包括XML Infoset、 XPath、和XML Base

DOM扩展：在DOM3后（2008年后），部分浏览器增加了众多私有的DOM方法和属性，后被其他浏览器接纳。HTML5规范中也增加了对于DOM的新功能，后被浏览器接纳。不专属于DOM标准，但在高版本浏览器中兼容，称为DOM扩展

## D OM树与节点类型

DOM对象的加载过程：

首先封装文档对象document

在解析html网页元素时，当检测到开始标签时，封装DOM元素，并将DOM元素挂在父节点上。

解析到结束标签时DOM元素与该元素子元素封装完毕。

网页的全部元素都转换为DOM对象后，只需要操作DOM对象就可以改变文档结构。注意事项：

在html元素解析成DOM元素时，所有的回车、换行等都被封装为文本类型的DOM对象。使用DOM对象要保证DOM对象加载完成后。

DOM对象的类型：在CORE DOM的接口中规定了在结构化的文档中有12个DOM类型，但很多都是应用于XML的，应用在html文档中的有效对象类型如下：

文档类型：文档全局对象document，封装了检索、创建文档元素的属性和方法。

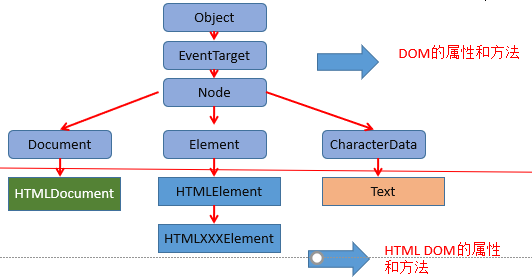
nodeType=9。

节点类型：也成为元素类型，封装了检索父子节点、操作当前节点的属性和方法，所有的

html元素都是节点类型。nodeType=1。

文本类型：节点类型下的文字内容对象。nodeType=3。

DOM对象的继承关系：浏览器都是通过继承关系来完成DOM元素的封装（不同浏览器有所不同），以下是chrome浏览器的继承关系。



## D ocument对象的属性及方法

本章节包含Document以及HTMLDocuemnt的兼容属性以及方法，按照功能可分为如下几类： 访问文档子节点

访问文档基本元素节点访问文档的特殊集合

创建元素、文本类型Dom对象

document访问HTML文档的基本元素

document.documentElement：获取HTML标签的dom对象。document.docType：获取HTML文档的文档头字符。document.body：获取body的dom对象。document.head：获取head标签的dom对象。document.title：获取title的文字标题。

/\* 获 取 title \*/ console.log(document.title);

/\* 获 取 head 对 象 \*/ console.log(document.head);

/\* 获 取 html 标 签 dom 对 象 \*/ console.log(document.documentElement);

document访问文档内子节点的方式

DomObj document.getElementById(‘id’):根据ID获取DOM对象，如果获取不到返回null。HTMLCollection document.getElementsByTagName(‘tagName’):根据标签名（忽略大小写）获取全部该标签的对象。

HTMLCollection document.getElementsByName(‘name’)

<body>

<div id="mydiv1">我是div1</div>

<p>我是p1</p>

<p>我是p2</p>

吃 ：<input type="checkbox" name="hobby" value="eat"/> 喝：<input type="checkbox" name="hobby" value="drink"/> 玩：<input type="checkbox" name="hobby" value="play"/> 乐：<input type="checkbox" name="hobby" value="happy"/>

</body>

<script type="text/javascript">

/\* 获取id为mydiv1的html元素DOM对象 \*/

var myDiv1 = document.getElementById("mydiv1"); console.log(myDiv1);

/\* 获取DOM对象的 innerText和 innerHTML 属性\*/

console.log(myDiv1.innerText); console.log(myDiv1.innerHTML);

/\* 根据html标签名称获取dom对象集合 \*/

var domP = document.getElementsByTagName("p"); console.log(domP);

for(var i=0;i<domP.length;i++){ console.log(domP[i].innerText)

}

/\* 根据name获取所有的dom对象集合 \*/

var checks = document.getElementsByName("hobby"); for(var i=0;i<checks.length;i++){

/\* check标签选中 \*/

checks[i].checked=true;

}

</script>

document对象封装了常用的网页元素集合，都是HTMLCollection类型的，都属于HTML DOM标准的实现，但在浏览器中完全兼容。

images： 全 部 图 片 DOM 对 象 集 合 。 anchors：全部带name属性的A标签DOM对象集合。links：全部带href属性的A标签DOM对象集合。forms： 全 部 表 单 DOM 对 象 集 合 。 Script：全部javascript标签DOM对象。

<img src="" alt="图片1"/>

<img src="" alt="图片2"/>

<a href="#">超链接1</a>

<a>超链接2</a>

<script type="text/javascript">

/\* 获取所有的img标签DOM对象 \*/ console.log(document.images);

/\* 获取所有的带有href属性的a标签的DOM对象 \*/ console.log(document.links);

</script>

document封装了用于创建元素类型、文本类型DOM对象的方法。document.write(‘html’):根据参数内容重新更新文档内的内容。

document.createElement(‘tagname’)：根据标签名创建元素类型DOM对象，此DOM对象处

于 游 离 状 态 ， 没 有 挂 接 在 DOM 数 上 ， 无 法 显 示 。 document.createTextNode(‘content’)：根据内容创建文本类型DOM对象，此DOM对象处于 游离状态，没有挂接在DOM数上，无法显示。

/\* 创建input标签，在内存中，没有在页面中显示 \*/ var input = document.createElement("input"); document.write("这是最后一行");

/\* 把input的DOM对象加入到body元素下 \*/

document.body.appendChild(input);

## E lement对象的属性及方法

Element对象封装了节点类型DOM对象特有的属性以及方法 对元素属性的操作

对元素样式的操作

对元素及子元素的操作

Element对象封装了节点类型DOM对象特有的属性以及方法

获取属性的方式：obj.getAttribute(‘attName’)：获取当前节点指定属性的值。

设定属性的方式：obj.setAttribute(‘attName’,’attValue’):设置当前节点指定属性的值。

* 删除属性的方式：obj.removeAttribute(‘attName’):删除当前节点指定属性的值。

<a href=["http://www.baidu.com](http://www.baidu.com/)" id="myid1">百度</a>

<script type="text/javascript">

/\* 根据id获取元素的DOM对象 \*/

var element = document.getElementById("myid1");

/\* 获 取 a 标 签 的 href 属 性 \*/ console.log(element.getAttribute("href"));

/\* 设 置 a 标 签 的 href 属 性 \*/ element.setAttribute("href",["http://www.163.com](http://www.163.com/)");

/\* 删 除 属 性 \*/ element.removeAttribute("href");

</script>

HTMLElement对象封装了设定元素样式的方法，属于HTMLDOM范畴，但浏览器完全兼容 设定或获取CSS属性的方式

obj.style.color = ‘red’：设置或获取当前元素的css属性值

obj.style.fontSize= ‘10px’：设置或获取当前元素的css属性值，cssAttribute取消了-，采用换字母大写的方式（驼峰写法）。

<a href=["http://www.baidu.com](http://www.baidu.com/)" id="myid1" style="color:blue;font-size:50px">

百度</a>

<script type="text/javascript">

/\* 根据id获取元素的DOM对象 \*/

var element = document.getElementById("myid1");

/\* 设 置 红 色 和 字 体 打 下 \*/ element.style.color="red"; element.style.fontSize="5px";

</script>

Element对象封装了增加修改删除当前元素的方式： 增加元素到文档树的方式

parentObj.appendChild(obj):将指定元素挂接到父元素下。brotherObj.insertBefore(obj):将指定元素挂接到某个兄弟元素之前。

修改元素的方式

parent.replaceChild(newObj,oldObj):替换指定元素的某一个子节点。

删除元素的方式

parent.removeChild(obj):删除指定元素的某一个子元素。

<div id="mydiv1">

<p id="myp1">我是p标签</p>

</div>

<script type="text/javascript">

/\* 获取div的dom对象 \*/

var domDiv = document.getElementById("mydiv1");

/\* 获取p的dom对象 \*/

var domP = document.getElementById("myp1");

/\* 创建span标签的dom对象\*/

var spanElement = document.createElement("span")

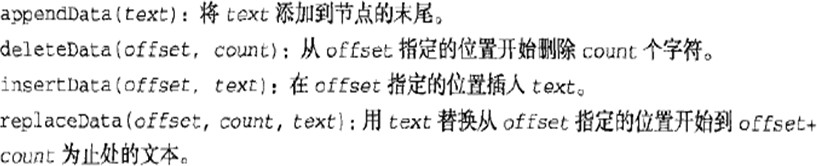
/\* span 设 置 Text 值 \*/ spanElement.innerText = "我是span";

/\* 替 换 \*/ domDiv.replaceChild(spanElement,domP);

</script>

## T ext对象的属性及方法

Text对象封装了文本类型DOM对象特有的属性以及方法： 对象属性：data：设定或获取文本内容。

对象方法：

<div id="mydiv1"></div>

<script type="text/javascript">

/\* 获取div的dom对象 \*/

var domDiv = document.getElementById("mydiv1");

//创建文本节点

var textNode =document.createTextNode("我搜");

//文本节点天骄到div中domDiv.appendChild(textNode);

//文本节点追加内容textNode.appendData("我是后续的");

</script>

## H TMLDOM简介

HTMLDOM在DOM的基础上扩展了每一个专有标签的方法和属性Html属性：

innerHTML：设置或获取当前元素内的html。innerText：设置或获取当前元素内的文字内容。outerHTML：获取元素的外部html。

获取大小的属性

offsetHeight/offsetWidth：获取用户可见的当前对象的高度和宽度，包含边线。

scrollHeight/scrollWidth:获取当前对象的实际宽度和高度，包含需要滚动的部分。位置属性：

offsetParent：获取当前元素距离最近的使用相对定位获取绝对定位的祖元素。offsetLeft/offsetTop:获取当前元素的左上角与其offsetParent的左上角的水平/垂直方向的偏移量。

### 由于HTMLDOM的API过于繁杂，建议其余模块查阅API文档进行开发，开发时注意兼容性。

**本 节总结**

CORE DOM接口中只声明了一般结构化文档的访问方式，HTML DOM接口中声明了操作HTML元素的方式

在网页开发中常用的节点类型为：文档类型、节点类型、文本类型。

document对象包含访问文档子节点访问文档基本元素节点的api接口、访问文档内子节点的api接口、访问文档的特殊集合的api接口、创建元素、文本类型Dom对象

Element对象封装了节点类型DOM对象特有的属性以及方法，主要包含对元素属性的操作的api接口、对元素样式的操作的api接口、对元素及子元素的操作的api接口