第一天 - JS 入门

# 一 Javascript 简介

JavaScript 是运行在客户端的脚本语言，是动态类型、弱类型、基于原型的语言，内置支持类型！

JavaScript 被数百万计的网页用来改进设计、验证表单、检测浏览器、创建 cookies，以及更多的应用。

JavaScript 是因特网上最流行的脚本语言。

JavaScript 很容易使用。

JavaScript 是一种轻量级的编程语言。

Javascript 文件的扩展名是.js。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由浏览器执行。

JavaScript 存在于全世界所有 Web 浏览器中，能够增强用户与 Web 站点和 Web 应用程序之间的交互。

Javascript 用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

# 二 Javascript 推荐书籍

《Javascript 高级程序设计》 图灵出版社

# 三 Javascript 简单介绍？

1. javascript的发展史
2. Javascript用来给HTML网页添加动态功能，如淘宝、京东图片轮播...
3. Javascript的生态环境

# 四 Javascript的标签位置、以及如何使用

1. HTML 页面使用 Javascript 的几种方式

对比 HTML 页面使用 CSS 的方式，页面中使用 Javascript 有内部 Javascript、外部 Javascript、内联（行内）Javascript 三种方式。

* 1. 内部 Javascript

// 方式一

<head>

<script> alert(‘Hello’);

</script>

</head>

// 方式二

<body>

<script> alert(‘Hello’);

</script>

</body>

* 1. 外部 Javascript（推荐使用）

// 方式一

<head>

<script src=”path/to/file.js”></script>

<script src=”path/to/file2.js”></script>

</head>

// 方式二

<body>

<script src=”path/to/file.js”></script>

<script src=”path/to/file2.js”></script>

</body>

* 1. 内联（行内）Javascript

// 方式一

<button onclick=”alert(‘我被点击了’)”>按钮</button>

// 方式二

<button onclick=”hello()”>按钮</button>

<head>

<script>

**function hello**() {

alert(‘我被点击了’);

}

</script>

</head>

// 或者

<body>

<script>

**function hello**() {

alert(‘我被点击了’);

}

</script>

</body>

推荐把外部 Javascript 文件的引入位置放在 body 结束标签的前面以得到最好的用户体验。外部脚本内不能包含 <script> 标签。

# 五 Javascript的三大系统弹出框方法

* 1. 警告提示弹出框：alert()

<script>

// 方式一

alert(“人生就像一杯茶，会苦一阵子，但不会苦一辈子！”);

// 方式二

Window.alert(“人生就像一杯茶，会苦一阵子，但不会苦一辈子！”);

</script>

* 1. 确认取消弹出框：confirm(param)

param是该方法中传入的参数，如“你今天吃饭了吗？”

该方法的两个返回值：true 或者 false

<script>

// 方式一

confirm(“你今天吃饭了吗？”)

// 方式二

window.confirm(“你今天吃饭了吗？”);

// 使用提示弹出框查看该方法的返回值

alert( confirm(“你今天吃饭了吗？”) )

</script>

* 1. 输入弹出框：prompt(param1,param2)

param1是输入弹出框的提示内容，param2是文本框默认值

该方法的返回值：文本框内容或者null

<script>

// 方式一

prompt(“请输入您的成绩：”,”成绩范围0 - 100之间的数字”);

// 方式二

window.prompt(“请输入您的成绩：”,”成绩范围0 - 100之间的数字”);

// 使用提示弹出框查看该方法的返回值

alert( confirm(prompt(“请输入您的成绩：”,”成绩范围0 - 100之间的数字”) )

</script>

# 六 Javascript的页面输出

向文档中写入内容：document.write(写入的内容)

<script>

document.write(“人生可以无中生有的是梦想，可以不劳而获的是贫穷”);

// 动态输出HTML，浏览器会自动的解析标签

document.write(“<h1>这是一个大标题</h1>”);

// document.write会重绘整个文档：如果在文档已完成加载后执行，则整个HTML网页将被覆盖

</script>

# 七 Javascript的控制台输出：错误调试以及查看对象中的属性

方法：console.log()，可以向控制台中输出一个或多个值，如果程序在编译不通过时，会在控制台中抛出相对应的错误。

<script>

console.log(“若是美好，叫做精彩；若是糟糕，那是经历！”);

console.log(“Hello,”,”world!”);

</script>

# 八 javascript严格区分大小写以及代码的书写规范

Javascript 对大小写敏感（区分大小写）

Javascript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性。您可以在文本字符串中使用反斜杠对代码行进行换行。

*// 可 以*document.write("Hello \ World!");

*// 不可以*document.write \ ("Hello World!");

Javascript 是脚本语言。浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译

# 九 javascript注释：单行注释// 多行注释/\*\*/

Javascript 不会执行注释，可以用来阻止代码执行。

我们可以添加注释来对 Javascript 进行解释，或者提高代码的可读性。单行注释以 // 开头；多行注释以 /\* 开头，以 \*/ 结束。

// alert(‘hello’);

/**\*** alert(‘a’);

abler(‘b’);

**\***/

alert(‘your age is 22’); // 提示用户

/**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* author: situ

**\*** since: 2012-09-21

* 测试例子

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/ function test () {

}

# 十 文档加载以及预加载

文档的加载时自上而下的，javascript属于脚本语言，所以在整个加载的过程中，javascript程序的执行也属于其中一部分，如果javascript程序在编译执行的过程中出现错误，不会阻断文档结构的加载，但会阻断程序的执行。在整个的加载过程中，如果出现了弹出框，则会阻断整个文档的加载。

预加载事件：window.onload = function(){}

<script>

window.onload = function(){

// 文档在加载完毕之后，执行的javascript程序

console.log(“我是在文档加载完毕之后，输出的...”);

}

</script>

# 十一 简单介绍window对象

每一个在浏览器中打开的文档，浏览器都会动态的给当前的文档创建一个window对象，该window对象表示浏览器中打开的窗口，当浏览器中存在多个网页时，每个网页的window之间是互相不影响的。window对象是javascript中最外围的一个对象，也是javascript里最重要的一个对象，是一个全局对象，所编写的程序会作为window对象的属性或者说方法而存在。

# 学习心得：

1. - JS 变量

# 一 什么是常量和变量？

* 1. 量是容器，存储数据的。变量是可变的容器，常量是不可变的容器。
  2. 变量声明的关键字：var ==> variable
  3. 变量声明时可以不赋值：变量的值是undefined
  4. 变量声明赋值：var 变量名称（标识符） =（等号赋值）所赋的值 ;（分号结束语）

*// js 定义变量的语句*

**var** x = 2;

**var** y = 3;

**var** z = x + y;

*// 代数的表达方式，在 JavaScript 中，这些字母被称为变量。*

设 x = 2

设 y = 3

设 z = x + y

求 z 的值?

* 1. 一次声明多个变量，使用逗号隔开，可以赋值，也可以不赋值
  2. 标识符：常量名称、变量名称、参数名称、函数名称、属性名称

*//* 一次声明多个变量

var name = “张三”,age = 18,sex = “男”,salary = null,marry;

*//* 标识符的命名规则

*/\**

1. 不能以数字开头
2. 不能出现除$和下划线之外的其它符号
3. 关键字不能作为标识符使用，保留字不建议作为标识符使用（建议：最好不要使用保留字）

\*/

* 1. 关键字和保留字

*//* 关键字:

*/\**

break do instanceof typeof case else new var finally

return void continue for switch while function this with

default if throw delete

\*/

还有一些目前在js中没有任何特殊用途，但将来很有可能作为关键字的保留字

*//* 保留字:

*/\**

abstract enum int short boolean export interface staic byte

extends long super char final native sunchronized class float

public package throws const goto private transient debugger inplements

protected volatile double import

\*/

# 二 Javascript 语句

Javascript 语句

Javascript 语句向浏览器发出的命令。语句的作用是告诉浏览器该做什么。

Javascript 语句使用分号作为一条语句的结束符号。

通常我们在每条可执行的语句结尾添加分号。（分号可选，要求必须使用） 使用分号的另一用处是在一行中编写多条语句。

# 三 Javascript 代码

Javascript 代码（或 Javascript）是 Javascript 语句的序列。浏览器会按照编写顺序来执行每条语句。

# 四 Javascript 代码块

JavaScript 语句通过代码块的形式进行组合。块由左花括号开始，由右花括号结束。

块的作用是使语句序列一起执行。

JavaScript 函数是将语句组合在块中的典型例子。

*// 一个简单的函数例子*

**function hello**() { alert(‘Hello World’);

}

alert(‘your age is 22’); // 提示用户

/**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* author: E动国际

**\*** since: 2018-03-30

* 测试例子

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***/ function test () {

}

**五 Javascript 数据类型简介**

JavaScript 变量还能保存其他数据类型，比如文本值 (name="Bill Gates")。

在 JavaScript 中，类似 "Bill Gates" 这样一条文本被称为字符串。

JavaScript 变量有很多种类型，但是现在，我们只关注数字和字符串。当您向变量分配文本值时，应该用双引号或单引号包围这个值。

当您向变量赋的值是数值时，不要使用引号。如果您用引号包围数值，该值会被作为文本来 处理。

**六 Javascript 数据类型**

简单数据类型：字符串、数字、布尔、null、undefined

复杂数据类型：对象（js对象、数组...）

javascript是弱类型语言，所以其类型是松散类型的，也就是说，变量本身是不存在类型的，变量的类型与其所赋的值是什么类型有关系

* 1. Javascript字符串：字符串是存储字符（如”Bill Gates”）的变量

字符串也可以是引号中的任意文本。凡是本单引号或者双引号引起来的内容都是字符串。

在一个字符串中，双引号可以和单引号混合使用，但一定要注意引号相互之间的匹配。

* 1. Javascript数字：整数、浮点数、NaN、Infinity、-Infinity

NaN：在计算无结果时，其运算结果就是NaN（not a number）

“a” \* “b” // NaN

undefined + null // NaN

......

Infinity：正无穷

1/0 // Infitity

-Infinity：负无穷

* 1. Javascript布尔类型：

布尔逻辑只有两个值：true 或者false

true==>1,false==>0

var b1 = true;

Var b2 = false;

* 1. javascript两个特殊值：null和undefined

都表示变量不含值

null是一个特殊的空对象，表示指向对象的空指针（不指向任何对象，在内存中不占位）。通常用于变量声明赋值和手动清除垃圾。

undefined是”undefined”类型的：

/\*

1. 变量声明未赋值
2. 对象下不存在该属性
3. 函数没有返回结果
4. 字符串和数组中不存在下标对应的字符或者元素时

\*/

var a; // undefined

var obj = {}

console.log(obj.name); // undefined

function fn(){}

fn(); // undefined

var str = “abcd”;

console.log(str[4]);

var arr = [];

console.log(arr[0]);

* 1. javascript对象：object

对象的创建：

// 方式一

var obj = {};

// 方式二

var obj = new Object()

对象属性的动态添加：

obj.name = “张三”;

obj[“name”] = “张三”;

对象属性的读取：

cosnole.log(obj.name);

cosnole.log(obj[“name”]); // 张三

对象属性值得修改：

obj.name = “李四”;

obj[“name”] = “李四”;

对象可操作属性的删除：

delete obj.name;

**七 Javascript 数据类型的检测**

两种语法结构：typeof操作符 typeof()方法，使用方式都是相同的

/\*

检测结果6种：number、string、boolean、object、undefined、function，检测结果都是字符串类型的

\*/

typeof 1; // “number”

typeof “abc”; // “string”

typeof true; // “boolean”

typeof null; // “object”

typeof undefined; // “undefined”

typeof alert; // “function”

**八 Javascript 数据类型转换**

* 1. 各种数据类型转换为字符串类型

对象下的方法：\*.toString()

\*是一个对象或者可以动态的转换为对象的数据

var num = 10;

num.toString(); // “10”

var n = null;

n.toString(); // 报错

var u = undefined;

u.toString(); // 报错

强制转换：String()

String(123); // “123”

String(true); // “true”

String(null); // “null”

String(undefined); // “undefined”

* 1. 各种数据类型转换为数字类型

强制转换：Number()

能把各种数据转换为数字类型的，如果数据对应的没有数字或者数据本身不是数字，则转换为NaN；所有数据中使用Numebr强转函数，特殊的有：””,” ”,null

Number(“”); // 0

Numebr(null); // 0

Number(“ ”); // 0

转换为整型：parseInt()

转换为浮点型：parseFloat()

// 1、数字能转换为数字

// 2、字符串类型的数字能转换为数字

// 3、以数字开头的字符串能转换为数字

// 4、除去以上3点包含的数据，其他的都转换为NaN

* 1. 各种数据类型转换为布尔类型

// 强转函数：Boolean(待转换的数据)

// 逻辑操作符：逻辑非 ! 取反，然后转换为布尔类型

// 所有数据中对应的布尔值为false的有：0、-0、false、""、null、undefined、NaN

// 所有的对象都是真值

// 所有的函数或者方法也都是真值

* 1. 隐式类型转换

// 隐式类型转换：

// if(判断条件){

// 代码块

// }

// if({}){

// alert(1);

// }

// 1 && 0 && 3

// if(1 && 0 && 3){

// }

// 1 - "2"

// 1 + “2”

// 2 \* “3”

// .....

1. Javascript运算符
2. **按条目划分**
   1. 一元运算符

只能处理一个操作数的运算符叫做一元运算符

数学的正负：+ -

有自动转型的功能，会把各种数据类型转换为数字类型

console.log(+true,+undefined,+”a”);

逻辑运算符：逻辑非 !

会把各种数据转换为对应的反的布尔值

console.log(!”a”,!true,!0);

自增自减：++ --

++ -- 由于进行自增和自减，必须配合变量使用

const i = 1;

i++ || ++i; // 错误的

++1 || 1++; // 错误的

++在前，先加后取

var i = 1;

console.log(++i,i); // 2 2

++在后，先取后加

var i = 1;

console.log(i++,i); // 1 2

typeof 操作符也是一元运算符

由于能处理一个操作数的运算符有很多，不能逐一列举。

* 1. 二元运算符

处理两个操作数的运算符叫做二元运算符，简单的举例，不再一一列举

逻辑运算符与或：&& ||

算术运算符：+ - \* / %

赋值运算符：= += -= \*= /= %= <<= >>= &= |=

位移运算符：<< >> & |

......

* 1. 三元运算符

三元运算符又叫条件运算符，会有一个运算结果，条件成立时，运算结果为第一个，条件不成立，运算结果为第二个

语法：判断条件 ? 结果1 : 结果2;

// 例子

var score = 90;

var info = score < 60 ? “不及格” : “及格”;

1. **按功能划分**
   1. 算术运算符：+ - \* / %

在javascript里，+有三个功能，字符串的拼接、数学运算相加以及数学里的正号

console.log(1 + 1,1 + undefined，1 + “1”);

求余：%

1. 任何数对0求余的结果是NaN
2. 任何整数对1求余的结果是0
3. 任何小数对1求余的结果是小数部分
4. 求余数小于被求余数，余数是求余数本身
5. 求余数大于被求余数，余数是除不尽剩下的那部分数字

console.log(1%0,10%1,1.123%1,5%10,9%5);

* 1. 赋值运算符：= += -= \*= /= %=

赋值的运算顺序是从右向左的（赋值运算符的优先级是比较低的）：

var num = 10 + 10;

+= // x += 3 等价于 x = x + 3;

问题:

num %= (num%=(num+=9)); // ?num

* 1. 比较运算符：> < >= <= == === != !==

比较运算符的运算结果都是布尔类型的：true || false

程序的执行顺序是从左向右的，先执行7>6,结果为true，然后再比较true>5

console.log(7>6>5); // false

> < >= <=：

在比较时，如果两边的操作数不是字符串类型的，进行比较时，会隐式的转换为数字，然后再进行比较；如果两边的操作数都是字符串，则比较的是unicode编码值。

console.log(true > 0,”a” > 1,”b” > “a”); // true false true

==：进行值的比较

1. null除了和undefined以及它本身之外的数据进行比较时，结果都是false
2. Undefined除了和null以及它本身之外的数据进行比较时，结果都是false
3. 如果两边的操作数是不同等类型的数据时，先使用Number强转函数转换，然后再进行比较；如果两边的操作数是同等类型时，则直接进行值的比较。
4. NaN和任何数据的比较结果都是false，包括它本身
5. 引用类型之间进行比较时，比较的是地址指针

console.log(null == undefined,null == 0,”” == 0,”” == “0”,NaN ==NaN,{} == {});

// true false true false false false

===：值和类型都相同时，结果为true

console.log(3 === “3”,3 === 3,3 === 4); // false true false

!=：值不相同，结果为true

console.log(3 != “3”); // false

!==：值或类型不相同，结果为true

console.log(3 !== “3”); // true

* 1. 逻辑运算符：逻辑与&& 逻辑或|| 逻辑非!

逻辑运算符用于测定变量或值之间的逻辑。

逻辑与或都是短路操作符，当运算找到结果时，则不会再继续向下运算。

逻辑与是找假值的，一旦找到假值，则不会再继续向下运算，整个逻辑表达式的运算结果就是对应的那个假值；如果整个逻辑表达式运算到最后，都没有找到假值，则运算结果就是最后一个操作数。

// 例子

console.log(window && true && null && “a” && alert); // null

console.log(window && true && “false” && “a” && alert); // alert

逻辑或是和逻辑与对应的，找真值。一旦找到真值，则不会再继续向下运算，整个逻辑表达式的运算结果就是对应的那个真值；如果整个逻辑表达式运算到最后，都没有找到真值，则运算结果就是最后一个操作数。

console.log(null || undefined || “” || true || 0 || NaN || false); // true

console.log(null || undefined || “” || -0 || 0 || NaN || false); // false

逻辑非是一元运算符：对数据对应的布尔值取反

console.log(!”a”,Boolean(“a”)); // false true

* 1. 字符串运算符：+ +=

字符串运算符的主要功能就是实现字符串之间的拼接或者合并

var str1 = “ab”;

var str2 = “cd”;

console.log(str1 + str2,str1 += str2); // abcd abcd

**第四天** 流程控制语句

流程控制语句用于给予不同的条件来执行不同的动作。

在javascript中，我们可以使用以下流程控制语句：

if语句 和 switch语句

* 1. if语句按照分支结构划分：单分支、双分支以及多分支结构
  2. If语句省略花括号时，有且仅有一条语句属于其相对应的代码块

// 以下程序会弹出2

if(false)

alert(1);

alert(2);

* 1. 单分支结构：只有当条件为true时，才会执行其相对应的代码块

// 语法

if(判断条件){

// 条件成立时，执行的代码块

}

// 例子

var weather = “rainy”;

if(weather == “rainy”){

alert(“出门带伞！”);

}

* 1. 双分支结构：当指定条件为true时，执行if对应的代码块；条件为false时，执行else对应的代码块

// 语法

If(判断条件){

// 判断条件成立时，执行的代码块

}else{

// 判断条件不成立时，执行的代码块

}

// 例子

var age = 17;

if(age < 18){

Info = “未成年”;

}else{

Info = “成年人”;

}

* 1. 多分支结构：存在多个判断条件，但只会执行第一个成立条件对应的代码块。

// 语法

If(条件1){

代码块1

}else if(条件2){

代码块2

}else{

代码块3

}

// 例子

var num = 0;

if(num < 0){

Info = “负数”;

}else if(num > 0){

Info = “正数”;

}else{

Info = “零”;

}

* 1. switch语句：当分支结构超过3个时，不能再使用多分支结构，使用switch语句

用于给予不同的条件来执行不同的代码块

使用default关键词来规定匹配不存在时做的事情

switch中出现的关键词：switch、case、break、default

// 语法

/\*

switch(带判断的值){

case 判断条件1:

代码块1

break;

case 判断条件2:

代码块2

break;

case 判断条件3:

代码块3

break;

case 判断条件4:

代码块4

break;

default:

默认代码块；

}

\*/

\*\*\*注意：

带判断的值和判断条件执行的结果进行全等比较

break漏写，会导致判断条件成立之后的所有的代码块的执行

// 例子

var week = 1;

switch(week){

case 0:

Info = “星期天”;

break;

case 1:

Info = “星期一”;

break;

case 2:

Info = “星期二”;

break;

case 3:

Info = “星期三”;

break;

default:

Info = “请输入一个合法的数字...”;

}

**第五天** 循环语句

**一 循环的概念：**

在一定的条件内，重复不断的执行某件事情。

**二 Javascript中，循环语句包括以下几种**

for - 是一种先判断后运行的循环语句，在循环次数固定时使用

while - 是一种先判断后运行的循环语句，在循环次数不固定时使用

do...while... - 是一种先运行后判断的循环语句，循环至少执行一次

for...in...循环遍历对象的属性（暂略）

**三 for循环语句**

// 语法

for(初始值;判断条件;步长更新){

循环体

}

循环的执行：

1. 初始值 2、判断条件 3、步长更新 4.循环体

for(1;2;3){4}

循环的执行顺序：

1==>2==>4==>

3==>2==>4==>

3==>2==>4==>

...

2==>4（循环结束）

初始值值执行一次

判断条件比循环体执行的次数多1

步长更新和循环体执行的次数相同

// 例子

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

console.log(i);

}

\*\*\*注意：程序的执行是自上而下的，因此，循环在未执行结束前，不会继续向下执行

// 死循环

1. 空语句

for(;;){}

1. 判断条件始终成立

for(var i=0;true;i++){}

1. 判断条件是大于或者大于等于，步长更新++或者+=

for(var i=10;i>=0;i+=1){}

1. 判断条件是小于或者小于等于，步长更新--或者-=

for(var i=0;i<=10;i-=1){}

以上程序都不会再向下执行，网页一直处于加载状态。

**三 while循环**

while循环在循环条件成立时，执行循环体

// 语法

while(判断条件){

条件成立时，执行循环体

}

// 例子

var i = 0;

while(i <= 10){

console.log(i);

i++;

}

注意：如果忘记增加条件中所用变量的值，该循环永远不会结束，能导致浏览器崩溃。

**四 do...while...循环**

do...while 循环是while循环的变体。该循环会在执行判断条件之前，提前执行一次，然后再进行条件判断，如果成立，继续执行循环

// 语法

do{

// 循环体

}while(判断条件)

// 例子

var i = 0;

do{

console.log(i);

i++;

}while(i < 10)

注意：如果忘记增加条件中所用变量的值，该循环永远不会结束，能导致浏览器崩溃。

**五 跳转语句：break和continue语句，配合循环使用**

break 语句用于跳出循环（中断当前循环）

continue 语句用于跳出当前循环，继续下一次循环

break 语句跳出循环后，则不会再继续执行该循环断开之后的代码。

continue 语句跳出当前循环后，则会继续向下执行当前循环的下一次循环

// break语句

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

if(i == 5) break;

console.log(i);

}

// 0 1 2 3 4

// continue语句

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

if(i == 5) continue;

console.log(i);

}

// 0 1 2 3 4 6 7 8 9

**六 javascript错误处理**

try 语句测试代码块的错误

catch 语句处理错误

throw 语句创建自定义错误

* 1. 错误一定会发生

当javascript引擎执行javascript代码时，会发生各种错误；可能是语法错误，通常是程序员造成的编码错误或错别字。可能是拼写错误或语言中缺少的功能（可能由于浏览器差异）。可能是由于来自服务器或用户的错误输出而导致的错误。当然，也可能是由于喜多其它不可预知的因素。

* 1. javascript抛出错误

当错误发生时，当事情出问题时，javascript引擎通常会停止，并生成一个错误消息，描述这种情况的技术语：javascript讲抛出一个错误。

* 1. javascript测试和捕捉

try语句允许我们定义在执行时进行错误测试的代码块.

catch语句允许我们定义当try代码块发生错误时，所执行的代码块。

javascript语句try和catch是成对出现。

// 语法

try{

// 执行的代码块

}catch(error){

// 被执行代码有错误在这里处理

}

// 例子

try{

alert(“hello”);

}catch(error){

alert(error.message); // 没做任何处理，知识显示错误信息

}

* 1. throw语句

throw 语句语句允许我们创建自定义错误

正确的技术术语是：创建或抛出异常（exception）。

如果把throw与try和catch一起使用，那么您能控制程序流，并生成自定义的错误消息。

// 语法

throw 异常信息（异常信息可以是javascript字符串、数字、逻辑值或对象）

1. javascript入门

**一、函数入门**

1. 什么是函数？

函数是由事件驱动的，或者被调用执行时，可重复使用的代码块

1. 函数定义的两种方式？
2. 函数声明：关键字function

// 语法

function 函数名(){

// 代码块 || 函数体

}

1. 函数表达式

// 方式一

var 函数名 = function (){

// 代码块 || 函数体

}

// 方式二

Var 函数名 = new Function(参数字符串,函数体字符串);

1. 函数的调用

// 语法

函数名();

// 函数声明和函数表达式在调用上的区别：

函数声明可以先调用再声明

函数表达式必须先声明再调用

此块知识内容牵涉到存储方面的知识，再后面的章节中会做出讲解

1. 函数的参数

参数分为两种：形式参数、实际参数

1. 形式参数出现在函数封装时
2. 实际参数出现在函数调用时
3. 形式参数只能来自于实际参数，实际参数的值不能来自于实际参数
4. 形式参数的个数没必要和实际参数的值一一对应
5. 所有的形式参数都是局部变量

function 函数名(参数1,参数2,参数3){

}

函数名(1,2,3,4,5)

1. 函数的返回值

// 语法

function 函数名(){

return 函数的返回值

}

返回值：

1. 函数没有返回值时，函数的执行结果是undefined
2. 一个函数中可以出现多个return，但只会执行一个return语句
3. return可以返回任何数据类型，包括表达式
4. return有中断函数体的功能
5. 函数的重载

函数重复声明时，只会存在一个函数，另外一个会被覆盖掉

function fn(){

fonsole.log(1);

}

fn(); // 2

function fn(){

fonsole.log(2);

}

fn(); // 2

1. 函数的封装

函数封装的顺序：

原材料==》进厂==》加工==》出厂

原材料：实参

进厂：形参

加工：函数体

出厂：函数返回值

// 例子：简单的数学运算

function oprate(num1,num2,o){

return eval(num1 + o + num2);

}

oprate(2,3,”\*”);

**二、对象入门**

对象是带有属性和方法的特殊数据类型，属于复杂数据类型。

Javascript中的所有事物都是对象：字符串、数值、数组、函数.....一切事物皆对象

Javascript允许自定义对象

Javascript提供多个内建对象。比如String、Date、Array...等

创建自定义javascript对象

// 创建空对象

var obj = new Object();

var obj = new Object; // 不推荐

var obj = {}; // 对象直接量方式（推荐）

// 给对象添加属性

obj.type = “笔记本电脑”;

obj.name = “联想”;

obj.price = “6500”;

var obj = {

type: “笔记本电脑”,

name: “联想”,

price: 6500

};

// 访问对象的属性：

var type = obj.type;

var name = obj[“name”];

// 给对象增加方法：

obj.getName = function(){

return this.name; // obj.name

}

var obj = {

getName: function(){

return this,name;

}

}

// 修改对象的属性值：

obj.price = 8000;

// 删除对象的属性值

delete obj.type;

// 调用对象的方法：

obj.getName();

obj[“getName”](); // 不推荐

// 遍历对象的属性（for /in 语句）

for(var o in obj){

console.log(o); // o是字符串类型

console.log(obj[o]); // console.log(obj.o); 错误

}

学习心得：

1. Javascript内置对象
2. 内置对象Number

所有javascript数字均为64位

整数（不使用小数点或指数计数法）最多为15位

小数的最大位数是17，但是浮点运算并不是100%准确。

var x = 0.1 + 0.2; // 值为0.30000000000000004

如果前缀为0，则javascript会把数值常量解释为八进制数，如果前缀为0和“x”，则解释为16进制数

var y = 0377;

var z = 0xFF;

常用方法：\*.toFixed(保留位数)

\*必须为数字变量

保留位数在0 - 20之间，包含0和20。默认不保留小数位。

遵循四舍五入的原则

返回值为字符串类型的数字

var num = 10;

var n = num.toFix(2);

console.log(n); // 10.00

1. 内置对象String

String对象用于处理已有的字符块

// 字符串的长度属性：length

// 该属性属于只读的属性，不能修改

var str = “abcdef”;

console.log(str.length);

// 常用方法：

转换方法：

字符串转数组的方法：\*.split(分隔符)

字符串转换大小写的方法：\*.toUpperCase()

\*.toLowerCase()

字符转unicode编码值：\*.charCodeAt(index)

unicode编码值转字符：String.formCharCode(unicode1,unicode2....)

位置方法：

从指定字符串中从前向后查找某个字符或者字符串首次出现的位置；如果不存在，则返回-1：\*.indexOf(字符或者字符串)

从指定字符串中从前向后查找某个字符或者字符串最后出现的位置；如果不存在，则返回-1：\*.lastIndexOf(字符或者字符串)

操作方法：

字符串的拼接：\*.concat(各种数据)

字符串的截取：\*.slice(startIndex,endIndex)

\*.substring(startIndex,endIndex)

\*.substr(startIndex,length)

1. 内置对象Math

常用方法：

绝对值方法：Math.abs()

向上取整方法：Math.ceil()

向下取整方法：Math.floor()

求最大值方法：Math.max()

求最小值方法：Math.min()

次方的方法：Math.pow()

四舍五入取整方法：Math.round()

随机数方法：Math.random()

随机整数公式：

范围：min - max ==> (10 - 20)

值 = Math.floor(Math.random()\*(max - min - 1)) + min

1. 内置对象Boolean

Boolean（逻辑）对象用于将非逻辑值转换为逻辑值（true 或 false）

开发过程中一般不用逻辑对象进行逻辑值的转换，常见的方式是利用javascript的隐式逻辑转换功能达到非逻辑值转换成逻辑值的目的。

隐式转换规则：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 转换为true | 转换为false | 说明 |
| 字符串 | 任何非空字符串 | 空字符串 | 空格不是空字符串 |
| 数字 | 非0数字 | 0 |  |
| null | -- | null |  |
| undefined | -- | undefined |  |
| 对象 | 任何对象 | null |  |

1. 内置对象 Date

日期对象用于处理日期和时间。Date对象是以特定的时间段存储日期时间的，以世界统一时间（UTC）为标准，向前和向后各推285616年。

* 1. 定义（声明）Date对象

var date = new Date(); // 实例化日期时间对象

* 1. 组件方法：get方法对应的都有一个UTC方法

var date = new Date();

// 获取年的方法：date.getFullYear()

// 设置年的方法：date.setFullYear()

月的取值范围：0 - 11 0代表1月份，11月份

// 获取月的方法：date.getMonth()

// 设置月的方法：date.setMonth()

日期的取值范围：1 - 31 当超出范围后，可向前或者向后推

// 获取日期的方法：date.getDate()

// 设置日期的方法：date.setDate()

小时的取值范围：0 - 23

// 获取小时的方法：date.getHours()

// 设置小时的方法：date.setHours()

分钟的取值范围：0 - 59

// 获取分钟的方法：date.getMinutes()

// 设置分钟的方法：date.setMinutes()

秒的取值范围：0 - 59

// 获取秒的方法：date.getSeconds()

// 设置秒的方法：date.setSeconds()

// 日期大小的比较：比较的是时间戳，一个13位的数字

var now = new Date();

var date = new Date();

Date.setFullYear(2010);

if(now > date){

console.log(“比今天早”);

}else{

console.log(“比今天晚”);

}

// 得到7天之后的日期

var myDate = new Date();

myDate.setDate(myDate.getDate() + 7);

学习心得：

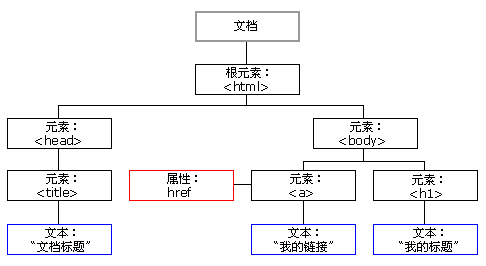
1. **HTML DOM**

DOM 是document object model（文档对象模型）的简称。

通过 HTML DOM，javascript 可访问HTML文档中的所有元素。

当网页被加载时，浏览器会根据html页面结构创建该页面的文档对象模型。

HTML DOM模型被构造为对象的树，如下图

****

节点类型： 节点名（nodeName） 节点值（nodeValue） 节点类型（nodeType）

文档节点 #document null 9

元素节点 大写的标签名称 null 1

注释节点 #comment 注释内容 8

属性节点 属性名称 属性值 2

文本节点 #text 文本内容 3

通过可编程的对象模型，javascript获得了足够的能力来操作HTML。

Javascript 能够改变页面中的所有HTML元素

Javascript 能够改变页面中的所有HTML属性

Javascript 能够改变页面中的所有CSS样式

Javascript 能够改变页面中的所有H事件作出反应

1. 查找HTML元素

通常，你需要通过javascript操作HTML页面中的元素。

为了做到这件事情，您必须首先找到钙元素。有六种方法来做这件事：

* 1. 通过id选择器找到HTML元素（最常用，效率高）

var div = document.getElementById(“box”); // box是id名称

如果找到该元素，则该方法将以对象（在div中）的形式返回该元素。

如果未找到该元素，则myDiv为null。

* 1. 通过标签选择器找到HTML元素

var dvs = document.getElementsByTagName(“div”); // div标签名称

如果未找到该元素，则dvs为[]（伪数组）。

* 1. 通过类选择器找到HTML元素

该方法不支持IE8以下版本的浏览器

var boxs = document.getElementsByClassName(“box”); // box为类名

* 1. 通过name属性找到表单元素

var form = document.getElementByName(“form”); // form为name属性名称

* 1. 通过各种选择器找到单个HTML元素

id选择器#box，类选择器.box，标签选择器标签名称以及各种伪类选择器

var box = document.querySelector(“#box”);

* 1. 通过各种选择器找到多个HTML元素

var dvs = document.querySelector(“div”);

1. HTML属性

attributes:标签上所有存在属性的集合

childElementCount:父元素下所有子元素的数量

childNodex:父元素下所有的子节点

children:父元素下所有的子元素

classList:元素上类名的集合

className:元素上的类名

clientWidth、clientHeight:元素的宽高（不包含边框）

offsetWidth、offsetHeight:元素的宽高（包含边框）

offsetLeft、offsetTop:元素的左上偏移量

firstChild:父元素下的第一个子节点

firstElementChild:父元素下的第一个子元素

lastChild:父元素下的最后一个子节点

lastElementChild:父元素下的最后一个子元素

innerHTML:元素的内部结构

textContent、innerText:元素内部的文本内容

previousSibling:上一个兄弟节点

previousElementSibling:上一个兄弟元素节点

nextSibling:下一个兄弟节点

nextElementSibling:下一个兄弟元素节点

nodeName:节点名称

nodeType:节点类型

nodeValue:节点值

以on开头的都是DOM0级事件

parentNode、parentElement:父节点、父元素

scrollLeft、scrollTop:滚动左上边距

style:元素的行内样式对象

tagName:元素的标签名称

1. 标签上属性的操作方法：

获取属性方法：元素.getAttribute(属性名称)

设置属性方法：元素.setAttribtue(属性名称,属性值)

删除属性方法：元素.removeAttribute(属性名称)

1. DOM的操作方法

创建元素节点的方法：document.createElement(标签名称)

创建文本节点的方法：document.createTextNode(文本内容)

追加节点的方法：父元素.appendChild(子节点)

删除节点的方法：父元素.removeChild(子节点)

替换节点的方法：父元素.replaceChild(新节点,旧节点)

插入节点的方法：父元素.insertChild(新节点,旧节点)

克隆节点的方法：节点.cloneNode(true || false)

碎片整理的方法：document.createDocumentFragment()

1. 获取元素使用样式最终值的方法

以下两种方法获取的样式值带单位，例如100px

IE：元素.currentStyle[属性名称]

非IE：window.getComputedStyle(元素，伪类选择器)[属性名称]

学习心得：

1. **Javascript事件**

对事件做出反应

我们可以在事件发生执行javascript，比如当用户在HTML元素上点击时。

事件分为三种：

1. 内联模型：

<button onclick=”alert(1)”>点击</button>

1. DOM0级事件：

DOM0级之间都是以on开头，后跟上事件名称

绑定事件：元素.on+事件名称 = function(){}

删除事件：元素.on+事件名称 = null

1. DOM2级事件：（暂略，后面讲解）
2. 事件类型
   1. 鼠标事件

点击事件（click）、双击事件（dblclick）、鼠标经过事件（mouseover）、鼠标移开事件（mouseout）、鼠标进入事件（mouseenter）、鼠标离开事件（mouseleave）、鼠标移动事件（mousemove）

* 1. 键盘事件

键盘摁下事件（keydown）、键盘弹起事件（keyup）、键盘摁压事件（keypress）

* 1. HTML事件

表单提交事件（submit）、表单重置事件（reset）、改变事件（change）、获取焦点事件（focus）、失去焦点事件（blur）、滚动事件（scroll）

1. 事件对象

非IE：元素.on+事件名称 = function(event){}

IE：window.event

1. 阻止默认行为

IE：event.returnValue = false;

非IE：event.preventDefault();

1. 事件流

事件捕获：从最不具体的元素向最具体的元素捕获

事件冒泡：从最具体的元素向最具体的元素冒泡

1. 阻止事件冒泡

IE：event.cancelBubble = true;

非IE：event.stopPropagation();

1. 事件委托：利用事件冒泡的原理，把同类型的事件委托到其父辈元素上。
2. 目标元素

IE：event.srcElement;

非IE：event.target;

1. **HTML BOM**

BOM 是浏览器对象模型（Browser Object Model）

浏览器对象模型（BOM）使javascript有能力与浏览器“对话”（操作浏览器）。

浏览器对象模型尚无正式标准。

由于现代浏览器已经（几乎）实现了javascript交互性方面的相同方法和属性，隐刺常被认为是BOM的方法和属性。

1. window 对象

所有浏览器都支持window对象。它表示浏览器窗口。

所有javascript 全局对象、函数以及变量均自动成为对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

甚至HTML DOM 的 document 也是 window对象的属性之一。

1. window 尺寸

有三种方法能够确定浏览器窗口的尺寸（浏览器的饰扣，不包括工具栏和滚动条）

对于Internet Explorer、Chrome、Firefox、Opera、yiji Safari：

window.innerWidth、window.innerHeight // 浏览器的可视宽高

对于Internet Explorer8、7、6、5：

document.documentElement.clientWidth || document.body.clientWidth

document.documentElement.clientHeight || document.body.clientHeight

1. 打开新窗口

window.open()

1. window screen屏幕

window.screen // 对象包含有关用户屏幕的信息

screen.availWidth // 可用的屏幕宽度，以像素计，减去界面特性，比如窗口任务栏

screen.availHeihgt // 可用的屏幕高度，以像素计，减去界面特性，比如窗口任务栏

1. Window location

Window.location 对象用于获得当前页面的地址（URL），并把浏览器重定向到新的页面。

location.hostname // 返回web主机的域名

location.pathname // 返回当前页面的路径和文件名

location.port // 返回web主机的端口

location.protocol // 返回所使用的的web协议

location.href // 返回当前页面的url

location.reload() // 重新加载页面（刷新页面）

location.replace() // 跳转到新的页面（不能返回）

1. Window history历史

返回上一级浏览历史记录：history.back();

跳转下一级浏览历史记录：history.forward()

跳转任意一级浏览历史记录：history.go()

1. Window navigator

Window.navogator 对象包含有关访问者浏览器的信息。可以用于浏览器检测，但由于navigator 数据可被浏览器使用者改变、浏览器无法报告晚于浏览器发布的新操作系统等原因可能误导开发者。UA信息可以查看navigator.userAgent。

1. 间歇调用和延时调用

间歇调用：setInterval(函数,事件间隔)

在一个设定的时间间隔之后来重复执行代码，而不是在函数被调用后立即执行。称之为计时事件。实现计时事件的函数成为定时器

延时调用：setTimeout(函数,事件间隔)

在一个设定的时间间隔之后来执行代码（只执行一次），而不是在函数被调用后立即执行。称之为计时事件。实现计时事件的函数成为定时器

1. 清除间歇调用和清除延时调用

clearInterval(间歇调用返回值)

clearTimeout(延时调用返回值)

学习心得：

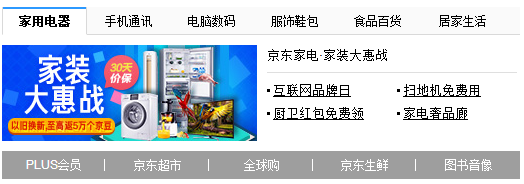
2. Javascript有多少种数据类型？分别是什么？
3. 变量值的互换：
4. null和undefined的区别？
5. == 和 === 的区别？
6. i++和++i的区别？
7. 简述逻辑运算符？
8. 运算符的优先级？
9. 流程控制语句有几种？以及使用总结？
10. 简述if双分支和三元运算符？
11. 判断一个字符串以某个字母开头？
12. 循环语句有几种以及使用场景？
13. 嵌套循环的执行顺序？
14. 成绩输入判断
15. 订单号输入判断
16. 灯泡开关



1. 购物车商品数量加减



1. tab切换



1. 实现字符串的反转
2. 实现字符串的去重
3. 统计字符串中某个字符出现的最大次数并显示该字符
4. 实现驼峰命名：get-element-by-id==>getElementById
5. 千分符：12345678==>12,345,789
6. 求1-1000范围内的质数
7. 99乘法口诀表
8. 金字塔

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\*

\* \*

\* \*

\* \*

\* \* \* \* \*

1. 在浏览器上输出1 - 1000之间的质数，每6个换行
2. 有一群猴子，摘了一堆桃子，第一天吃掉所有桃子的一半，没过瘾，又多吃了一个，第二天吃掉剩余桃子的一半，又没过瘾，又多吃了一个，以此往复，到第十天，还剩下一个桃子。问：猴子摘了多少颗桃子？
3. 现在共有100匹马跟100块石头，马分3种，大型马；中型马跟小型马。其中一匹大马一次可以驮3块石头，中型马可以驮2块，而小型马2头可以驮一块石头。问需要多少匹大马，中型马跟小型马？（问题的关键是刚好必须是用完100匹马）
5. 2/1 3/2 5/3 8/5 13/8 21/13..... 20这样的数字，求和
6. 数组的去重
7. 水仙花数
8. 1-1000之间同时被3/5/7整除的数字
9. 实现数组内数字的求平均数
10. 数组的排序（不使用系统方法）
11. 字符串的方法以及使用
12. 数组的方法以及使用
13. 数学对象的方法以及使用
14. 日期时间对象的方法以及使用
16. 封装函数：求任意范围内的n个不重复的随机数字
17. 封装函数：ar url = "http://www.baidu.com? name=张三& sex=男&age=27&type=1 ";

把上述的url地址中？之后的字符串处理成

{

name: "张三",

sex: "男",

age: "27",

type: "1"

}

function fn(url){

// 加工

// 出厂

}

fn(url)

1. 封装函数：实现简单的数学运算

function cal(num1,num2,oprate){

}

cal(2,3,”\*”);

1. 封装函数：实现任意多个实参的求和
2. 封装函数：忽略字符串中所有的空格
3. 封装函数：实现数组indexOf方法的功能
4. 封装函数：统计数组中元素出现的个数
5. 封装函数：实现getElementById方法的功能
6. 封装函数：实现getElementsByClassName方法的兼容
7. 封装函数：不使用系统方法，实现字符串或者数组的去重
9. 用javascript实现兼容所有浏览器的placeHolder
10. 封装一个函数：实现传入参数的返回类型
11. 根据一个由对象组成的数组，并把该数组中的数据渲染到表格中
12. 一个数组中有若干个对象，每个对象中有一个id属性，id属性的值为number类型的，按照id属性的值，对该数组进行排序
13. 实现表单验证
15. 闭包
16. 递归
17. Javascript引用类型
18. 函数进阶
19. Javascript作用域
20. Javascript中this指针
21. Javascript函数中的隐藏对象arguments
23. 随机点名



1. 随机抽奖



1. 无缝滚动



2. 三级联动



1. 日历



1. 购物车



2. 淘宝轮播图



1. 京东放大镜

