

Lecture 6

JavaScript基础



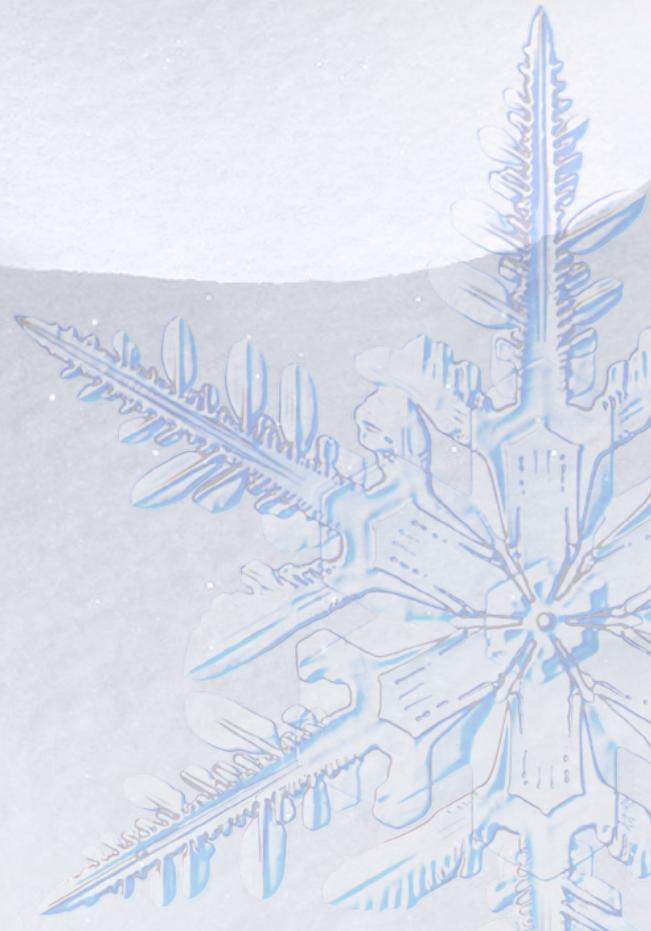


目录

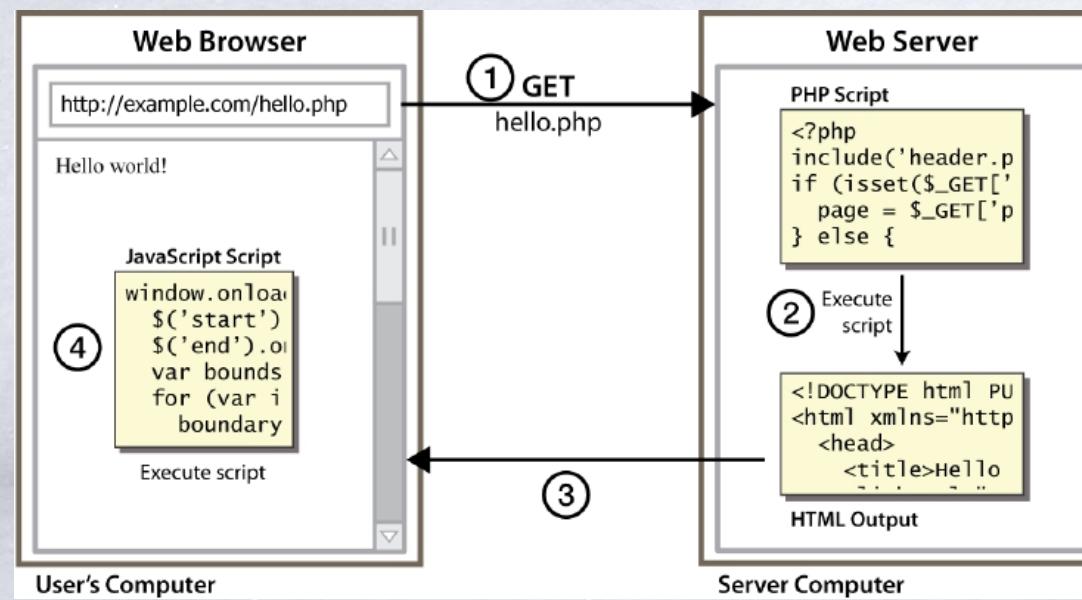
* 客户端基础

* JavaScript概述

* JavaScript基本语法



客户端脚本



客户端编程

* HTML适合于开发静态页面

- * 可以指定文本/图像布局, 链接, ...
- * Web页面每次访问时看起来都是一样的
- * 为了开发交互/响应式页面, 必须以某种形式集成编程

* 客户端编程

- * 程序是用单独的编程(或脚本)语言编写的
 - * 例如, JavaScript, JScript, VBScript
- * 程序被嵌入到Web页面的HTML中, 使用(HTML)标记来标识程序组件
 - * 例如, <script type="text/javascript"> ... </script>
- * 浏览器在加载页面时执行程序, 将程序的动态输出与HTML的静态内容集成在一起
- * 还允许用户(客户端)输入信息并对其进行处理, 可在将输入提交到远程服务器之前对其进行验证

客户端与服务器端编程

* 客户端脚本(JavaScript)的好处:

- * 可用性:可以修改页面而不必返回到服务器(更快的UI)
- * 效率:可以在不等待服务器的情况下对页面进行小而快速的更改
- * 事件驱动:可以响应用户操作, 如点击和按键源代码
- * 平台独立性:任何支持脚本的浏览器都可以解释代码

* 服务器端编程(例如PHP)的好处:

- * 安全性:可以访问服务器的私有数据;客户端看不到
- * 兼容性:不受浏览器兼容性问题的影响
- * 功能全:可以写文件, 打开与服务器的连接, 连接数据库, ...

常见脚本任务

✿ 向Web页面添加动态特性

- * 表单数据的验证(可能是最常用的应用程序)
- * 图像翻转
- * 时间敏感或随机页面元素
- * 处理cookies

✿ 定义Web接口

- * 利用按钮，文本框，可点击的图像，提示等

✿ 客户端脚本的限制

- * 因为脚本代码嵌入在页面中，所以它对外界是可见的
- * 出于安全原因，脚本所能做的事情是有限的
 - * 例如，无法访问客户端的硬盘驱动器
- * 因为它们被设计为在任何机器平台上运行，所以脚本不包含特定于平台的命令
- * 脚本语言功能不全
 - * 例如，JavaScript对象非常粗糙，不适合大型项目开发

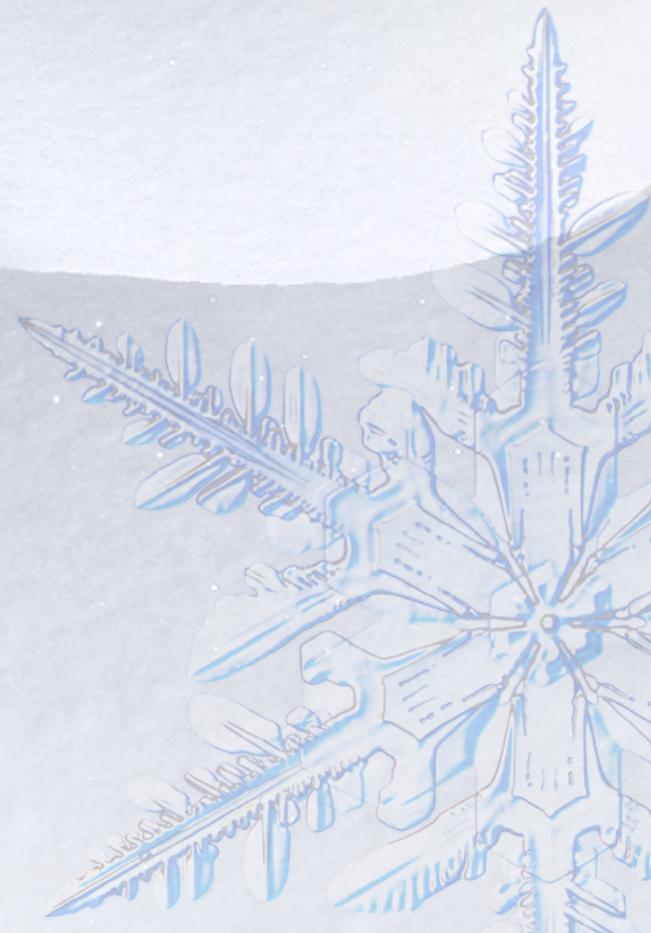


目录

* 客户端基础

* JavaScript概述

* JavaScript基本语法



为何 JavaScript?



从“玩具”到流行

- * 网页, AJAX, Web 2.0 / 3.0
- * 大量web应用
- * APP
- * 控制硬件, Arduino, Tessel, Espruino, NodeBots等, 应用于嵌入式系统、IOT、机器人等领域.
- * 桌面程序 (Electron) , SpaceX 龙飞船的触控 UI 基于 Chromium + JavaScript 技术栈开发, 开放的 Web 技术就此成为了人类首个应用到载人航天领域的 GUI 技术栈
- * 服务端
- * 命令行工具

Always bet on JS



Brendan Eich 在 2011 年一次名为「JSLOL」[Eich 2011e] 的会议演讲中，是这么描述 JavaScript 的：

- * 最早他们说 JavaScript 没法做「富互联网应用」。
- * 然后他们说 JavaScript 没法快起来。
- * 然后他们说 JavaScript 没法修复语言问题。
- * 然后他们说 JavaScript 没法做多核与 GPU 运算。
- * 他们每次都错了！
- * 我建议：永远押宝在 JS。

<https://github.com/doodlewind/jshistory-cn>

<https://cn.history.js.org/>



阿特伍德定律



Atwood's Law是Jeff Atwood在2007年提出的

- * “any application that can be written in JavaScript, will eventually be written in JavaScript.”
- * 任何可以用JavaScript来写的 应用， 最终都将用JavaScript来写”

The screenshot shows a Twitter post from Jason Mayes (@jason_mayes). The post includes three categories at the top: "JavaScript Developers" (red), "Non-JS Developers" (yellow), and "Others" (blue). The tweet text reads: "So #JS made it into space! #SpaceX uses #Chromium + #JavaScript for the Dragon 2 flight interface along with C++ for flight computers. JavaScript is not a toy folk, time to accept it. #NASA @BrendanEich space.stackexchange.com/questions/9243...". Below the text is an image of a SpaceX Dragon 2 cockpit with multiple touchscreens displaying complex flight data. The bottom of the screen shows the Twitter interface with a timestamp (4:23 AM · Jun 1, 2020), a link to Twitter for Android, and engagement metrics (614 Retweets, 1.4K Likes).

JavaScript的优势

性能

- * JIT
- * 垃圾收集和动态绑定的开销

对象

- * JavaScript使用原型继承模型。

语法

- * 对于任何有C家族语言使用经验的人来说很熟悉，比如c++、Java、c#和PHP。

一等函数

- * JavaScript中的几乎所有东西都是对象，包括函数。

事件

- * 在浏览器内部，一切都在一个事件循环中运行。

可重用性

- * 最可移植、可重用的代码
- * JavaScript可以模块化和封装

发展史

- ✿ 由Netscape的Brendan Eich 开发
- ✿ JScript , Microsoft
- ✿ European Computer Manufacturers Association (ECMA)标准化
 - * <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- ✿ ECMAScript 2015
- ✿ ECMAScript 2016

JavaScript概述

- ✿ JavaScript是一种脚本语言
- ✿ JavaScript程序由JavaScript解释器/引擎计算和执行
 - * Rhino, SpiderMonkey, V8, Squirrelfish
- ✿ 主流用途和用法:
 - * 在运行时公开应用程序的对象，用于定制/嵌入用户逻辑。
 - * 改进网站的用户界面
 - * 让你的网站更容易浏览
 - * 在不重新加载页面的情况下替换页面上的图像
 - * 表单验证
 - * 页面修饰和特效
 - * 动态内容操作
 - * 新兴的Web 2.0:客户端功能在客户端实现
 - * 还有很多.....



JavaScript vs. Java

- ✿ 解释的，不是编译的
- ✿ 更宽松的语法和规则
 - * 更少、更“松散”的数据类型
 - * 变量不需要声明
 - * 错误通常是无声的(少数例外)
- ✿ 关键结构是函数而不是类
 - * “一等”函数
- ✿ 包含在网页中，并与其HTML/CSS内容集成



使用<script>

* 为网页嵌入脚本:

```
<script>  
    script commands and comments  
</script>
```

* 访问外部的脚步:

```
<script src="url">  
    script commands and comments  
</script>
```

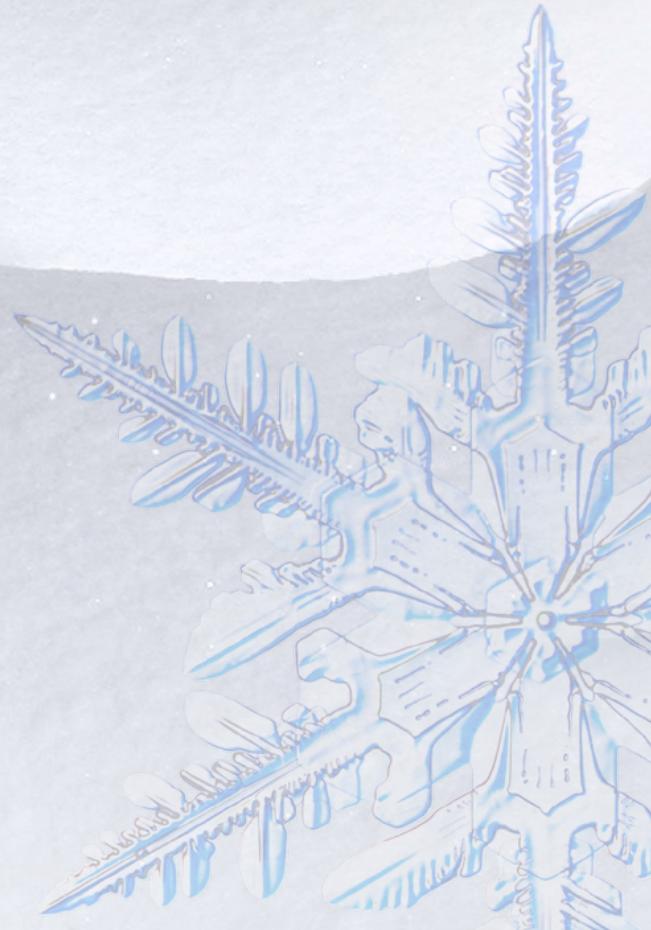


目录

❄️ 客户端基础

❄️ JavaScript概述

❄️ JavaScript基本语法



语言基础

❄ JavaScript大小写敏感

- * HTML 大小写不敏感; `onClick`, `ONCLICK`, ... are HTML

❄ 以回车或分号 (;)结束的语句

- * `x = x+1;` same as `x = x+1`
- * 分号减少错误

❄ 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。

注释(同Java)

* 与Java的注释语法相同

- * /* multi-line comment */
- * // single-line comment

对象

* 对象是命名属性的集合

- * 简单视图:哈希表或关联数组
- * 可以通过一组名称:值对进行定义
 - * `objBob = {name: " Bob", grade: 'A', level: 3};`
- * 新成员可以随时添加
 - * `objBob.fullname = 'Robert';`
- * 可以有方法

* 数组、函数也是对象

- * 对象的属性可以是函数(=方法)

变量与类型

❄ 变量

- * var, let, const(case sensitive)
- * Define implicitly by its first use, which must be an assignment
 - * Implicit definition has global scope, even if it occurs in nested scope?

❄ 类型没有指定，但JS确实有类型("loosely typed")

- * Number, Boolean, String, Array, Object, Function, Null, Undefined
- * can find out a variable's type by calling typeof

```
var name="Gates", age=60, job="CEO";
var carname="Benz";
var carname;
```

数值

- * 数值在内部是通过双精度浮点型的形式表示的 (没有 int vs. double)
- * 同样的运算符:+ - * / % ++ -- etc
- * 优先级相似java
- * 许多运算符自动转换类型:
 - * "2" * 3 is 6

```
var x=30;  
var y=4.2;  
var car_count=10;
```

0.1+0.2=?

IEEE 754

* <https://0.3000000000000004.com/#ada>

* EcmaScript的安全整数

- * Number.MAX_SAFE_INTEGER, 9007199254740991
- * 最大安全整数和最小安全整数之间的整数统称为安全整数
- * 54位有符号整数类型
- * 只有在所有的运算因子、运算结果以及中间结果都是安全整数的情况下，才能精确的整数计算，才适用于加法结合律和乘法分配律，一旦有一项的值不是安全整数，变不可控。

* JS仅有number这个数值类型，而number采用IEEE 754 64位双精度浮点数编码。
而浮点数表示方式具有以下特点：

- * 1. 浮点数可表示的值范围比同等位数的整数表示方式的值范围要大得多；
- * 2. 浮点数无法精确表示其值范围内的所有数值，而有符号和无符号整数则是精确表示其值范围内的每个数值；
- * 3. 浮点数只能精确表示 $m \times 2^e$ 的数值；
- * 4. 当biased-exponent为 $2^{e-1}-1$ 时，浮点数能精确表示该范围内的各整数值；
- * 5. 当biased-exponent不为 $2^{e-1}-1$ 时，浮点数不能精确表示该范围内的各整数值。

* 由于部分数值无法精确表示（存储），于是在运算统计后偏差会愈见明显。

- * 通常推荐先将数字转换成整数。
- * 比较两个浮点数差值的绝对值，是否超过误差精度

```
function compareTwo(n1,n2) {  
    return Math.abs( n1 - n2 ) < Number.EPSILON;  
}  
compareTwo(0.1+0.2, 0.3);
```

- * <https://github.com/tc39/proposal-decimal>

String

- * 方法: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
 - * charAt returns a one-letter String (there is no char type)
- * length 属性 (非 Java 中的方法)
- * "" 或 ''
- * 连接 + :
 - * 1+1 is 2, but "1"+1 is "11"

```
var txt="Hello world!";
document.write(txt.length)
```

String

* 转义，与Java中的相同: \' \" \& \n \t \\

* 在数值和字符串之间转换:

```
var x=30;  
var s1="" +x;           //“30”  
var n1=parseInt("10 oranges"); //10  
var n2=parseFloat("hello"); //NaN
```

* 访问字符串的字符:

```
var firstLetter = s[0];          //fails in IE  
var firstLetter = s.charAt(0);  
var last Letter = s.charAt(s.length - 1);
```

Old:

```
var message = "The user " + user.firstName + " " + user.lastName +  
" cannot be " + action + " because " + validationError;
```

New:

```
var message = `The user ${user.firstName} ${user.lastName} cannot be  
${action} because ${validationError}`;
```

布尔式犯蠢类型

* 任何值都可以用作布尔值

- * `false`: 0, -0, 0.0, NaN, "", `null`, and `undefined`
- * `true`: anything else

* 将值显式转换为布尔值

- * `var boolValue = Boolean(otherValue);`
- * `var boolValue = !!(otherValue);`

```
if (o !== null) ....
```

null, NaN, undefined



NaN

- * isNaN()
- * typeof(NaN)
- * 比如字符串转换成数值失败时



undefined



null



- * typeof(null) -> object

(-)Infinity: 大到无法表示的值

```
var x = 1000 / 0;  
isNaN(x);
```

例子

* 平常业务中比较建议尽量不要使用 == 和 !=。这两个比较的时候会做一些强制的类型转换，所以比较结果很可能有误。务必使用 === 和 !==。

`undefined < null`

`undefined > null`

`undefined === null`

`NaN === NaN`

`NaN !== NaN`

`"11" < "2"`

`"2" < "5"`

`5 < "11"`

`false`

`false`

`false`

`false`

`true`

`true`

`true`

`true`

Math对象

- ❄ methods: abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan
- ❄ properties: E, PI

```
var pi_value=Math.PI;  
var sqrt_value=Math.sqrt(30);
```

逻辑运算符

* > < >= <= && || != === !==

* 大多数逻辑运算符自动转换类型:

- * `5<"7"` 为 true
- * `42 == 42.0` 为 true
- * `"5.0" == 5` 为 true

* === 与 !== 是严格的相等检查，同时比较类型和值

- * `"5.0" === 5` is false

if/else 条件语句

- * 结构与Java的if/else语句相同
- * JavaScript几乎允许任何东西作为条件

```
if (condition){  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

for 循环

- * for
- * for-in
- * for-of

```
var x;  
var txt="";  
var person={fname:"Bill",lname:"Gates",age:56};  
for (x in person){  
    txt=txt + person[x];  
}
```

while, do/while

✳ 同Java

✳ break和continue关键字也和Java中的一样

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
while (cars[i]){
  document.write(cars[i] + "<br>");
  i++;
}
```

数组

- * 两种方法初始化数组
- * 长度属性(在添加元素时根据需要增长)

```
var empty=[];
var mycars=new Array();
mycars[0]="Saab"
mycars[1]="Volvo"
mycars[2] = "BMW"

var misc=new Array(1.1,true,"BMW");
```

Array方法

- * 数组可以实现多种数据结构: list, queue, stack, ...
- * 方法: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
 - * push and pop add / remove from back
 - * unshift and shift add / remove from front
 - * shift and pop return the element that is removed

```
var fruits = ["Banana", "Orange"];
fruits.push("Apple");
fruits.unshift("Mango");
fruits.pop();
fruits.shift();
fruits.sort();
alert(fruits);
```

函数

```
function functionName(parameters) {  
    //  
}
```

* 声明可以出现在函数体中

- * Local variables, “inner” functions
- * =>
 - * (arg1, arg2) => { //something here }

* 匿名函数

- * (function (x,y) {return x+y}) (2,3);

参数



参数传递

- * 基本类型按值传递，对象按引用传递



调用可以提供任意数量的实参

- * `functionname.length` : # of arguments in definition
- * `functionname.arguments.length` : # args in call