# JavaScript基础入门

第2天课堂笔记（本课程共11天）

# 目录

[JavaScript基础入门 1](#_Toc3085)

[目录 2](#_Toc11044)

[一、JavaScript作业讲解 11](#_Toc16290)

[1.1 复习 11](#_Toc29375)

[1.2 作业 14](#_Toc20125)

[二、运算符 16](#_Toc21143)

[2.1 数学运算符 16](#_Toc23132)

[2.2 关系运算符 17](#_Toc14497)

[2.3 逻辑运算符 19](#_Toc27400)

[2.4 赋值运算符 21](#_Toc8795)

[三、条件分支语句 24](#_Toc11834)

[3.1 if语句 24](#_Toc23754)

# 一、JavaScript作业讲解

## 1.1 复习

昨天讲解的JS是编程最最简单的一些基础知识，复习一下：

JavaScript书写环境，<script>标签

|  |
| --- |
| 1. <script type=”text/javascript”> 2. </script> |

alert()函数，至于什么是函数，先别管。alert弹出警告框

|  |
| --- |
| 1. alert("你好"); |

控制台F12，console就是用来检测程序有没有错误的

|  |
| --- |
| 1. console.log("你好"); |

我们已经知道了两种常见错误，

[引用错误？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError)

[语法错误？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/SyntaxError)

SyntaxError 语法错误 ：比如符号是中文的

ReferenceError 引用错误 ： 比如变量没有声明

数字的直接量：数字、NaN、Infinity

2/0 结果是Infinity , 0/0结果是NaN

字符串的直接量：双引号、单引号

|  |
| --- |
| 1. "字符串" 2. "45454" 3. "" |

变量： 变量的命名开头必须是字母、美元符号、下划线，后续的可以有字母、美元符号、下划线、数字；换句话说，不能以数字开头。不能有怪异符号，不能是保留字和关键字。

变量的定义，或者叫做变量的声明，用var

|  |
| --- |
| 1. var a = 100; |

var一次之后，就可以不用var了。

用逗号隔开多个：

|  |
| --- |
| 1. var a = 100 , b = 200; |

JS中变量声明可以提升，也就是说，可以先使用变量，然后再声明 (预解析)

|  |
| --- |
| 1. a = 100; 2. var a; 3. Console.log(a) |

一个变量如果仅仅var了，但是没有赋值，默认的值是

|  |
| --- |
| 1. undefiend |

变量的类型，实际上就是所有值的类型

基本的类型5种：

number类型 ： 数字、NaN、Infinity

string类型 ： 字符串

undefined类型 : 只有一个值，就是undefined

null类型 ： 只有一个值，就是null

boolean类型 ： 只有两个值true和false

引用类型：

我们日后再说

用typeof能够检测值的类型。

parseInt、parseFloat

parseInt是无条件的截断小数点后面的任何东西，不四舍五入。

|  |
| --- |
| 1. parseInt(3.9888); // 3 2. parseInt("3年3月"); //3 3. parseInt("哈哈3年3月"); //NaN |

parseFloat就是获取小数点的，将string变为浮点数

|  |
| --- |
| 1. parseFloat("3.14.123") //3.14 |

数学运算符

+ - \* / %

% 叫做求余，也有的书上求模

|  |
| --- |
| 1. 100 % 10 //0 2. 15 % 5 //0 |

|  |
| --- |
| 1. 3454325435 % 3 |

的计算结果只能是0、1、2。 不能是3、4、5、6……

运算顺序是先选乘性(\*/%)的，然后是加性(+-)的。

|  |
| --- |
| 1. 2 + 5634 \* 5 % 5 / 2 |

解：原式 = 2 + 28170 % 5 / 2

= 2 + 0 / 2

= 2

## 1.2 作业

1）计算下列算式，并将结果输出：



|  |
| --- |
| 1. console.log((123 + 45 \* 78) / (34 + 543) - 9 \* (324 - 34)); |

2）表示气温有两种方法：摄氏度和华氏度。两者的关系是：



编写程序，让用户输入摄氏度，弹出对应的华氏温度。

|  |
| --- |
| 1. <script type="text/javascript"> 2. //第一步，让用户输入数字 3. var sheshidu = parseFloat(prompt("请输入摄氏温度")); 4. //第二步，计算 5. var huashidu = 9 / 5 \* sheshidu + 32; 6. //第三部，显示 7. alert("你好，华氏度是" + huashidu + "\n给钱！"); 8. </script> |

3） 用JS计算下列算式，并将结果在弹出：

 alert(Math.pow((23+Math.pow(5,7)/45),2))

这道题目就是用来练习嵌套的，

|  |
| --- |
| 1. Math.pow(**(**23 + **Math.pow(5,7))**/45,2); |

4） 编写程序，提示用户输入六边形的边长，然后弹出它的面积，计算六边形面积的公式是：



|  |
| --- |
| 1. <script type="text/javascript"> 2. //第一步，让用户输入边长 3. var a = parseFloat(prompt("请输入六边形的边长")); 4. //第二步，计算 5. var s = 3 \* Math.sqrt(3) / 2 \* Math.pow(a,2); 6. //第三步，显示 7. alert("面积是" + s); 8. </script> |

1. 如果今天是星期二，那么1000天后是星期几？用户输入一个天数，计算这个天数后是星期几。
2. 0123456 1000%7 = 6 + 2 = 3%7 = 3 (tianshu +1)% 7 + 1

如果今天是星期2，那么7天后是也是星期2，14天后也是星期2。

今天1天后是星期3，那么8天后也是星期3,15天后也是星期3。

所以49天后，是星期2。因为49 % 7 结果是0

所以79天后，是星期4。 因为79 % 7 结果是2，相当于2天后

所以，1000天后的星期就要看简化为相当于几天后？

1000 % 7 结果是6， 所以就相当于6天后的星期。

6天后就是星期1。

综合算式就是： (2 + 1000 % 7) % 7

如果用户输入天数，那么就是

|  |
| --- |
| 1. (2 + a % 7) % 7 |

6） 用户输入一个三位数，弹出各个数位的和。

比如：

用户输入155，就弹出11 1+5+5=11

用户输入316，就弹出10

用户输入989，就弹出26

用户输入678，就弹出21

415 => 1 115 /100 =parseInt(4.11)

356 => 1 356 %100 =56 / 10 = parseInt(5.6)

316 => 6 316%10 =6

**这里就有一个“算法”的概念，“算法”就是人想出来的让机器解决事情的方法。**

机器alphaGo能打败李世乭，实际上就是算法为王，算法非常好。

机器能干的事情就是计算、流程控制。

用户输入数字简单，但是拆分数字需要我们的智慧了。

比如用户输入316，如何单独得到各个数位？？？

个位简单， 就是 316 % 10 得到6

百位也简单，就是parseInt(316 / 100) 得到3

十位比较复杂， parseInt(316 / 10) % 10 得到1

# 二、运算符

[运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators)

运算符（Operators，也翻译为操作符），是发起运算的最简单形式。

运算符的分类见仁见智，我们的课程对运算符进行如下分类：

数学运算符(Arithmetic operators)

比较运算符(Comparison operators)

逻辑运算符(Logical operators)

赋值运算符(Assignment operators)

按位运算符(Bitwise operators)

条件 (三元) 运算符(Conditional operator)

## 2.1 数学运算符

+ - \* / % ()­

■ 这里面要知道%的意思，我们昨天已经算过很多了，不赘述。

■ 要知道计算顺序，先乘除求余，然后加减，我们昨天已经算过很多了，不赘述。

parseInt parseFloat

■ 隐式转换，所有带有字符串的运算都会尽可能的转为数字进行计算，加号比较特殊。

数学运算符的正统，是number和number的数学运算，结果是number。出于面试的考虑，有一些奇奇怪怪的数学运算：

数学运算中，**只有纯字符串、布尔值、null能够帮你进行隐式转换**：

ECMAScript规范对于运算的规范？ <http://www.ecma-international.org/ecma-262/6.0/>

JavaScript秘密花园的翻译？http://bonsaiden.github.io/JavaScript-Garden/zh/#types.casting

|  |
| --- |
| 1. //隐式转换：就是没有写parseInt()、parseFloat()自己帮你转格式 3. console.log(3 \* **"8"**); //24 4. console.log(**"3"** \* **"8"**); //24 5. console.log(**"48"** / "2"); //24 6. console.log(**"24"** % 55); //24 |

|  |
| --- |
| 1. console.log(3 \* **null**); //0 隐式转换的时候**null将被转为0** 2. console.log(3 \* **false**); //0 隐式转换的时候**false将被转为0** 3. console.log(3 \* **true**); //3 隐式转换的时候**true将被转为1** |

**不纯的字符串和undefined是不能帮你进行隐式转换的**，结果都是NaN

|  |
| --- |
| 1. console.log(3 \* **"8天"**); //NaN 数学运算中，不纯的字符串没法隐式转换 2. console.log(3 \* **undefined**); //NaN 数学运算中，undefined不能进行隐式转换 |

加法比较特殊，因为+同时是加法和连字符的符号，所以加法在面对字符串的时候没有隐式转换

|  |
| --- |
| 1. //加法没有隐式转换 2. console.log(3 + **"8"**); //38 3. console.log(3 + **undefined**); //NaN 4. console.log(3 + **null**); //3 5. console.log(3 + **false**); //3 6. console.log(3 + **true**); //4 |

总结：

无论哪种运算，只要出现了undefined参与运算，结果都是NaN。

然后"4"、false、true、null都能进行隐式转换。

加号比较特殊，面对"4"没有隐式转换的

## 关系运算符

[比较操作符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Equality)

> 大于

< 小于

>= 大于等于

<= 小于等于

== 等于

!= 不等于

=== 全等于

!== 不全等

关系运算符的正统，number和number进行数学运算，得到的答案boolean。

[Boolean?](https://developer.mozilla.org/zh-CN/docs/Glossary/Boolean)

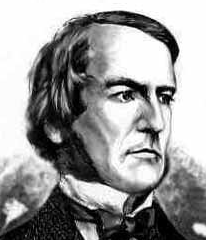
|  |
| --- |
| 1. console.log(8 > 5); 2. console.log(7 < 4); |

关系运算的结果，是boolean类型的。



true和false叫做布尔值。boolean类型，boolean类型只有两个值，就是true和false。表示真、假。

[布尔](http://baike.baidu.com/link?url=uH7zOTr9WwkgiGRQpPihS5m_TrqIFI43I94bIIQidge6CuFuCNecu0NNFC4g2b6jE9q5QuWevY9VVqwR7m3MOWveexE-D0PRkHKU7pzAbOIJ4Q0gUycYESGKNyzBtb9y4xerHZGkf0XIsxgRxREcJhgp6MOU_5M58yDxdUGf4S4a4xXpV1m6zrARGbxqCe9O)？



== 我们叫做“相等判断”，它会帮你进行一个隐式转换，尽可能的得到true的答案：

|  |
| --- |
| 1. console.log(5 == "5"); //true |

===我们叫做“全等判断”，**不仅仅比较数值是否相等，还比较类型是否相等**

[相等和全等？](http://dorey.github.io/JavaScript-Equality-Table/)

!= 是==的反面，如果==运算是true，那么!=就是false

!==是===的反面，如果===运算是true，那么!==就是false

|  |
| --- |
| 1. console.log(5 != "5"); //false ， 脑子要反着想一下，5=="5"结果是t，所以就是f 2. console.log(5 !== "5"); //true ， 脑子要反着想一下，5==="5"结果是f，所以就是t |

正统的运算讲完了，number和number进行关系运算，结果是boolean。

1. string 和 string 也能够进行关系运算，**比较的就是字符编码顺序**

字符编码顺序，就是数字、大写字母、小写字母

|  |
| --- |
| 1. "a" < "b" //true 2. "A" < "B" //true 3. "A" < "a" // true ，大写字母在字符集里面是在小写字母前面 4. "1" < "A" //true ，数字在字母前端 5. "blank" < "blue" //true 因为一位一位比，直到比出大小 6. "25678" < "3" //true 因为是string和string比，比的是字符编码顺序 |

② 与数字进行关系运算时，纯数字字符串被转为数字，null转换为0，true转换转为1， false转换为0

null不能进行和0的相等判定

|  |
| --- |
| 1. null < 0.00001 //true 2. null > -0.0001 //true 3. null == 0 //false 具体原因，我们后面讲解Object的时候介绍 4. false == 0 //true 5. true == 1 //true |

③ NaN不等于自己，不全等于自己

|  |
| --- |
| 1. NaN == NaN //false 2. NaN === NaN //false 3. NaN != NaN //true 4. NaN !== NaN //true |

④ string 和 number比，string会被隐式转换为number

|  |
| --- |
| 1. "25" < 3 //false |

需要注意的是，我们已经了解了一些不正统的运算，所以不要出洋相，不能连续使用关系运算符！！

比如我们想验证3大于2,2大于1：

表达式：

|  |
| --- |
| 1. 3 > 2 > 1 |

的值是多少？

解：原式=(3>2) >1 = true > 1 = false (因为true会被当做1来与1进行比较)

也就是说，不能连续使用关系运算符！！因为一旦连续使用了，实际上还是从左至右计算，所以就有上一步的boolean参与了下一步的运算。

**剧透一下，如果想要使用连续关系运算，1<2 && 2<3**

## 2.3 逻辑运算符

[逻辑运算符](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Logical_Operators" \l "Logical_NOT)就三个：

&& 逻辑与运算 并且

|| 逻辑或运算 || 或者

! 逻辑非运算 ! 反向

正统来说，参与逻辑运算的是boolean和boolean，得到的结果也是boolean

值按照真值表来定

&& 逻辑与，“且”

|  |  |  |
| --- | --- | --- |
| a && b | | |
| a | b | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 假 |
| 假 | 真 | 假 |
| 假 | 假 | 假 |

“都真才真”，“有假就假”。

命题1：“地球是圆的” 真的

命题2：“建哥很帅” 真的

命题1 **且** 命题2 真的

命题1：“1+1=3” 假的

命题2：“地球是方的” 假的

命题1 **且** 命题2 假的

|  |
| --- |
| 1. //逻辑运算符 2. console.log(true **&&** true); //t 3. console.log(true **&&** false); //f 4. console.log(false **&&** true); //f 5. console.log(false **&&** false); //f |

|| 逻辑或，“或者”的意思

|  |  |  |
| --- | --- | --- |
| a || b | | |
| a | b | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 真 |
| 假 | 真 | 真 |
| 假 | 假 | 假 |

“有真就真”，“都假才假”

命题1： 1 + 1 = 2

命题2：“建哥很帅”

命题1 或者 命题2 总结果是真

|  |
| --- |
| 1. console.log(true **||** true); //t 2. console.log(true **||** false); //t 3. console.log(false **||** true); //t 4. console.log(false **||** false); //f |

!就是“逻辑非”，相反的

|  |
| --- |
| 1. console.log(**!**true); //f 2. console.log(**!**false); //t 3. console.log(**!!!!!!!!!**false); //t |

**运算顺序是非、与、或**

|  |
| --- |
| 1. true || false && !true || false; |

解：原式 = true || false && **false** || false

= true || false || false

= true || false

= true

逻辑运算符最最有意思的事情，就是所谓的“短路语法”。

就是你发现没有，

如果计算一个且运算的时候，比如a && b，a如果就是一个false，那么就不会管b是什么，直接输出false就行了，等于说直接输出a。

如果计算一个且运算的时候，比如 a && b ，a如果就是一个true，那么也不用管b是什么，直接把b当做结果输出就行了。

也就是说，本质上计算机进行a&&b运算的时候，不是在进行逻辑分析，这小子就想着要么扔a，要么扔b。如果a是负性的，那么直接扔出a；如果a是正性的，直接扔出b。

—— 短路语法。 要么a被短路，要么b被短路。

负性的：false，null， 0， NaN， 空字符串("")，undefined

正性的：除了上面的，全是正性的。

|  |
| --- |
| 1. false && 8 //false 因为计算机发现，且运算a已经是false了，直接输出false 2. null && 8 //null 因为计算机发现，且运算a已经是false性的了，直接扔出来null 3. true && 13 //13 因为计算机发现，且运算a是true，所以总结果就是看b，直接扔出b 4. 12 && 13 //13 因为计算机发现，12当做true，所以总结果看b，直接扔出b 5. 13 && 12 //12 因为计算机发现，13当做true，所以总结果看b，直接扔出b 6. undefined && 哈哈 //undefined 不报错，因为a已经是负性的了，所以直接扔出a，哈哈不管 7. 哈哈 && undefined //报错 8. true && NaN //NaN 扔后面 |

|| 逻辑或的短路也是类似的，a||b

计算机发现a是真，那么扔a；如果a是假，那么扔b

|  |
| --- |
| 1. 0 || 18 //18 前面假，扔后面 2. 18 || 0 //18 前面真，扔前面 3. undefined || NaN //NaN 前面假，扔后面 4. NaN || undefined //undefined 前面假，扔后面 |

|  |
| --- |
| 1. 88 || 99 && 66 || 55 |

解：原式 = 88 || 66 || 55

= 88 || 55

= 88

|  |
| --- |
| 1. undefined && ("3" != 3) || NaN && null |

解： 原式 = undefined && false || NaN && null

= undefined || NaN && null

= undefined || NaN

= NaN

总结一下短路语法：

a&&b， 计算机要么执行a要么执行b。a真执行b，a假执行a；

a||b， 计算机要么执行a要么执行b。a真执行a，a假执行b。

千万不要背，从真值表中自己推倒。

## 2.4 赋值运算符

= 赋值

+= 简便写法

-= 简便写法

\*= 简便写法

/= 简便写法

%= 简便写法

++

--

[赋值运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Assignment_Operators" \l "Assignment)

赋值运算的参与者，一定是变量。

|  |
| --- |
| 1. var a = 1; 2. a **+=** 2; //这行语句等价于**a = a + 2;** 3. console.log(a); //3 |

|  |
| --- |
| 1. var b = 6; 2. b **/=** 3; //等价于b = b / 3 3. console.log(b); //2 |

|  |
| --- |
| 1. var c = 100; 2. c **%=** 10; //等价于c = c % 10; 3. console.log(c); //0 |

|  |
| --- |
| 1. var a = "我"; 2. a **+=** "爱"; 3. a **+=** "你"; 4. console.log(a); |

[++运算符？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Increment)：

|  |
| --- |
| 1. var e = 10; 2. e++; //等价于e=e+1 3. console.log(e); //11 |

++可以与输出语句写在一起，++写在变量前和写在变量后不是一个意思

a++ : 先用a的原值，然后a加1；

++a ：先给a加1，然后用a的新值

|  |
| --- |
| 1. var f = 10; 2. console.log(f++); //10 ，先引用原值，然后加1 |

等价于：

|  |
| --- |
| 1. //等价于 2. var f= 10; 3. console.log(f); //先输出f 4. f++; //然后f加1 |

|  |
| --- |
| 1. var g = 10; 2. console.log(++g); //11 ， 这次是先加1，然后输出 |

++有花式玩儿法：

|  |
| --- |
| 1. var a = 8; 2. console.log(4 + a++); //12 ， 先使用原来的a的值，就是4+8，输出12.然后a加1 3. console.log(a); //9 |

|  |
| --- |
| 1. var i = 9; 2. console.log(++i % 5); //0 , 先把i加1，然后使用i，10%5=0 3. console.log(i); //10 |

[运算符的计算顺序？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)：

++ -- !非 **→→→** 数学 **→→→** 比较 **→→→** 逻辑 **→→→** 赋值

|  |
| --- |
| 1. var a = 3 < 6 && 7 < 14; //true |

原式 = true && true

= true

|  |
| --- |
| 1. var a = 1 + 2 < 3 + 3 && 3 + 4 < 2 \* 7; |

原式 = 3 < 6 && 7 < 14

= 上一题

= true

|  |
| --- |
| 1. var a = false + true && 13; |

原式 = 0 + 1 && 13

= 1 && 13

= 13

|  |
| --- |
| 1. var a = 15; 2. false + a++ + true > 8 && 13 || 6 |

原式 = false + 15 + true > 8 && 13 || 6

= 16 > 8 && 13 || 6

= true && 13 || 6

= 13 || 6

= 13

# 三、条件分支语句

## 3.1 [if语句？](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/if...else)

[语句判断？](http://dorey.github.io/JavaScript-Equality-Table/)

如果……否则…… ， 让程序出现分支

|  |
| --- |
| 1. <script type="text/javascript"> 2. var a = 8; 3. if(a >= 60){ 4. alert("及格了"); 5. }else{ 6. alert("不及格"); 7. } 8. </script> |

语法：

|  |
| --- |
| 1. if(测试表达式){ 2. 测试表达式为真执行的语句 3. }else{ 4. 测试表达式为假执行的语句 5. } |

if英语里面的如果的意思，else就是否则的意思。else不要拼写为eles。

在语法层面有两个特例：

① 可以没有else部分

|  |
| --- |
| 1. var a = 1; 2. if(a >= 60){ 3. alert("及格了"); 4. } |

② 如果要执行的语句，只有一行语句，那么就是单行if，就可以省略大括号。//**不推荐使用**

|  |
| --- |
| 1. var a = 112; 2. if(a >= 60) 3. alert("及格了"); 4. else 5. alert("不及格"); |

注意if语句是一个结构体，注意哪些语句是在结构体中，哪些语句不是结构体：

|  |
| --- |
| 1. var a = 355; 2. if(a > 30){ 3. console.log("哈哈"); 4. console.log("嘻嘻"); 5. console.log("呵呵"); 6. }else{ 7. console.log("纳尼"); 8. console.log("呸呸") 9. } 10. console.log("么么哒"); /\*一定会执行，在if结构体外面\*/ |



如果a是3，不满足条件：



“殊途同归”

|  |  |
| --- | --- |
|  |  |

小练习：用户输入一个年龄，判断用户能不能考取驾照。交规说： 大于等于18，小于等于70才能考取驾照。

|  |
| --- |
| 1. //让用户输入年龄 2. var age = parseInt(prompt("请输入年龄")); 3. //判断 4. if(age >= 18 && age <= 70){ 5. alert("可以考取驾照"); 6. }else{ 7. alert("年龄不符合要求"); 8. } 9. alert("谢谢惠顾"); |

**多分支的if**

if……else if …… else if…… else if…… else ……

如果……否则如果……否则如果……否则如果……否则……

**跳楼现象**，用户会选择一个分支执行（跳楼），选择这个分支了，就暗含了上一个分支没有满足：

|  |
| --- |
| 1. var score = 76; 2. if(score >= 80){ 3. alert("优秀");  **//不满足条件，所以不执行，走楼梯下楼** 4. }else if(score >= 70){ 5. alert("良好"); **//走到这个楼层的，一定暗含小于85。这一楼层满足，跳楼了** 6. }else if(score >= 60){ 7. alert("及格"); **//不执行** 8. }else{ 9. alert("不及格") **//不执行** 10. } |

语法：

|  |
| --- |
| 1. if(测试表达式1){ 2. 测试表达式1为真的时候做的事情 3. }else if(测试表达式2){ 4. 测试表达式1为假，且测试表达式2为真的时候做的事情 5. }else if(测试表达式3){ 6. 测试表达式1为假，测试表达式2为假，且测试表达式3为真的时候做的事情 7. } 8. …… |

if语句的嵌套：

用户先输入自己的性别，比如男、女。然后输入自己的年龄。判断能否结婚。

男的22以上

女的20以上

if语句可以嵌套，如果里面再套一层如果：

|  |
| --- |
| 1. //判断 2. **if(sex == "男"){** 3. **//男** 4. **if(age >= 22){** 5. **alert("男同志你好，可以结婚");** 6. **}else{** 7. **alert("你还是个小正太，不能结婚！");** 8. **}** 9. **}else{** 10. **//女** 11. **if(age >= 20){** 12. **alert("女同志你好，可以结婚");** 13. **}else{** 14. **alert("你还是个小萝莉，不能结婚！");** 15. **}** 16. **}** |

楼层中else只能有一个。 else if可以有多个。