

JavaScript进阶

- JS语法、表达式及语句
- JS赋值、算数、关系运算符
- JS逻辑运算符进阶

JavaScript进阶

---JS语法、表达式及语句

参见《深入理解JS》第7、13章、《JS权威指南》第4、5章



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **JS语法、表达式及语句综述**
- **JS严格模式**
- **switch详解、for...in**



JS语法、表达式及语句综述

• 字面量

- 对象字面量 `var obj = {x:12, y:23};`
- 数组字面量 `var arr = [1,2,true,'xyz'];`

• 标识符与保留字

- 标识符用来给变量或函数进行命名，以字母、下划线或\$为开始
- 保留字：arguments、break、case、catch、class、const等（参见教材7.6）

• 表达式（expression）与语句（statement）

- 表达式代码中基本的单位，它将产生一个值，用于需要值的地方，如：`if(a>b){...}`
- 语句表示了一种行为，如：`var obj = {x:1,y:b}; //创建对象obj`
- JS期望语句的地方都可以写表达式（表达式语句），如：`x+3;`

JS语法、表达式及语句综述

• 表达式及表达式分类

- 原始表达式、对象及数组初始化表达式、函数定义表达式、属性访问表达式
- 函数调用表达式、对象创建表达式、算术表达式、关系表达式、逻辑表达式、赋值表达式

• 语句及语句分类

参见实例demo02 Part1和Part2

- 表达式语句、复合语句、条件语句(if-else、switch)、循环语句 (for、for...in)

• ES5中的块 (**ES5中没有块作用域**，所以带来了很多问题)

```
{           if(true){           if(false){
    var a = 20;      var a = 20;      var b = 30;
}           }           }
console.log(a);    console.log(a);    console.log(b);
```

内容提纲

- JS语法、表达式及语句综述
- JS严格模式
- switch详解、for...in



JS严格模式

- ES5中的运行模式

- 严格模式和非严格模式（松散模式）

- 严格模式的目的

- 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为
- 消除代码运行的一些不安全之处，保证代码运行的安全
- 提高编译器效率，增加运行速度

- 启用严格模式的方式

- 针对整个脚本文件使用 'use strict'
- 针对函数使用 'use strict'

参见实例demo04 使用严格模式的不同方式

JS严格模式下语法和行为的改变 — (全局变量)

- 严格模式下全局变量需显式声明

```
function sloppyFunc() {  
    sloppyVar = 123;  
}  
sloppyFunc();  
console.log(sloppyVar);  
123
```

```
function sloppyFunc() {  
    'use strict'  
    sloppyVar = 123;  
}  
sloppyFunc();  
console.log(sloppyVar);
```

► Uncaught ReferenceError:
at sloppyFunc (<anonymous>:5:1)
at <anonymous>:5:1

参见实例demo05

JS严格模式下语法和行为的改变 二（函数中的this）

- 一般函数中的this（严格模式）为`undefined`，非严格下为全局变量

```
> function thisTest(){  
    'use strict'  
    console.log(this);  
}  
thisTest();
```

undefined

- 可以用此特性来判断当前是否为严格模式

```
> //'use strict'  
function isStrictMode(){  
    return this===undefined?true:false;  
}  
isStrictMode();  
← false
```

JS严格模式下语法和行为改变 三（属性、变量及函数参数）

- 严格模式下禁止删除不可改变的属性和未定义的变量

```
var str = "abc";  
function sloppyFunc() {  
    str.length = 7;  
    console.log(str.length);  
}  
sloppyFunc();
```

3

```
var str = "abc";  
function strictFunc() {  
    'use strict'  
    str.length = 7;  
    console.log(str.length);  
}  
strictFunc();
```

► Uncaught TypeError: Cannot assign

- 严格模式下禁止函数参数重名

```
function f(a, a, b) {  
    return a+b;  
}  
f(2,3,4);
```

7

```
"use strict";  
function f(a, a, b) {  
    return a+b;  
}  
f(2,3,4);
```

Uncaught SyntaxError: Duplicate parameter

内容提纲

- JS语法、表达式及语句综述
- JS严格模式
- **switch详解、for...in**



switch详解

• switch语句中的case

- case在比较时使用的是全等操作符比较,因此不会发生类型转换
- case 后可以是一个表达式 (如 $i < 60$)

```
var i = "1";
switch(i){
    case 1:
        console.log("case 1 Number");
        break;
    default:
        console.log("输出: default");
}
```

输出: default

```
var i = 65;
switch(true){
    case i >= 60:
        alert('及格');
        break;
    case i < 60:
        alert('不及格');
        break;
    default:
        alert('default');
}
```

switch详解

- switch语句中的穿透性及其应用

```
var i = 1; // i=2, 3, 4
switch(i){
  case 1:
    console.log("case 1");
  case 2:
    console.log("case 2");
    break;
  case 3:
    console.log("case 3");
    // break;
  case 4:
    console.log("case 4");
}
```

case 1

case 2

从满足第一case处
开始执行，直到遇
到break为止，若都
没有break则直到
default结束为止

利用switch的穿透
性:求某月某日是一
年中的第几天

参见实例demo09 switch贯穿案例

for...in

- for...in常用来遍历对象

```
var obj = {x:10,y:20,z:"30"};  
for(var k in obj){  
    console.log(k,obj[k],typeof obj[k]);  
}
```

```
x 10 number
```

```
y 20 number
```

```
z 30 string
```

- for...in遍历数组（忽略空缺）

```
var arr = [2,, "33"];  
for(var i in arr){  
    console.log(i,arr[i]);  
}
```

```
0 2
```

```
2 33
```


The background of the slide is decorated with various abstract shapes in shades of green and yellow. These shapes, which include circles, ovals, and irregular blobs, are scattered across the top and right sides of the slide, creating a modern, organic feel.

Have a Break!



河北师范大学软件学院
Software College of Hebei Normal University

JavaScript进阶

---JS赋值、算数、关系运算符

内容提纲

- **JS赋值运算符**
- **JS算数运算符**
- **JS关系运算符**



赋值运算符（基本内容参见教材9.2）

- 注意=与==（表达式要发反写，有什么好处）

```
> var a = 34
    if(a = 45){
        console.log("hi");
    }
```

hi

< undefined

```
> var a = 34
    if(45 = a){
        console.log("hi");
    }
```

✖ Uncaught ReferenceError: Invalid left-hand side in assignment

```
> var a = 34
    if(45 == a){
        console.log("hi");
    }
```

内容提纲

- JS赋值运算符
- JS算数运算符
- JS关系运算符



算数运算符

- 算数运算符与类型转换

```
"1"+"2"; //"12"
```

```
"1"+2; //"12"
```

```
1+{}; //"1[object Object]"
```

```
true+true; //2
```

```
"5"-2; //3
```

- 一元运算符（++、--）

```
var x = "1";
```

```
console.log(++x); //2 注意++和--的隐式类型转换
```

```
var x = "1";
```

```
console.log(x+=1); //11
```

内容提纲

- JS赋值运算符
- JS算数运算符
- JS关系运算符



关系运算符

- == 与 ===

== (如果类型不同, 先转换再比较, 注: 引用类型到基本类型的转换方向)

=== (若类型不同则false, 若类型相同则判断同 ==)

回顾值类型与引用类型的比较结果 (3===3、{}==={}、NaN===NaN)

> 3===3

> NaN===NaN

> {}==={}

< true

< false

< false

- != 与 !==

!= (如果类型不同, 先转换再比较)

!== (先判断类型, 若类型不同则返回true, 若类型相同则判断同! =)

The background features several clusters of overlapping circles and ovals in various shades of green and yellow, creating a modern, organic feel.

Have a Break!



河北师范大学软件学院
Software College of Hebei Normal University

JavaScript进阶

---JS逻辑运算符进阶



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **&&与||的基本理解及应用**
- **&&与||的深层次理解（非布尔类型）**
- **&&与||在实际中的应用**



&&与||基本理解和应用

• 最常见情况（运算符两边的操作数都是布尔类型）

- 对于&&来说，除了两侧都为真时为真，其他情况都为假
- 对于||来说，除了两侧都为假时为假，其他情况都为真

```
console.log(2>1&&4<5);  
console.log(true&&(!2));  
console.log(false&&("2" == 2));  
console.log(false&&>false);
```

true

false

false

false

```
console.log(2>1||4<5);  
console.log(true||(!2));  
console.log(false||("2" == 2));  
console.log(false||false);
```

true

true

true

false

参见实例demo14

内容提纲

- &&与||的基本理解及应用
- &&与||的深层次理解（非布尔类型）
- &&与||在实际中的应用



&&与||深层次理解

- 当逻辑运算符&&和||两侧的操作数不是布尔类型时
 - 首先将左操作数转换成布尔类型
 - 对转换后的左操作数进行逻辑判断 (true or false)
 - 根据短路原则返回原始左操作数或原始右操作数
- 短路原则（忽略对右操作数的判断）
 - 对于&&，转换后的左操作数若为true，则直接返回原始右操作数，若为false则直接返回原始左操作数
 - 对于||，转换后的左操作数若为true，则直接返回原始左操作数，若为false则直接返回原始右操作数
 - 通过短路原则，可以用&&和||来实现复杂的条件语句来代替if-else

&&与||深层次理解

- 实例：&&和||两侧的操作数不是布尔类型时

```
console.log(2&&4);  
console.log(0&&4);  
console.log({x:2}&&{name:"Jame"});  
console.log(null&&"hello");  
console.log({}&&"world");
```

4

0

► Object {name: "Jame"}

null

world

```
console.log(2||4);  
console.log(0||4);  
console.log({x:2}||{name:"Jame"});  
console.log(null||"hello");  
console.log({}||"world");
```

2

4

► Object {x: 2}

hello

► Object {}

内容提纲

- **&&与||的基本理解及应用**
- **&&与||的深层次理解（非布尔类型）**
- **&&与||在实际中的应用**



&&与||在实际中的应用

- 遵循短路特性，使用&&和||可用来实现条件语句

```
var score = 76;  
if(score>90){  
    console.log("优");  
}else if(score>75){  
    console.log("良");  
}else if(score>60){  
    console.log("及格");  
}else{  
    console.log("不及格");  
}
```

//通过&&和||的组合实现如上功能，注：小括号优先级最高

```
console.log((score>90&&"优")||(score>75&&"良")||(score>60&&"及格")||"不及格");
```

&&与||在实际中的应用

• 使用||来设置函数参数的默认值

- 函数定义时可以给参数指定默认值，调用时若未传参数则该参数的值取它定义时的默认值
- JS (ES6之前) 不能直接为函数的参数指定默认值，可以通过 || 来实现

```
var sum = function(a,b,c){  
    b = b||4;  
    c = c||5;  
    return a+b+c;  
}  
console.log(sum(1,2,3)); //1+2+3  
console.log(sum(1,2));  //1+2+5  
console.log(sum(1));    //1+4+5
```

案例中：未传实参的话，形参初始为undefined，undefined转为布尔类型为false，根据||短路原则直接返回右操作数，等效代码如下

```
var sum = function(a,b=4,c=5){  
    return a+b+c;  
}
```

参见实例demo17 前半部分

问题：思考sum(1,0,0)返回多少？ 1还是10

&&与||在实际中的应用

• sum函数的问题及完善

- 若实参转换为布尔类型为false，返回值则可能不是预期结果，如sum(1,0,0)为10
- 增加判断，确保实参转换为布尔类型时为true

```
var sum = function(a,b,c){  
    if(b!=false){b = b||4;} //(b!=false)&&(b=b||4);  
    if(c!=false){c = c||5;} //(c!=false)&&(c=c||5);  
    return a+b+c;  
};  
console.log(sum(1,2,3)); //1+2+3  
console.log(sum(1,2));   //1+2+5  
console.log(sum(1));      //1+4+5  
console.log(sum(1,0,0)); //1+0+0
```

参见实例demo17 后半部分

总结

- **&&与||的基本理解及应用**
- **&&与||的深层次理解（非布尔类型）**
- **&&与||在实际中的应用**



补充

- ! 运算符

! (A&&B) === ! A || ! B

! (A||B) === ! A && ! B

用!!实现布尔类型转换

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

- 使用||实现默认返回值

```
function foo(a,b){  
    return (a+b)||"和不能为0";  
}
```

foo(-2,2);//和不能为0



The background of the slide is decorated with various abstract shapes in shades of green and yellow. These shapes, which include circles, ovals, and teardrop-like forms, are scattered across the top and right sides of the slide, creating a modern and organic feel.

Thank You!



河北师范大学软件学院
Software College of Hebei Normal University

作业

- 复习本章课件及练习
- 查看参考教程相关章节
- 预习JS函数

