赋值运算符

使用 `=` 进行变量赋值

```
let url = 'houdunren.com';

```

#算术运算符

包括以下几种算术运算符。

运算符	说明
*	乘法
/	除法
+	加法
–	减法
%	取余数

```
let a = 5,b = 3;
console.log(a * b); //15
console.log(a % b); //2

```

#复合运算符

可以使用 `*=、/=、+=、-=、%=` 简写算术运算。即 `n*=2` 等同于 `n=n*2`。

```
let n = 2;
n *= 2;
console.log(n);

```

对变量加减相应数值。

```
let n = 2;
n += 3;
console.log(n); //0
n -= 5;
console.log(n); //5

```

`n+=3` 是 `n=n+3` 的简写形式

#一元运算符

#前置操作

前置操作会在表达式最先执行。

```
let n = 1;
++n
console.log(n);
--n
console.log(n);

```

++n 就是 n=n+1 的简写形式。
使用后置操作符，`++n` 会在最先执行，所以 f 的结果是 33。

```
let n = 2;
let f = 30 + ++n;
console.log(f);

```

#后置操作

后置操作会在表达式最后执行。

```
let n = 1;
n++
console.log(n);

```

使用后置操作符，`n++` 会在最后执行，所以 f 的结果是 32。

```
let n = 2;
let f = 30 + n++;
console.log(f);

```

参与数学计算

```
let a = 1;
b = a++ + 2;
console.log(b); //3

```

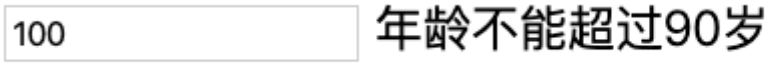
#比较运算符

运算符	说明
>	大于
<	小于
>=	大于或等于
<=	小于等于
==	强制类型转换比较
===	不强制类型转换比较

下面来体验不同类型的比较结果

```
let a = 1,b = 2,c = '1';

console.log(a < b); //true
console.log(a == b); //false
console.log(a == c); //true
console.log(a === c); //false
console.log(a == true); //true
console.log(a === true); //false

```

以下示例不允许年龄超过 90 岁




```
<input type="text" name="age" />
<span id="msg"></span>
<script>
  let span = document.querySelector("#msg");
  document
    .querySelector('[name="age"]')
    .addEventListener("keyup", function() {
      span.innerHTML = this.value >= 90 ? "年龄不能超过90岁" : "";
    });
</script>

```

#逻辑运算符

#逻辑与

使用 `&&` 符号表示逻辑与，指符号两端都为 true 时表达式结果为 true。

```
let a = true,b = true;
if (a && b) {
    console.log('表达式成立');
}

```

#逻辑或

使用 `||` 符号表示逻辑或，指符号左右两端有一方为 true，表达式即成立。

```
let a = true,b = false;
if (a || b) {
    console.log('表达式成立');
}
```

```
5
```

#逻辑非

使用 `!` 符号表示逻辑非，即原来是 true 转变为 false，反之亦然。

```
let a = true,b = false;
if (a && !b) {
    console.log('表达式成立');
}

```

#优先级

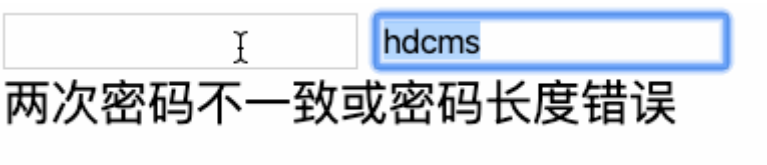
下列中因为 `&&` 的优先级高所以结果是 `true`。

```
console.log(true || false && false);

```

可以使用 `()` 来提高优先级

```
console.log((true || false) && false);

```

#密码比对实例




```
<input type="text" name="password" />
<input type="text" name="confirm_password" />
<br />
<span name="msg"></span>
<script>
  function queryByName(name) {
    return document.querySelector(`[name='${name}']`);
```

```
  }
  let inputs = document.querySelectorAll(
    "[name='password'],[name='confirm_password']"
  );

  [...inputs].map(item => {
    item.addEventListener("keyup", () => {
      let msg = "";
      if (
        queryByName("password").value !=
          queryByName("confirm_password").value ||
        queryByName("password").value.length < 5
      ) {
        msg = "两次密码不一致或密码长度错误";
      }
      queryByName("msg").innerHTML = msg;
    });
  });

```

#短路运算

下例中 `a` 为真值，就已经知道结果了就不会再判断 `f` 的值了。

```
let a = true,f = false;
console.log(a || f);

```

同理当 `f` 值为假时，就已经可以判断 `&&` 的结果了，就没有判断 `a`的必要了。

```
let a = true,f = false;
console.log(f && a);

```

使用短路特性赋值

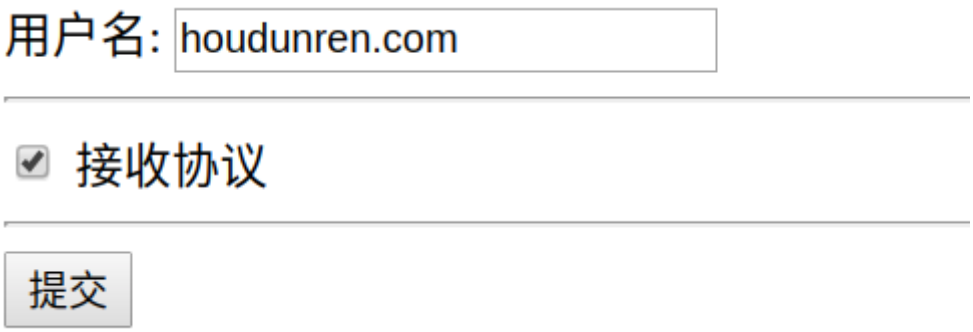
```
let sex = prompt("你的性别是? ") || "保密";
console.log(sex);

```

当 opt.url 没有值时，使用短路特性设置 url 的值

```
let opt = {
    url: ''
};

function getUrl(opt) {
    opt.url = 'houdunren.com';
}
opt.url || getUrl(opt);
console.log(opt.url);

```

#**实例操作**

下面的例子在用户输入表单项并接收协议后才可提交




```
<body>
<form action="
https://www.houdunren.com
<form action="
  用户名: <input type="text" name="username" />
  <hr />
  <input type="checkbox" name="copyright" /> 接收协议
  <hr />
  <input type="submit" />
</form>
</body>
<script>
function query(el) {
  return document.querySelector(el);
```

```
}
query("#form").addEventListener("submit", function(event) {
  let username = query('input[name="username"]').value;
  let copyright = query('input[name="copyright"]').checked;
  console.log(!!username);
  if (!username || copyright === false) {
    alert("请填写用户名并接受协议");
    event.preventDefault();
  }
});
</script>

```

#流程控制

#if

当条件为真时执行表达式代码块。

```
let state = true;
if (true) {
    console.log('表达式成立');
}

```

如果只有一条代码块，可以不用写 `{}`

```
let state = true;
if (true)
    console.log('表达式成立');
console.log('一直都显示的内容');

```

#if/else

下面是使用多条件判断密码强度的示例


```
<body>
  <input type="password" name="title" />
  <span></span>
</body>
<script>
  let input = document.querySelector("[name='title']");
  input.addEventListener("keyup", function() {
    let length = this.value.length;
    let msg;
    if (length > 10) {
      msg = "密码已经无敌了";
    } else if (length > 6) {
      msg = "密码安全性中级";
    } else {
      msg = "这密码，要完的节奏";
    }
    document.querySelector("span").innerHTML = msg;
  });
</script>

```

#三元表达式

是针对 `if` 判断的简写形式。

```
let n = true ? 1 : 2;
console.log(n); //1

let f = true ? (1 == true ? 'yes' : 'no') : 3;
console.log(f); // yes

```

下面是创建 DIV 元素的示例，使用三元表达式设置初始值

```
function div(options = {}) {
  let div = document.createElement("div");
  div.style.width = options.width ? options.width : "100px";
  div.style.height = options.height ? options.height : "100px";
  div.style.backgroundColor = options.bgcolor ? options.bgcolor : "red";
  document.body.appendChild(div);
}
div();

```

#switch

可以将 `switch` 理解为 `if` 的另一种结构清晰的写法。

- 如果表达式等于

case 中的值，将执行此

case 代码段

- break 关键字会终止

switch 的执行

- 没有任何

case匹配时将执行

default 代码块

- 如果

case执行后缺少 break 则接着执行后面的语句

```
let name = '视频';
switch (name) {
    case '产品':
        console.log('hdcms.com');
        break;
    case '视频':
        console.log('houdunren.com');
        break;
    default:
        console.log('houdunwang.com')
}

```

case 合用示例

```
let error = 'warning';
switch (error) {
  case 'notice':
  case 'warning':
      console.log('警告或提示信息');
      break;
  case 'error':
      console.log('错误信息');
}

```

在`switch` 与 `case` 都可以使用表达式

```
function message(age) {
  switch (true) {
    case age < 15:
      console.log("儿童");
      break;
    case age < 25:
      console.log("青少年");
      break;
    case age < 40:
      console.log("青年");
      break;
    case age < 60:
      console.log("中年");
      break;
    case age < 100:
      console.log("老年");
      break;
    default:
      console.log("年龄输出错误");
  }
}
message(10);

```

下面例子缺少 break 后，会接着执行后面的 switch 代码。

```
switch (1) {
  case 1:
    console.log(1);
  case 2:
    console.log(2);
  default:
    console.log("default");
}

```

#while

循环执行语句，需要设置跳出循环的条件否则会陷入死循环状态。下面是循环输出表格的示例。

```
let row = 5;
document.write(`<table border="1" width="100">`);
while (row-- != 0) {
  document.write(`<tr><td>${row}</td></tr>`);
}
document.write(`</table>`);

```

#do/while

后条件判断语句，无论条件是否为真都会先进行循环体。

下面通过循环输出三角形示例，要注意设置循环跳出的时机来避免死循环。

```
*
**
***
****
*****

function hd(row = 5) {
  let start = 0;
  do {
```

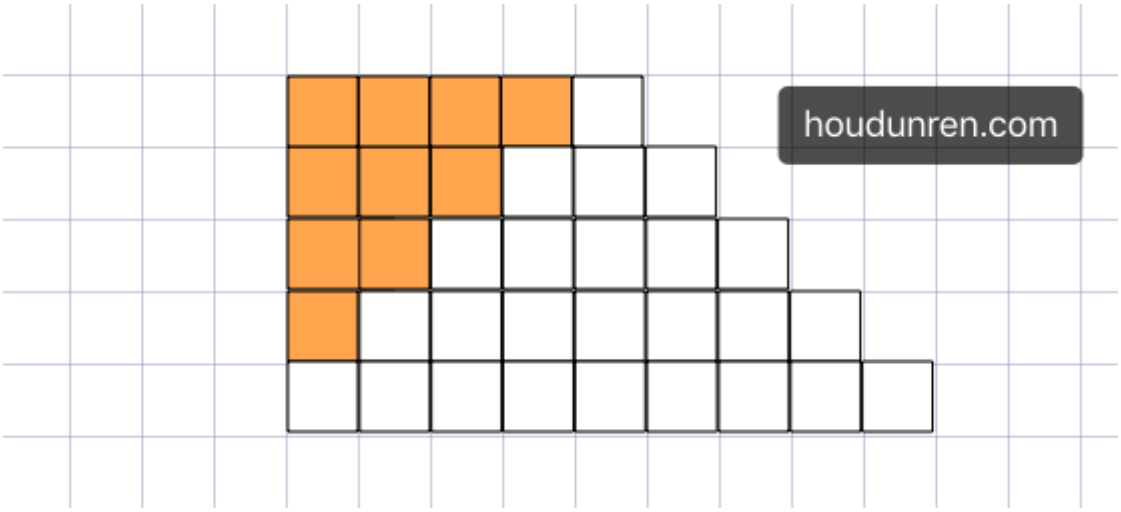
```
    let n = 0;
    do {
      document.write("*");
    } while (++n <= start);
    document.write("<br/>");
  } while (++start <= row);
}
hd();

```

#for

可以在循环前初始化初始计算变量。下面是使用`for`打印倒三角的示例

```
**********
*********
********
*******
******
*****
****
***
**
*

for (let i = 10; i > 0; i--) {
    for (let n = 0; n < i; n++) {
        document.write('*');
    }
    document.write("<br/>");
}

```

下面是使用循环制作杨辉三角的案例




```
    *
   ***
  *****
 *******
*********

for (let i = 1; i <= 5; i++) {
  for (let n = 5 - i; n > 0; n--) {
      document.write('^');
  }
  for (let m = i * 2 - 1; m > 0; m--) {
      document.write('*');
  }
  document.write("<br/>");
}

```

for 的三个参数可以都省略或取几个

```
let i = 1;
for (; i < 10; ) {
  console.log(i++);
}

```

#break/continue

break 用于退出当前循环，continue 用于退出当前循环返回循环起始继续执行。

获取所有偶数，所有奇数使用 `continue` 跳过

```
for (let i = 1; i <= 10; i++) {
  if (i % 2) continue;
  console.log(i);
}

```

获取三个奇数，超过时使用 `break`退出循环

```
let count = 0,num = 3;
for (let i = 1; i <= 10; i++) {
  if (i % 2) {
    console.log(i);
    if (++count == num) break;
  }
}

```

#label

标签(label) 为程序定义位置，可以使用`continue/break`跳到该位置。

下面取`i+n` 大于 15 时退出循环

```
houdunren: for (let i = 1; i <= 10; i++) {
  hdcms: for (let n = 1; n <= 10; n++) {
    if (n % 2 != 0) {
      continue hdcms;
    }
    console.log(i, n);
    if (i + n > 15) {
      break houdunren;
    }
  }
}

```

#for/in

用于遍历对象的所有属性，`for/in`主要用于遍历对象，不建议用来遍历数组。

遍历数组操作

```
let hd = [
  { title: "第一章 走进JAVASCRIPT黑洞", lesson: 3 },
  { title: "ubuntu19.10 配置好用的编程工作站", lesson: 5 },
  { title: "媒体查询响应式布局", lesson: 8 }
];
document.write(`
  <table border="1" width="100%">
  <thead><tr><th>标题</th><th>课程数</th></thead>
`);
for (let key in hd) {
  document.write(`
  <tr>
  <td>${hd[key].title}</td>
  <td>${hd[key].lesson}</td>
  </tr>
  `);
}
document.write("</table>");

```

遍历对象操作

```
let info = {
  name: "后盾人",
  url: "houdunren.com"
};
for (const key in info) {
  if (info.(key)) {
    console.log(info[key]);
  }
}

```

遍历 window 对象的所有属性

```
for (name in window) {
  console.log(window[name]);
}

```

#for/of

用来遍历 Arrays（数组），Strings（字符串），Maps（映射），Sets（集合）等可迭代的数据结构。

与 `for/in` 不同的是 `for/of` 每次循环取其中的值而不是索引。

> 后面在讲到遍历器章节后大家会对 for/of 有更深的体会

```
let arr = [1, 2, 3];
for (const iterator of arr) {
    console.log(iterator);
}

```

遍历字符串

```
let str = 'houdunren';
for (const iterator of str) {
    console.log(iterator);
}

```

使用迭代特性遍历数组（后面章节会介绍迭代器）

```
const hd = ["hdcms", "houdunren"];

for (const [key, value] of hd.entries()) {
  console.log(key, value); //这样就可以遍历了
}

```

使用`for/of` 也可以用来遍历 DOM 元素

```
<body>
  <ul>
```

```
    <li></li>
    <li></li>
  </ul>
</body>
<script>
  let lis = document.querySelectorAll("li");
  for (const li of lis) {
    li.addEventListener("click", function() {
      this.style.backgroundColor = "red";
    });
  }
</script>
```