类型检测

JS 提供了非常丰富的数据类型，开发者要学会使用最适合的数据类型处理业务 。

#typeof

`typeof` 用于返回以下原始类型

- 基本类型：number/string/boolean
- function
- object
- undefined

可以使用 typeof 用于判断数据的类型

```
let a = 1;
console.log(typeof a); //number

let b = "1";
console.log(typeof b); //string

//未赋值或不存在的变量返回undefined
var hd;
console.log(typeof hd);

function run() {}
console.log(typeof run); //function

let c = [1, 2, 3];
console.log(typeof c); //object

let d = { name: "houdunren.com" };
console.log(typeof d); //object

```

#instanceof

`instanceof` 运算符用于检测构造函数的 `prototype` 属性是否出现在某个实例对象的原型链上。

也可以理解为是否为某个对象的实例，`typeof`不能区分数组，但`instanceof`则可以。

> 后面章节会详细介绍原型链

```
let hd = [];
let houdunren = {};
console.log(hd instanceof Array); //true
console.log(houdunren instanceof Array); //false

let c = [1, 2, 3];
console.log(c instanceof Array); //true

let d = { name: "houdunren.com" };
console.log(d instanceof Object); //true

function User() {}
let hd = new User();
console.log(hd instanceof User); //true

```

#值类型与对象

下面是使用字面量与对象方法创建字符串，返回的是不同类型。

```
let hd = "houdunren";
let cms = new String("hdcms");
console.log(typeof hd, typeof cms); //string object

```

只有对象才有方法使用，但在JS中也可以使用值类型调用方法，因为它会在执行时将值类型转为对象。

```
let hd = "houdunren";
let cms = new String("hdcms");
console.log(hd.length); //9
console.log(cms.length); //5

```

#String

字符串类型是使用非常多的数据类型，也是相对简单的数据类型。

#声明定义

使用对象形式创建字符串

```
let hd = new String('houdunren');
// 获取字符串长度
console.log(hd.length);
// 获取字符串
console.log(hd.());

```

字符串使用单、双引号包裹，单、双引号使用结果没有区别。

```
let content = '后盾人';
console.log(content);

```

#转义符号

有些字符有双层含义，需要使用 \ 转义符号进行含义转换。下例中引号为字符串边界符，如果输出引号时需要使用转义符号。

```
let content = '后盾人 \'houduren.com\'';
console.log(content);

```

常用转义符号列表如下

符号	说明
\t	制表符
\n	换行
\\	斜杠符号
\'	单引号
\"	双引号 R

#连接运算符

使用 `+` 可以连接多个内容组合成字符串，经常用于组合输出内容使用。

```
let year = 2010,
name = '后盾人';
console.log(name + '成立于' + year + '年');

```

使用 `+=` 在字符串上追回字符内容

```
let web = '后盾人';
web += '网址: houdunren.com';
console.log(web); //后盾人网址: houdunren.com

```

#模板字面量

使用 ` ...` 符号包裹的字符串中可以写入引入变量与表达式

```
let url = 'houdunren.com';
console.log(`后盾人网址是${url}`); //后盾人网址是houdunren.com

```

支持换行操作不会产生错误

```
let url = 'houdunren.com';
document.write(`后盾人网址是${url}
大家可以在网站上学习到很多技术知识`);

```

使用表达式

```
function show(title) {
 return `后盾人`;
}
console.log(`${show()}`)

```

模板字面量支持嵌套使用


```
let lessons = [
 {title: '媒体查询响应式布局'},{title: 'FLEX 弹性盒模型'},{title: 'GRID 栅格系统'}
];

function template() {
  return `<ul>
      ${lessons.map((item)=>`
          <li>${item.title}</li>
      `).join('')}
  </ul>`;
}
document.body.innerHTML = template();

```

#标签模板

标签模板是提取出普通字符串与变量，交由标签函数处理

```
let lesson = 'css';
let web = '后盾人';
tag `访问${web}学习${lesson}前端知识`;

function tag(strings, ...values) {
    console.log(strings); //["访问", "学习", "前端知识"]
    console.log(values); // ["后盾人", "css"]
}

```

下面例子将标题中有后盾人的使用标签模板加上链接


```
let lessons = [
  { title: "后盾人媒体查询响应式布局", author: "后盾人向军" },
  { title: "FLEX 弹性盒模型", author: "后盾人" },
  { title: "GRID 栅格系统后盾人教程", author: "古老师" }
];

function links(strings, ...vars) {
  return strings
    .map((str, key) => {
      return (
        str +
        (vars[key]
          ? vars[key].replace(
              "后盾人",
              `<a href="
https://www.houdunren.com
              `<a href="
            )
          : "")
      );
    })
    .join("");
}

function template() {
  return `<ul>
    ${lessons
      .map(item => links`<li>${item.author}:${item.title}</li>`)
      .join("")}
</ul>`;
}
document.body.innerHTML += template();
```

```
31
```

#获取长度

使用`length`属性可以获取字符串长度

```
console.log("houdunren.com".length)

```

#大小写转换

将字符转换成大写格式

```
console.log('houdunren.com'.toUpperCase()); //HOUDUNREN.COM

```

转字符为小写格式

```
console.log('houdunren.com'.toLowerCase()); //houdunren.com

```

#移除空白

使用`trim`删除字符串左右的空白字符

```
let str = '   houdunren.com  ';
console.log(str.length);
console.log(str.trim().length);

```

使用`trimLeft`删除左边空白，使用`trimRight`删除右边空白

```
let name = " houdunren ";
console.log(name);
console.log(name.trimLeft());
console.log(name.trimRight());

```

#获取单字符

根据从 0 开始的位置获取字符

```
console.log('houdunren'.charAt(3))

```

使用数字索引获取字符串

```
console.log('houdunren'[3])

```

#截取字符串

使用 `slice、substr、substring` 函数都可以截取字符串。

- slice、substring 第二个参数为截取的结束位置
- substr 第二个参数指定获取字符数量

```
let hd = 'houdunren.com';
console.log(hd.slice(3)); //dunren.com
console.log(hd.substr(3)); //dunren.com
console.log(hd.substring(3)); //dunren.com

console.log(hd.slice(3, 6)); //dun
console.log(hd.substring(3, 6)); //dun
console.log(hd.substring(3, 0)); //hou 较小的做为起始位置
console.log(hd.substr(3, 6)); //dunren

console.log(hd.slice(3, -1)); //dunren.co 第二个为负数表示从后面算的字符
console.log(hd.slice(-2));//om 从末尾取
console.log(hd.substring(3, -9)); //hou 负数转为0
console.log(hd.substr(-3, 2)); //co 从后面第三个开始取两个

```

#查找字符串

从开始获取字符串位置，检测不到时返回 `-1`

```
console.log('houdunren.com'.indexOf('o')); //1
console.log('houdunren.com'.indexOf('o', 3)); //11 从第3个字符向后搜索

```

从结尾来搜索字符串位置

```
console.log('houdunren.com'.lastIndexOf('o')); //11
console.log('houdunren.com'.lastIndexOf('o', 7)); //1 从第7个字符向前搜索

```

search() 方法用于检索字符串中指定的子字符串，也可以使用正则表达式搜索

```
let str = "houdunren.com";
console.log(str.search("com"));
console.log(str.search(/\.com/i));

```

`includes` 字符串中是否包含指定的值，第二个参数指查找开始位置

```
console.log('houdunren.com'.includes('o')); //true
console.log('houdunren.com'.includes('h', 11)); //true

```

`startsWith` 是否是指定位置开始，第二个参数为查找的开始位置。

```
console.log('houdunren.com'.startsWith('h')); //true
console.log('houdunren.com'.startsWith('o', 1)); //true

```

`endsWith` 是否是指定位置结束，第二个参数为查找的结束位置。

```
console.log('houdunren.com'.endsWith('com')); //true
console.log('houdunren.com'.endsWith('o', 2)); //true

```

下面是查找关键词的示例

```

```

```
const words = ["php", "css"];
const title = "我爱在后盾人学习php与css知识";
const status = words.some(word => {
  return title.includes(word);
});
console.log(status);

```

#替换字符串

`replace` 方法用于字符串的替换操作

```
let name = "houdunren.com";
web = name.replace("houdunren", "hdcms");
console.log(web);

```

默认只替换一次，如果全局替换需要使用正则（更强大的使用会在正则表达式章节介绍）

```
let str = "2023/02/12";
console.log(str.replace(/\//g, "-"));

```

使用字符串替换来生成关键词链接

```
const word = ["php", "css"];
const string = "我喜欢在后盾人学习php与css知识";
const title = word.reduce((pre, word) => {
  return pre.replace(word, `<a href="?w=${word}">${word}</a>`);
}, string);
document.body.innerHTML += title;

```

使用正则表达式完成替换

```
let res = "houdunren.com".replace(/u/g, str => {
  return "@";
});
console.log(res);
```

```
5
```

#重复生成

下例是根据参数重复生成星号

```
function star(num = 3) {
 return '*'.repeat(num);
}
console.log(star());

```

下面是模糊后三位电话号码

```
let phone = "98765432101";
console.log(phone.slice(0, -3) + "*".repeat(3));

```

#类型转换

分隔字母

```
let name = "hdcms";
console.log(name.split(""));

```

将字符串转换为数组

```
console.log("1,2,3".split(",")); //[1,2,3]

```

隐式类型转换会根据类型自动转换类型

```
let hd = 99 + '';
console.log(typeof hd); //string

```

使用 `String` 构造函数可以显示转换字符串类型

```
let hd = 99;
console.log(typeof String(hd));

```

js 中大部分类型都是对象，可以使用类方法 `toString`转化为字符串

```
let hd = 99;
console.log(typeof hd.()); //string

let arr = ['hdcms', '后盾人'];
console.log(typeof arr.()); //string

```

#Boolean

布尔类型包括 `true` 与 `false` 两个值，开发中使用较多的数据类型。

#声明定义

使用对象形式创建布尔类型

```
console.log(new Boolean(true)); //true
console.log(new Boolean(false)); //false

```

但建议使用字面量创建布尔类型

```
let hd =true;

```

#隐式转换

基本上所有类型都可以隐式转换为 Boolean 类型。

数据类型	true	false
String	非空字符串	空字符串
Number	非 0 的数值	0 、NaN

Array	数组不参与比较时	参与比较的空数组
Object	所有对象	
undefined	无	undefined
null	无	null
NaN	无	NaN

当与 boolean 类型比较时，会将两边类型统一为数字 1 或 0。

如果使用 Boolean 与数值比较时，会进行隐式类型转换 true 转为 1，false 转为 0。

```
console.log(3 == true); //false
console.log(0 == false); //true

```

下面是一个典型的例子，字符串在与 Boolean 比较时，两边都为转换为数值类型后再进行比较。

```
console.log(Number("houdunren")); //NaN
console.log(Boolean("houdunren")); //true
console.log("houdunren" == true); //false
console.log("1" == true); //true

```

数组的表现与字符串原理一样，会先转换为数值

```
console.log(Number([])); //0
console.log(Number([3])); //3
console.log(Number([1, 2, 3])); //NaN
console.log([] == false); //true
console.log([1] == true); //true
console.log([1, 2, 3] == true); //false

```

引用类型的 Boolean 值为真，如对象和数组

```
if ([]) console.log("true");
if ({}) console.log("true");

```

#显式转换

使用 `!!` 转换布尔类型

```
let hd = '';
console.log(!!hd); //false
hd = 0;
console.log(!!hd); //false
hd = null;
console.log(!!hd); //false
hd = new Date("2020-2-22 10:33");
console.log(!!hd); //true

```

使用 `Boolean` 函数可以显式转换为布尔类型

```
let hd = '';
console.log(Boolean(hd)); //false
hd = 0;
console.log(Boolean(hd)); //false
hd = null;
console.log(Boolean(hd)); //false
hd = new Date("2020-2-22 10:33");
console.log(Boolean(hd)); //true

```

#实例操作

下面使用 Boolean 类型判断用户的输入，并给出不同的反馈。

```
while (true) {
  let n = prompt("请输入后盾人成立年份").trim();
  if (!n) continue;
  alert(n == 2010 ? "回答正确" : "答案错误! 看看官网了解下");
  break;
}

```

#Number

#声明定义

使用对象方式声明

```
let hd = new Number(3);
console.log(hd+3); //6

```

Number 用于表示整数和浮点数，数字是 `Number`实例化的对象，可以使用对象提供的丰富方法。

```
let num = 99;
console.log(typeof num);

```

#基本函数

判断是否为整数

```
console.log(Number.isInteger(1.2));

```

指定返回的小数位数可以四舍五入

```
console.log((16.556).toFixed(2)); // 16.56

```

#NaN

表示无效的数值，下例计算将产生 NaN 结果。

```
console.log(Number("houdunren")); //NaN

console.log(2 / 'houdunren'); //NaN

```

NaN 不能使用 `==` 比较，使用以下代码来判断结果是否正确

```
var res = 2 / 'houdunren';
if (Number.isNaN(res)) {
        console.log('Error');
}

```

也可以使用 `Object.is` 方法判断两个值是否完全相同

```
var res = 2 / 'houdunren';
console.log(Object.is(res, NaN));

```

#类型转换

Number

使用 Number 函数基本上可以转换所有类型

```
console.log(Number('houdunren')); //NaN
console.log(Number(true)); //1
console.log(Number(false)); //0
console.log(Number('9')); //9
console.log(Number([])); //0
console.log(Number([5])); //5
console.log(Number([5, 2])); //NaN
console.log(Number({})); //NaN

```

parseInt

提取字符串开始去除空白后的数字转为整数。

```
console.log(parseInt('  99houdunren')); //99
console.log(parseInt('18.55')); //18

```

parseFloat

转换字符串为浮点数，忽略字符串前面空白字符。

```
console.log(parseFloat('  99houdunren')); //99
```

```
console.log(parseFloat('18.55')); //18.55

```

比如从表单获取的数字是字符串类型需要类型转换才可以计算，下面使用乘法进行隐式类型转换。

```
<input type="text" name="num" value="66">
<script>
  let num = document.querySelector("[name='num']").value;
  console.log(num + 5); //665

  console.log(num * 1 + 5); //71
</script>

```

#舍入操作

使用 `toFixed` 可对数值舍入操作，参数指定保存的小数位

```
console.log(1.556.toFixed(2)); //1.56

```

#浮点精度

大部分编程语言在浮点数计算时都会有精度误差问题，下面来看 JS 中的表现形式

```
let hd = 0.1 + 0.2
console.log(hd)// 结果: 0.30000000000000004

```

这是因为计算机以二进制处理数值类型，上面的 0.1 与 0.2 转为二进制后是无穷的

```
console.log((0.1).(2))
//0.0001100110011001100110011001100110011001100110011001101
console.log((0.2).(2))
//0.0011001100110011001100110011001100110011001100110011001101

```

处理方式

一种方式使用 toFixed 方法进行小数截取

```
console.log((0.1 + 0.2).toFixed(2)) //0.3

console.log(1.0 - 0.9) //0.09999999999999998
console.log((1.0 - 0.9).toFixed(2)) //0.10

```

将小数转为整数进行计算后，再转为小数也可以解决精度问题

```
Number.prototype.add = function (num) {
 //取两个数值中小数位最大的
  let n1 = this.().split('.')[1].length
  let n2 = num.().split('.')[1].length

  //得到10的N次幂
  let m = Math.pow(10, Math.max(n1, n2))

  return (this * m + num * m) / m
}
console.log((0.1).add(0.2))

```

推荐做法

市面上已经存在很多针对数学计算的库 mathjs (opens new window)、decimal.js (opens new window)等，我们就不需要自己构建了。下面来演示使用 decimal.js (opens new window)进行浮点计算。

```
<script src="
https://cdn.bootcss.com/decimal.js/10.2.0/decimal.min.js
<script src="

<script>
        console.log(Decimal.add(0.1, 0.2).valueOf())
</script>

```

#Math

`Math` 对象提供了众多方法用来进行数学计算，下面我们介绍常用的方法，更多方法使用请查看 MDN 官网 (opens new window)了解。

#取极限值

使用 `min` 与 `max` 可以取得最小与最大值。

```
console.log(Math.min(1, 2, 3));

console.log(Math.max(1, 2, 3));

```

使用`apply` 来从数组中取值

```
console.log(Math.max.apply(Math, [1, 2, 3]));

```

#舍入处理

取最接近的向上整数

```
console.log(Math.ceil(1.111)); //2

```

得到最接近的向下整数

```
console.log(Math.floor(1.555)); //1

```

四舍五入处理

```
console.log(Math.round(1.5)); //2

```

#random

`random` 方法用于返回 >=0 且 <1 的随机数（包括 0 但不包括 1）。
返回 0~5 的随机数，不包括 5

```
const number = Math.floor(Math.random() * 5);
console.log(number);
```

```
3
```

返回 0~5 的随机数，包括 5

```
const number = Math.floor(Math.random() * (5+1));
console.log(number);
```

下面取 2~5 的随机数（不包括 5）公式为：min+Math.floor(Math.random()*(Max-min))

```
const number = Math.floor(Math.random() * (5 - 2)) + 2;
console.log(number);
```

下面取 2~5 的随机数（包括 5）公式为：min+Math.floor(Math.random()*(Max-min+1))

```
const number = Math.floor(Math.random() * (5 - 2 + 1)) + 2;
console.log(number);
```

下面是随机点名的示例

```
let stus = ['小明', '张三', '王五', '爱情'];
let pos = Math.floor(Math.random() * stus.length);
console.log(stus[pos]);
```

随机取第二到第三间的学生，即 1~2 的值

```
let stus = ['小明', '张三', '王五', '爱情'];
let pos = Math.floor(Math.random() * (3-1)) + 1;
console.log(stus[pos]);
```

#Date

网站中处理日期时间是很常用的功能，通过 `Date` 类型提供的丰富功能可以非常方便的操作。

#声明日期

获取当前日期时间

```
let now = new Date();
console.log(now);
console.log(typeof date); //object
console.log(now * 1); //获取时间戳

//直接使用函数获取当前时间
console.log(Date());
console.log(typeof Date()); //string

//获取当前时间戳单位毫秒
console.log(Date.now());

```

计算脚本执行时间

```
const start = Date.now();
for (let i = 0; i < 2000000; i++) {}
const end = Date.now();
console.log(end - start);

```

当然也可以使用控制台测试

```
console.time("testFor");
for (let i = 0; i < 20000000; i++) {}
console.timeEnd("testFor");

```

根据指定的日期与时间定义日期对象

```
let now = new Date('2028-02-22 03:25:02');
console.log(now);

now = new Date(2028, 4, 5, 1, 22, 16);
console.log(now);
```

```
6
```

使用展示运算符处理更方便

```
let info = [2020, 2, 20, 10, 15, 32];
let date = new Date(...info);
console.dir(date);

```

#类型转换

将日期转为数值类型就是转为时间戳单位是毫秒

```
let hd = new Date("2020-2-22 10:33:12");
console.log(hd * 1);

console.log(Number(hd));

console.log(hd.())

console.log(date.getTime());

```

有时后台提供的日期为时间戳格式，下面是将时间戳转换为标准日期的方法

```
const param = [1990, 2, 22, 13, 22, 19];
const date = new Date(...param);
const timestamp = date.getTime();
console.log(timestamp);
console.log(new Date(timestamp));

```

#对象方法

格式化输出日期

```
let time = new Date();
console.log(
```

```
  `${time.getFullYear()}-${time.getMonth()}-${time.getDate()}
${time.getHours()}:${time.getMinutes()}:${time.getSeconds()}`
);

```

封装函数用于复用

```
function dateFormat(date, format = "YYYY-MM-DD HH:mm:ss") {
  const config = {
    YYYY: date.getFullYear(),
    MM: date.getMonth() + 1,
    DD: date.getDate(),
    HH: date.getHours(),
    mm: date.getMinutes(),
    ss: date.getSeconds()
  };
  for (const key in config) {
    format = format.replace(key, config[key]);
  }
  return format;
}
console.log(dateFormat(new Date(), "YYYY年MM月DD日"));

```

下面是系统提供的日期时间方法，更多方法请查看 MDN 官网(opens new window)

方法	描述
Date()	返回当日的日期和时间。
getDate()	从 Date 对象返回一个月中的某一天 (1 ~ 31)。
getDay()	从 Date 对象返回一周中的某一天 (0 ~ 6)。
getMonth()	从 Date 对象返回月份 (0 ~ 11)。
getFullYear()	从 Date 对象以四位数字返回年份。
getYear()	请使用 getFullYear() 方法代替。
getHours()	返回 Date 对象的小时 (0 ~ 23)。
getMinutes()	返回 Date 对象的分钟 (0 ~ 59)。
getSeconds()	返回 Date 对象的秒数 (0 ~ 59)。

getMilliseconds()	返回 Date 对象的毫秒(0 ~ 999)。
getTime()	返回 1970 年 1 月 1 日至今的毫秒数。
getTimezoneOffset()	返回本地时间与格林威治标准时间 (GMT) 的分钟差。
getUTCDate()	根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。
getUTCDay()	根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。
getUTCMonth()	根据世界时从 Date 对象返回月份 (0 ~ 11)。
getUTCFullYear()	根据世界时从 Date 对象返回四位数的年份。
getUTCHours()	根据世界时返回 Date 对象的小时 (0 ~ 23)。
getUTCMinutes()	根据世界时返回 Date 对象的分钟 (0 ~ 59)。
getUTCSeconds()	根据世界时返回 Date 对象的秒钟 (0 ~ 59)。
getUTCMilliseconds()	根据世界时返回 Date 对象的毫秒(0 ~ 999)。
parse()	返回 1970 年 1 月 1 日午夜到指定日期（字符串）的毫秒数。
setDate()	设置 Date 对象中月的某一天 (1 ~ 31)。
setMonth()	设置 Date 对象中月份 (0 ~ 11)。
setFullYear()	设置 Date 对象中的年份（四位数字）。
setYear()	请使用 setFullYear() 方法代替。
setHours()	设置 Date 对象中的小时 (0 ~ 23)。
setMinutes()	设置 Date 对象中的分钟 (0 ~ 59)。
setSeconds()	设置 Date 对象中的秒钟 (0 ~ 59)。
setMilliseconds()	设置 Date 对象中的毫秒 (0 ~ 999)。
setTime()	以毫秒设置 Date 对象。
setUTCDate()	根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。
setUTCMonth()	根据世界时设置 Date 对象中的月份 (0 ~ 11)。
setUTCFullYear()	根据世界时设置 Date 对象中的年份（四位数

	字）。
setUTCHours()	根据世界时设置 Date 对象中的小时 (0 ~ 23)。
setUTCMinutes()	根据世界时设置 Date 对象中的分钟 (0 ~ 59)。
setUTCSeconds()	根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。
setUTCMilliseconds()	根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。
toSource()	返回该对象的源代码。
toString()	把 Date 对象转换为字符串。
toTimeString()	把 Date 对象的时间部分转换为字符串。
toDateString()	把 Date 对象的日期部分转换为字符串。
toGMTString()	请使用 toUTCString() 方法代替。
toUTCString()	根据世界时，把 Date 对象转换为字符串。
toLocaleString()	根据本地时间格式，把 Date 对象转换为字符串。
toLocaleTimeString()	根据本地时间格式，把 Date 对象的时间部分转换为字符串。
toLocaleDateString()	根据本地时间格式，把 Date 对象的日期部分转换为字符串。
UTC()	根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。
valueOf()	返回 Date 对象的原始值。

#moment.js

Moment.js 是一个轻量级的 JavaScript 时间库，它方便了日常开发中对时间的操作，提高了开发效率。

更多使用方法请访问中文官网 http://momentjs.cn (opens new window)或 英文官网 https://momentjs.com(opens new window)

```
1 <script src="
  https://cdn.bootcss.com/moment.js/2.24.0/moment.min.js
  <script src="
2
```

获取当前时间

```
1 console.log(moment().format("YYYY-MM-DD HH:mm:ss"));
2
```

设置时间

```
1 console.log(moment("2020-02-18 09:22:15").format("YYYY-MM-DD HH:mm:ss"));
2
```

十天后的日期

```
1 console.log(moment().add(10, "days").format("YYYY-MM-DD hh:mm:ss"));
```