前言

变量的数据类型转换：将一种数据类型转换为另外一种数据类型。

通常有三种形式的类型转换：

- 转换为字符串类型
- 转换为数字型
- 转换为布尔型

你会专门把某个数据类型转换成 null 或者 undefined 吗？不会，因为这样做，没有意义。

typeof 运算符

> 我们先来讲一下 typeof，再讲类型转换。

`typeof()` 表示“**获取变量的数据类型**”，返回的是小写，语法为：（两种写法都可以）

```
// 写法1
typeof 变量;

// 写法2
typeof(变量);
```

typeof 这个运算符的返回结果就是变量的类型。那返回结果的类型是什么呢？是字符串。

返回结果：

typeof 的代码写法	返回结果
typeof 数字	number
typeof 字符串	string
typeof 布尔型	boolean
typeof 对象	object
typeof 方法	function
typeof null	object
typeof undefined	undefined

备注 1：为啥 `typeof null` 的返回值也是 object 呢？因为 null 代表的是**空对象**。

备注 2：`typeof NaN` 的返回值是 number，上一篇文章中讲过，NaN 是一个特殊的数字。

返回结果举例：

```
console.log(type []); // 空数组的打印结果：object

console.log(type {}); // 空对象的打印结果：object
```

代码解释：这里的空数组`[]`、空对象`{}`，为啥他们在使用 typeof 时，返回值也是`object`呢？因为这里的 返回结果`object`指的是**引用数据类型**。空数组、空对象都是**引用数据类型 Object**。

变量的类型转换的分类

类型转换分为两种：显示类型转换、隐式类型转换。

显示类型转换

- toString()
- String()
- Number()
- parseInt(string)
- parseFloat(string)
- Boolean()

隐式类型转换

- isNaN ()
- 自增/自减运算符：`++`、`--`
- 正号/负号：`+a`、`-a`
- 加号：`+`
- 运算符：`-`、`*`、`/`

隐式类型转换（特殊）

- 逻辑运算符：`&&`、`||`、`!`。非布尔值进行**与或**运算时，会先将其转换为布尔值，然后再运算，但运算结果是**原值**。具体可以看下一篇文章《运算符》。
- 关系运算符：`<`、`>``<=``>=`等。关系运算符，得到的运算结果都是布尔值：要么是true，要么是false。具体可以看下一篇文章《运算符》。

针对上面这两种类型转换，这篇文章来详细介绍。

其他的简单类型 --> String

方法一（隐式类型转换）：字符串拼接

格式：变量+"" 或者 变量+"abc"

举例：

```
var a = 123; // Number 类型
console.log(a + ''); // 转换成 String 类型
console.log(a + 'haha'); // 转换成 String 类型
```

上面的例子中，打印的结果，都是字符串类型的数据。实际上内部是调用的 String() 函数。也就是说，`c = c + ""` 等价于 `c = String(c)`。

方法二：调用 toString()方法

语法：

```
变量.toString()
```

【重要】该方法**不会影响到原变量**，它会将转换的结果返回。当然我们还可以直接写成`a = a.toString()`，这样的话，就是直接修改原变量。

注意：null 和 undefined 这两个值没有 toString()方法，所以它们不能用方法二。如果调用，会报错。

另外，Number 类型的变量，在调用 toString()时，可以在方法中传递一个整数作为参数。此时它将会把数字转换为指定的进制，如果不指定则默认转换为 10 进制。例如：

```
var a = 255;

//对于Number调用toString()时可以在方法中传递一个整数作为参数
//此时它将会把数字转换为指定的进制,如果不指定则默认转换为10进制
a = a.toString(2); // 转换为二进制

console.log(a); // 11111111
console.log(typeof a); // string
```

方法三（强制转换）：使用 String()函数

语法：

```
String(变量)
```

使用 String()函数做强制类型转换时：

- 对于 Number 和 Boolean 而言，本质上就是调用 toString()方法。
- 但是对于 null 和 undefined，则不会调用 toString()方法。它会将 null 直接转换为 "null"。将 undefined 直接转换为 "undefined"。

prompt()：用户的输入

我们在 JS 基础的第 01 篇里，就讲过，`prompt()`就是专门用来弹出能够让用户输入的对话框。重要的是：用户不管输入什么，都当字符串处理。

其他的数据类型 --> Number 【重要】

使用 Number() 函数

情况一：字符串 --> 数字

- 1.如果字符串中是纯数字，则直接将其转换为数字。
- 2.只要字符串中包含了非数字的内容（`小数点`按数字来算），则转换为 NaN。
- 3.如果字符串是一个空串或者是一个全是空格的字符串，则转换为 0。

情况二：布尔 --> 数字

- true 转成 1
- false 转成 0

情况三：null --> 数字

- 结果为：0

情况四：undefined --> 数字

- 结果为：NaN

补充：怎么理解这里的 **NaN** 呢？可以这样理解，使用 Number() 函数之后，**如果无法转换为数字，就会转换为 NaN。**

使用 parseInt()函数：字符串 -> 整数

parseInt()的作用是将字符串中的有效的整数内容转为数字。parse 表示“转换”，Int 表示“整数”（注意`Int`的拼写）。例如：

```
parseInt("5");
```

得到的结果是数字 5。

parseInt()的转换情况如下。

情况一：字符串 --> 数字

- 1.**只保留字符串最开头的数字**，后面的中文自动消失。
- 2.如果字符串不是以数字开头，则转换为 NaN。
- 3.如果字符串是一个空串或者是一个全是空格的字符串，转换时会报错。

情况二：Boolean --> 数字

- 结果为：NaN

情况三：Null --> 数字

- 结果为：NaN

情况四：Undefined --> 数字

- 结果为：NaN

Number() 函数和 parseInt() 函数的区别：

就拿`Number(true)`和`parseInt(true)/parseFloat(true)`来举例，二者在使用时，是有区别的：

- Number(true) ：千方百计地想转换为数字。
- parseInt(true)/parseFloat(true) ：先转为字符串，再提取出最前面的数字部分；没提取出来，那就返回 NaN。

parseInt()具有以下特性：

（1）**只保留字符串最开头的数字**，后面的中文自动消失。例如：

```
console.log(parseInt("2017在公众号上写了6篇文章")); //打印结果：2017

console.log(parseInt("2017.01在公众号上写了6篇文章")); //打印结果仍是：2017    （说明只会取整数）

console.log(parseInt("aaa2017.01在公众号上写了6篇文章")); //打印结果：NaN （因为不是以数字开头）
```

（2）如果对**非 String**使用 parseInt()或 parseFloat()，它会**先将其转换为 String** 然后再操作。【重要】

比如：

```
var a = 168.23;
console.log(parseInt(a)); //打印结果：168  （因为是先将c转为字符串"168.23"，然后然后再操作）

var b = true;
console.log(parseInt(b)); //打印结果：NaN （因为是先将a转为字符串"true"，然后然后再操作）

var c = null;
console.log(parseInt(c)); //打印结果：NaN  （因为是先将b转为字符串"null"，然后然后再操作）

var d = undefined;
console.log(parseInt(d)); //打印结果：NaN  （因为是先将b转为字符串"undefined"，然后然后再操作）
```

（3）自动带有截断小数的功能：**取整，不四舍五入。**

例 1：

```
var a = parseInt(5.8) + parseInt(4.7);
console.log(a);
```

打印结果：

```
9
```

例 2：

```
var a = parseInt(5.8 + 4.7);
console.log(a);
```

打印结果：

```
10;
```

（4）带两个参数时，表示在转换时，包含了进制转换。

代码举例：

```
var a = '110';

var num = parseInt(a, 16); // 【重要】将 a 当成 十六进制 来看待，转换成 十进制 的 num

console.log(num);
```

打印结果：

```
272
```

如果你对打印结果感到震惊，请仔细看上面的代码注释。就是说，无论 parseInt() 里面的进制参数是多少，最终的转换结果是十进制。

我们继续来看下面的代码，打印结果是多少。

```
var a = '5';

var num = parseInt(a, 2); // 将 a 当成 二进制 来看待，转换成 十进制 的 num

console.log(num); // 打印结果：NaN。因为 二进制中没有 5 这个数，转换失败。
```

parseFloat()函数：字符串 --> 浮点数（小数）

parseFloat()的作用是：将字符串转换为**浮点数**。

parseFloat()和 parseInt()的作用类似，不同的是，parseFloat()可以获得有效的小数部分。

代码举例：

```
var a = '123.456.789px';
console.log(parseFloat(a)); // 打印结果：123.456
```

parseFloat() 的几个特性，可以参照 parseInt()。

转换为 Boolean

将其他的数据类型转换为 Boolean，可以使用 Boolean()函数。情况如下：

- 情况一：数字 --> 布尔。除了 0 和 NaN，其余的都是 true。也就是说，`Boolean(NaN)`的结果是 false。
- 情况二：字符串 ---> 布尔。除了空串，其余的都是 true。全是空格的字符串，转换结果也是 true。字符串`'0'`的转换结果也是 true。
- 情况三：null 和 undefined 都会转换为 false。
- 情况四：引用数据类型会转换为 true。注意，空数组`[]`和空对象`{}`，转换结果也是 true，这个一点，很多人都不知道。

PS：转换为 Boolean 的这几种情况，**很重要**，开发中会经常用到。

知识补充：其他进制的数字

- 16 进制的数字，以`0x`开头
- 8 进制的数字，以`0`开头
- 2 进制的数字，`0b`开头（不是所有的浏览器都支持：chrome 和火狐支持，IE 不支持）

比如`070`这个字符串，如果我调用 parseInt()转成数字时，有些浏览器会当成 8 进制解析，有些会当成 10 进制解析。

所以，比较建议的做法是：可以在 parseInt()中传递第二个参数，来指定当前数字的进制。例如：

```
var a = "070";

a = parseInt(a, 8); //将 070 当成八进制来看待，转换结果为十进制。
console.log(a); // 打印结果：56。这个地方要好好理解。
```

隐式类型转换

重点：**隐式类型转换，内部调用的都是显式类型的方法。**下面来详细介绍。

isNaN() 函数

语法：

```
isNaN(参数);
```

解释：判断指定的参数是否为 NaN（非数字类型），返回结果为 Boolean 类型。也就是说：**任何不能被转换为数值的参数，都会让这个函数返回 true。**

执行过程：

（1）先调用`Number(参数)`函数；

（2）然后将`Number(参数)`的返回结果和`NaN`进行比较。

代码举例：

```
console.log(isNaN('123')); // 返回结果：false。

console.log(isNaN('abc')); // 返回结果：true。因为 Number('abc') 的返回结果是 NaN

console.log(isNaN(null)); // 返回结果：false

console.log(isNaN(undefined)); // 返回结果：true

console.log(isNaN(NaN)); // 返回结果：true
```

自增/自减运算符：`++`、`--`

举例 1：

```
var a = "666";
a++;

console.log(typeof a); // 打印结果：  number
console.log(a); // 打印结果：667
```

执行过程：

（1）先调用`Number(参数)`函数；

（2）然后将`Number(参数)`的返回结果进行 加 1 操作。

举例 2：

```
var a = 'abc';
a++;
console.log(typeof a); // 打印结果：number
console.log(a); // 打印结果：NaN。因为 Number('abc')的结果为 NaN，再自增后，结果依然是 NaN
```

正号/负号：`+a`、`-a`

> 注意，这里说的是正号/负号，不是加号/减号。

任何值做`+a`、`-a`、`/a`运算时，运算结果都会自动转换为 Number 类型。 内部调用的是 Number() 函数。

举例：

```
var a = '666';
var b = +a;

console.log(typeof a); // 打印结果：string。说明 a 的数据类型保持不变。
console.log(a); // 打印结果：666

console.log(typeof b); // 打印结果：number。说明 b 的数据类型发生了变化。
console.log(b); // 打印结果：666
```

加号：`+`

情况一：字符串 + 数字

- 当加号的两边，只要有一个是字符串的时候，就会调用 String() 函数将数字转为字符串，然后再计算。导致最终的运算结果是字符串。

情况二：Boolean + 数字

- Boolean 型和数字型相加时， true 按 1 来算 ，false 按 0 来算。这里其实是先调 Number() 函数，将 Boolean 类型转换为 Number类型，然后再和 数字相加。

情况三： null + 数字

- 等价于：0 + 数字

情况四： undefined + 数字

- 计算结果：NaN

运算符：`-`、`*`、`/`

1、任何非 Number 类型的值做`-`、`*`、`/`运算时，会将这些值转换为Number然后再运算(内部调用的是 Number() 函数 ），运算结果是 Number 类型。（注：`任何值 + 字符串`是特例，运算结果是字符串 ）

比如：

```
    result1 = true + 1;  // 2 = 1+ 1

    result2 = true + false; // 1 = 1+ 0

    result3 = 1 + null; // 1 = 1+ 0

    result4 = 100 - '1' // 99
```

2、任何的值和字符串做加法运算，都会先转换为字符串，然后再做拼串操作。

比如：

```
    result1 = 1 + 2 + '3'  // 33

    result2 = '1' + 2 + 3; // 123
```

3、任何值和NaN做运算的结果都是NaN。