数据类型转换

1. 强制转换

使用 Number()、String()、Boolean() 三个函数将各种类型的值转换成数值、字符串、布尔值。

1.1 `Number()`

使用 `Number()` 函数，可以将任意类型的值转换成数值。分两种情况进行讨论，参数是原始类型或参数是复合类型。

1.1.1 原始类型

```
Number(324);        // 324  // 数值：转换后还是原来的值

Number("324");      // 324  // 字符串：如果可以被解析为数值，则转换为相应的数值
Number("324abc");   // NaN  // 字符串：如果不可以被解析为数值，返回 NaN
Number("");         // 0  // 空字符串转为 0

Number(true);       // 1  // 布尔值：true 转成 1
Number(false);      // 0  // 布尔值：false 转成 0

Number(undefined); // NaN  // undefined：转成 NaN

Number(null);       // 0 // null：转成0
```

`Number()` 函数将字符串转为数值，要比 `parseInt()` 函数严格很多。基本上，只要有一个字符无法转成数值，整个字符串就会被转为 `NaN`。`parseInt()` 逐个解析字符，而 `Number()` 函数整体转换字符串的类型。

```
parseInt("42px"); // 42
Number("42px");   // NaN
```

`parseInt()` 和 `Number()` 函数都会自动过滤一个字符串前导和后缀的空格。

```
parseInt("\t\v\r12.34\n"); // 12
Number("\t\v\r12.34\n");   // 12.34
```

但是如果中间有空格的话，`Number()` 函数还是返回 `NaN`。

```
Number("12 34");   // NaN
parseInt("12 34"); // 12
```

因为空格无法转换为数值，所以 `Number()` 函数整体无法转换。`parseInt()` 函数可以转换空格前面的字符。

1.1.2 对象

简单的规则是，`Number()` 方法的参数是对象时，将返回 `NaN`，除非是包含单个数值的数组。

```
Number({ a: 1 });  // NaN
Number([1, 2, 3]); // NaN
Number([5]);       // 5
```

`Number()` 背后的转换规则：

> 1. 调用对象自身的 `valueOf()` 方法。如果返回原始类型的值，则直接对该值使用 `Number()` 函数，不再进行后续步骤。
> 2. 如果 `valueOf()` 方法返回的还是对象，则改为调用对象自身的 `toString()` 方法。如果 `toString()` 方法返回原始类型的值，则对该值使用 `Number()` 函数，不再进行后续步骤。
> 3. 如果 `toString` 方法返回的是对象，就报错。

```
let o = { x: 1 };
if (typeof o.valueOf() === "object") {
  if (typeof o.toString() === "object") {
    console.log("error", new Error("error"));
  } else {
    console.log("typeof", typeof o.toString());
    console.log("toString", o.toString());
    console.log("Number-toString", Number(o.toString()));
  }
} else {
  console.log("Number-valueOf", Number(o.valueOf()));
}
// typeof string
// toString "[object Object]"
// Number-toString NaN
```

上面代码中，`Number` 函数将 `o` 对象转为数值。首先调用 `o.valueOf()` 方法，结果返回对象本身；于是，继续调用 `o.toString()` 方法，这时返回字符串`[object Object]`，对这个字符串使用 `Number` 函数，得到 `NaN`。

如果 `toString` 方法返回的不是原始类型的值，结果就会报错。

```
let obj = {
  valueOf: function () {
    console.log("valueOf");
    return {};
  },
  toString: function () {
    console.log("toString");
    return {};
```

```
  },
};
Number(obj);
// valueOf
// toString
// TypeError: Cannot convert object to primitive value
```

由以上输出结果可知，先调用了 `obj` 对象的 `valueOf()` 方法，再调用了 `obj` 对象的 `toString()` 方法。**`valueOf()` 和 `toString()` 都是可以自定义的。**

```
Number({
  valueOf: function () { return 2; }
});
// 2，返回 `valueOf()` 方法的值

Number({
  toString: function () { return 3; }
});
// 3，返回 `toString()` 方法的值

Number({
  valueOf: function () { return 2; },
  toString: function () { return 3; }
});
// 2，`valueOf()` 方法先于 `toString()` 方法执行
```

1.2 `String()`

`String()` 可以将任意类型的值转换成字符串。

1.2.1 原始类型值

- 数值：转为相应的字符串。
- 字符串：转换后还是原来的值。
- 布尔值：`true` 转为字符串 `"true"`，`false` 转为字符串 `"false"`。
- `undefined`：转为字符串 `"undefined"`。
- `null`：转为字符串 `"null"`。

```
String(123);       // "123"
String(NaN);       // "NaN"
String("abc");     // "abc"
String(" abc ");   // " abc "
String(true);      // "true"
String(false);     // "false"
String(undefined); // "undefined"
String(null);      // "null"
```

1.2.2 对象

`String()` 方法的参数如果是对象，返回一个类型字符串；如果是数组，返回该数组的字符串形式；如果是函数，返回函数本身。

```
String({ a: 1 });         // "[object Object]"
String([1, 2, 3]);        // "1,2,3"
String(function f() {}); // "function f() {}"
```

`String()` 方法背后的转换规则，与 `Number()` 方法基本相同，只是互换了 `valueOf()` 方法和 `toString()` 方法的执行顺序：

1. 先调用对象自身的 `toString()` 方法。如果返回原始类型的值，则对该值使用 `String()` 函数，不再进行以下步骤。
2. 如果 `toString()` 方法返回的是对象，再调用原对象的 `valueOf()` 方法。如果 `valueOf()` 方法返回原始类型的值，则对该值使用 `String()` 函数，不再进行以下步骤。
3. 如果 `valueOf()` 方法返回的是对象，就报错。

如果 `toString()` 方法和 `valueOf()` 方法，返回的都是对象，就会报错。

```
let obj = {
  valueOf: function () {
    console.log("valueOf");
    return {};
  },
  toString: function () {
    console.log("toString");
    return {};
  },
};

String(obj);
// toString
// valueOf
// Uncaught TypeError: Cannot convert object to primitive value
```

由以上输出结果可知，先调用了 `obj` 对象的 `toString()` 方法，再调用了 `obj` 对象的 `valueOf()` 方法。

```
String({
  toString: function () {
    return 3;
  },
});
// "3"，返回 toString() 方法的值（数值 3）

String({
  valueOf: function () {
```

```
    return 2;
  },
});
// "[object Object]", 调用对象的 toString() 方法，返回 "[object Object]"

String({
  valueOf: function () {
    return 2;
  },
  toString: function () {
    return 3;
  },
});
// "3", toString() 方法先于 valueOf() 方法执行
```

1.3 Boolean()

`Boolean()` 函数可以将任意类型的值转为布尔值。 除了以下六个值的转换结果为 `false`，其他的值全部为 `true`。

```
Boolean(undefined); // false
Boolean(null);      // false
Boolean(0);         // false 包括 -0
Boolean(NaN);       // false
Boolean("");        // false
Boolean(false);     // false
```

所有对象（包括空对象）的转换结果都是 `true`，甚至连 `false` 对应的布尔对象 `new Boolean(false)` 也是 `true`。

```
Boolean({});                // true
Boolean([]);                // true
Boolean(new Boolean(false)); // true
Boolean(function f() {});    // true
```

所有对象的布尔值都是 `true`，这是因为 JavaScript 语言设计的时候，出于性能的考虑，如果对象需要计算才能得到布尔值，对于 `obj1 && obj2` 这样的场景，可能会需要较多的计算。为了保证性能，就统一规定，对象的布尔值为 `true`。

2. 自动转换

自动转换是以强制转换为基础的。有三种情况，JavaScript 会自动转换数据类型。

（1）不同类型的数据相互运算

```
123 + "abc"; // "123abc"
```

（2）对非布尔值类型的数据求布尔值

```
const a = 0;
a ? "hello" : "world"; // "world"
```

（3）对非数值类型使用一元运算符

```
+"1";   // 1
+"[1]"; // NaN
```

自动转换规则是这样的：预期什么类型的值，就调用该类型的转换函数。比如，某个位置预期是字符串，就调用 `String()` 函数进行转换。如果该位置既可以是字符串，又可以是数值，就默认转换为数值。

由于自动转换具有不确定性，而且不易排查错误，在预期为布尔值、数值、字符串的地方，全部使用 `Boolean()`、`Number()`和 `String()`函数进行显式转换。

2.1 自动转换为布尔值

JavaScript 预期为布尔值的地方，就会将其转换为布尔值。系统内部会自动调用 `Boolean` 函数。比如在使用 `if` 语句，三元运算符 `?:`，取反运算 `!` 时，将以下六个值转换为 `false`，其余转换为 `true`。

```
Boolean(undefined); // false
Boolean(null);      // false
Boolean(0);         // false 包括 -0
Boolean(NaN);       // false
Boolean("");        // false
Boolean(false);     // false
```

2.2 自动转换为字符串

JavaScript 遇到预期为字符串的地方，就会将非字符串的值自动转为字符串。具体规则是，先将复合类型的值转为原始类型的值，再将原始类型的值转为字符串。

字符串的自动转换，主要发生在字符串的加法运算时。当一个值为字符串，另一个值为非字符串，则后者转为字符串。

```
"5" + 1;              // "51"
"5" + true;           // "5true"
"5" + false;          // "5false"
"5" + {};             // "5[object Object]"
"5" + [];             // "5" [] 转换为字符串结果是空字符串
"5" + function () {}; // "5function (){}" 函数转换为字符串结果是函数本身
"5" + undefined;      // "5undefined"
"5" + null;           // "5null"
```

2.3 自动转换为数值

JavaScript 遇到预期为数值的地方，就会将参数值自动转换为数值。系统内部会自动调用 `Number()` 函数。

除了加法运算符（+）有可能把运算子转为字符串，其他运算符，例如-,*,/,都会把运算子自动转成数值。

```
"5" - "2";     // 3
"5" * "2";     // 10
true - 1;      // 0
false - 1;     // -1
"1" - 1;       // 0
"5" * [];      // 0  Number([]) === 0
false / "5";   // 0
"abc" - 1;     // NaN  Number("abc") === NaN, `NaN` 与任何数（包括它自己）的运算，得到的都是 `NaN`。
null + 1;      // 1  Number(null) === 0
undefined + 1; // NaN  Number(undefined) === NaN
```

一元运算符也会把运算子转换为数值。

```
+"0";  // 0
+"a";  // NaN
-"  "; // -0
-"\n"; // -0
-"\t"; // -0
```

空字符串，空格，制表符、换行符进行一元运算符转换，都会得到数值类型，值为 `0`。换行符 `\n` 制表符 `\t` 都是空白字符。

```
"\n" === "\u000A"; // true
"\t" === "\u0009"; // true
"\0" === "\u0000"; // true
```