

The background of the image is a dark, atmospheric forest. The trees are tall and thin, their trunks reaching upwards. In the foreground, there are fallen branches and debris, creating a sense of depth and mystery. The lighting is low, with some highlights on the tree trunks, suggesting a dimly lit environment like a forest at night or in heavy fog.

如何读懂 TC39

# 目录

- TC39 是什么
- 为什么要读规范
- 发版流程
- ECMA-262
- ECMA-402
- ECMA-404
- ECMA-414

# TC39 是什么

揭开 TC39 的神秘面纱

是一个由 **JavaScript** 开发者、实现者、学者等组成的团体，与 **JavaScript** 社区合作维护和发展 **JavaScript** 的标准。

TC39 包含四个规范：

- ECMA-262: ECMAScript
- ECMA-402: 国际化 API
- ECMA-404: JSON
- ECMA-414: 规定哪些规范是与 ECMAScript 有关的，目前只有 262、402 和 404

# 为什么需要读规范

碰到稀奇古怪的问题怎么办

加法运算符你知道多少？

```
□ + {} // '[object Object]'  
□ + □ // ''  
{ } + □ // 0  
{ } + { } // NaN  
□ + {} == { } + □ // true  
{ } + □ != □ + { } // true
```

为什么有些方法在原型上工作，有些不行？

```
> Array.prototype.push(42)
1
> Array.prototype
[ 42 ]
> Array.isArray(Array.prototype)
true
> Set.prototype.add(42)
TypeError: Method Set.prototype.add called on incompatible receiver #<Set>
    at Set.add (<anonymous>)
> Set.prototype
Set {}
```

# 发版流程

一个提案的一生

从 ES5 到 ES6 经过了十年，从 ES6 来看，发版周期过长存在 2 个问题：

- 版本之间的时间跨度太长，提早定稿的特性要等待非常长的时间，一直等到规范正式发布（才能被实现和使用），而靠后的特性往往赶在最后发版期限之前才定稿，存在风险
- 语言特性的设计与实现和使用相隔太久，在实现和使用阶段才发现设计缺陷为时已晚

为此，TC39 (ECMA 国际组织第 39 号技术委员会) 启动了新的流程，一份提案要成为 JavaScript 标准，要经过五个流程。

在 Github 上公布了所有的提案，每两个月讨论一次提案。

TC39 每年都会选取一个时间点，对规范进行快照，使其成为当年的 ECMAScript 语言标准，并附上版本号。

其实我觉得我们现在不必关注每年发布了什么特性，比如 ES2023 发布了什么特性，而只需要关注有什么提案完成了即可，官方每年都推出一个版本可能是想表示每年都有一个什么样的进展吧，但其实不公布也没事，具体落到实处的还是看有什么提案完成了，像其他的规范，比如 CSS 规范就没有要每年发布一个版本，甚至已经没有版本的概念了，虽然一直说的是 CSS3，但是已经是划分为多个模块，各个模块独自发展。

1. Stage 0: 稻草人阶段 (strawman)
  1. 任何人都可以提交 pull request 到 [GitHub - tc39/ecma262: Status, process, and documents for ECMA262](#)
  2. 可以是一个提议，想法，初步描述
2. Stage 1: 提案阶段 (proposal)
  1. TC39 成员作为 champion
  2. TC39 审阅通过
  3. 有实现的 Demo 或者 Polyfill
  4. 初步描写标准的语义语法，算法复杂度，解决的问题等
3. Stage 2: 草案阶段 (Draft)
  1. 有两个或两个以上的实现（包括 babel 这类的转译实现）
  2. 使用正式的语言描述该语法，api 等

#### 4. Stage 3: 候选阶段 (candidate)

1. 至少 2 个实现，可以为实验性实现
2. ECMAScript spec editor 通过审核
3. TC39 review 通过
4. 文档编写完成

#### 5. Stage 4: 完成阶段 (finished)

1. 编写 test 262 测试用例
2. 通过两个实现该特性的内核测试
3. ECMAScript spec editor 通过审核
4. 开发者表示支持和认可

# ECMA-262

## ECMAScript 规范

ECMA-262 是有关于 ECMAScript 的规范，里面描述了语言的所有细节，浏览器和 Node.js 均是以此规范而进行实现的。但是除此之外，浏览器和 Node.js 还实现了一些它们特有的东西。

由于这是一门规范，事无巨细的包含了语言的所有细节，因此不会有人从头到尾读一遍的，相反，只看与你试图寻找的东西相对应的部分，并且在该部分只看你需要的东西。

我们来看下哪些是 ECMAScript 规范：

语法，如 `for ... in`



语义，如 `typeof null`



`Object`, `Array`, `Function`, `Number`, `Math`,  
`Proxy`, `globalThis`, ...



`console`, `setTimeout`, `setInterval`,  
`clearTimeout`, `clearInterval`



这些东西在浏览器和 Node.js 中都有，但都是非标准的。

`Buffer`, `process`, `global`



这些都是只针对 Node.js 的 `globals`, `globalThis` 是由标准定义的，并且浏览器进行了实现

`module`, `exports`, `require`, `__dirname`,  
`__filename`



这些是仅适用于 Node.js 模块的全局值

`window`, `alert`, `confirm`, `document`, ...



浏览器独有

# 例子

`String.prototype.substring`

不运行代码，下面的代码会返回什么：

```
String.prototype.substring.call(undefined, 2, 4);
```

这里我们有两种猜想：

1. 将 `undefined` 转化为 `"undefined"`，然后返回 `[2, 4)` 的值，即 `"de"`
2. 拒绝将 `undefined` 作为参数，直接抛出错误

这时你去搜 MDN，但是 MDN 也没有给出这种情况的解释，这个时候你就需要看规范了。

打开 <https://tc39.es/ecma262/>，然后搜索 `String.prototype.substring` 可以看到一条记录，详细的描述了功能和整个的算法流程：

#### 22.1.3.24 `String.prototype.substring ( start, end )`

This method returns a substring of the result of converting this object to a String, starting from index `start` and running to, but not including, index `end` of the String (or through the end of the String if `end` is `undefined`). The result is a String value, not a String object.

If either argument is `NaN` or negative, it is replaced with zero; if either argument is strictly greater than the length of the String, it is replaced with the length of the String.

If `start` is strictly greater than `end`, they are swapped.

It performs the following steps when called:

1. Let `O` be ? `RequireObjectCoercible(this value)`.
2. Let `S` be ? `ToString(O)`.
3. Let `len` be the length of `S`.
4. Let `intStart` be ? `ToIntegerOrInfinity(start)`.
5. If `end` is `undefined`, let `intEnd` be `len`; else let `intEnd` be ? `ToIntegerOrInfinity(end)`.
6. Let `finalStart` be the result of clamping `intStart` between 0 and `len`.
7. Let `finalEnd` be the result of clamping `intEnd` between 0 and `len`.
8. Let `from` be `min(finalStart, finalEnd)`.
9. Let `to` be `max(finalStart, finalEnd)`.
10. Return the `substring` of `S` from `from` to `to`.

**NOTE** This method is intentionally generic; it does not require that its `this` value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

我们看第一步：

Let  $O$  be ? RequireObjectCoercible ( $this$  value).

首先对  $this$  进行了 RequireObjectCoercible 抽象操作，在这个抽象操作的前面有一个  $?$  表示这个抽象操作可能抛出错误，如果是  $!$  就表示不会抛出错误。接着我们看下 RequireObjectCoercible 这个抽象操作做了什么，它试图将值转化为对象，如果不能转换，则抛出错误，从表中可以看到，对于 undefined 会抛出一个 TypeError。

### 7.2.1 RequireObjectCoercible (*argument*)

The abstract operation RequireObjectCoercible takes argument *argument* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript language value or a throw completion. It throws an error if *argument* is a value that cannot be converted to an Object using ToObject. It is defined by [Table 14](#):

Table 14: RequireObjectCoercible Results

Argument Type	Result
Undefined	Throw a <b>TypeError</b> exception.
Null	Throw a <b>TypeError</b> exception.
Boolean	Return <i>argument</i> .
Number	Return <i>argument</i> .
String	Return <i>argument</i> .

下图是在谷歌浏览器中的运行结果：

```
> String.prototype.substring.call(undefined, 2, 4);
✖ ▶Uncaught TypeError: String.prototype.substring called on null or undefined
  at substring (<anonymous>)
  at <anonymous>:1:28
```

推荐一篇文章，可以帮助你更好的理解标准中的一些简写和符号，并且也给出了一些例子，教你如何看规范，上面的例子就取自于这篇文章：[How to Read The ECMAScript Specification](#)。

# ECMA-402

## ECMAScript 国际化 API 规范

提供语言敏感功能，作为 ECMAScript 的补充，支持需要适应不同人类语言和国家使用的语言和文化习惯的程序。

ECMAScript 提供了一些本地化的函数，但用户对此没有控制权，其行为由实现定义，如：

- `Array.prototype.toLocaleString`
- `String.prototype.localeCompare`
- `Date.prototype.toLocaleString`
- ...

国际化 API 提供了几个关键的语言敏感功能，包括：

- 字符串比较
- 数字格式化
- 日期和时间格式化
- 相对时间格式化
- ...

所有的功能都打包在 `Intl` 对象中，以免命名冲突：

- `Intl.Collator`
- `Intl.DateTimeFormat`
- `Intl.DisplayNames`
- `Intl.NumberFormat`
- ...

本地化字符比较，根据拼音排序：

```
const username = ["陈坤", "邓超", "杜淳", "冯绍峰", "韩庚", "胡歌", "黄晓明", "贾乃亮", "李晨"];  
  
// ['陈坤', '邓超', '杜淳', '冯绍峰', '韩庚', '胡歌', '黄晓明', '贾乃亮', '李晨']  
username.sort(new Intl.Collator("zh").compare);
```

数字格式化：

```
// 12,345.68人民币  
new Intl.NumberFormat("zh-Hans", {  
  style: "currency",  
  currency: "CNY",  
  currencyDisplay: "name",  
}).format(12345.6789);
```

日期格式化：

```
// 2023/03/09 18:43:47
const res = new Intl.DateTimeFormat("zh", {
  year: "numeric",
  // 如果不足两位，会自动补零
  month: "2-digit",
  day: "2-digit",
  hour: "2-digit",
  minute: "2-digit",
  second: "2-digit",
  // 设置为 false 表示采用24小时制
  hour12: false,
}).format(new Date());
```

# ECMA-404

## JSON 规范

JSON 是一个在多语言之间进行数据交换的一种格式，它是纯文本，不过是结构化的，最初起源于 JavaScript，不过现在已经成为了多语言之间进行数据交换的一种格式。为了在多个语言之间正确解析数据，需要规定 JSON 的格式，ECMA-404 正是规定 JSON 文本格式的规范。

JSON 的文本由遵循 JSON 值语法的 token 组成，这些 token 由六个结构化 token、strings, numbers，和其他三个字面值组成。

六个结构化 token:

- [
- {
- ]
- }
- :
- ,

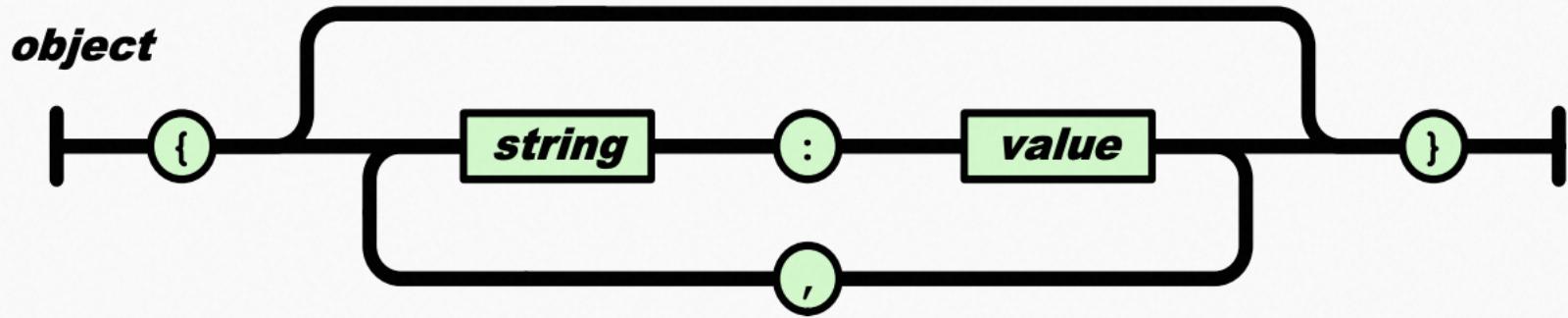
三个字面值:

- true
- false
- null

JSON 的值只能为 object, array, number, string, true, false, null。

# Object

对于 `object`, 它由一对 `{}` 进行包裹, 其中包括 0 个或更多的键值对, 键是 `string`, `value` 是一个合法的 JSON 的值, 每对键值对使用 `,` 进行分隔。

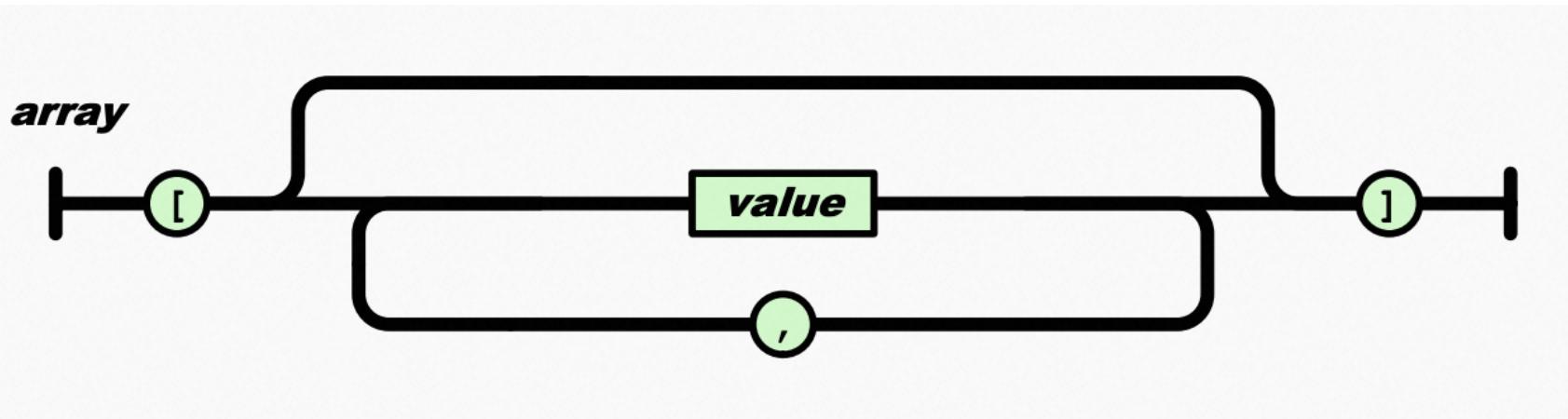


对于作为键的字符串, JSON 语法没有什么限制, 不要求唯一, 不要求有特别的顺序。

不支持循环引用。

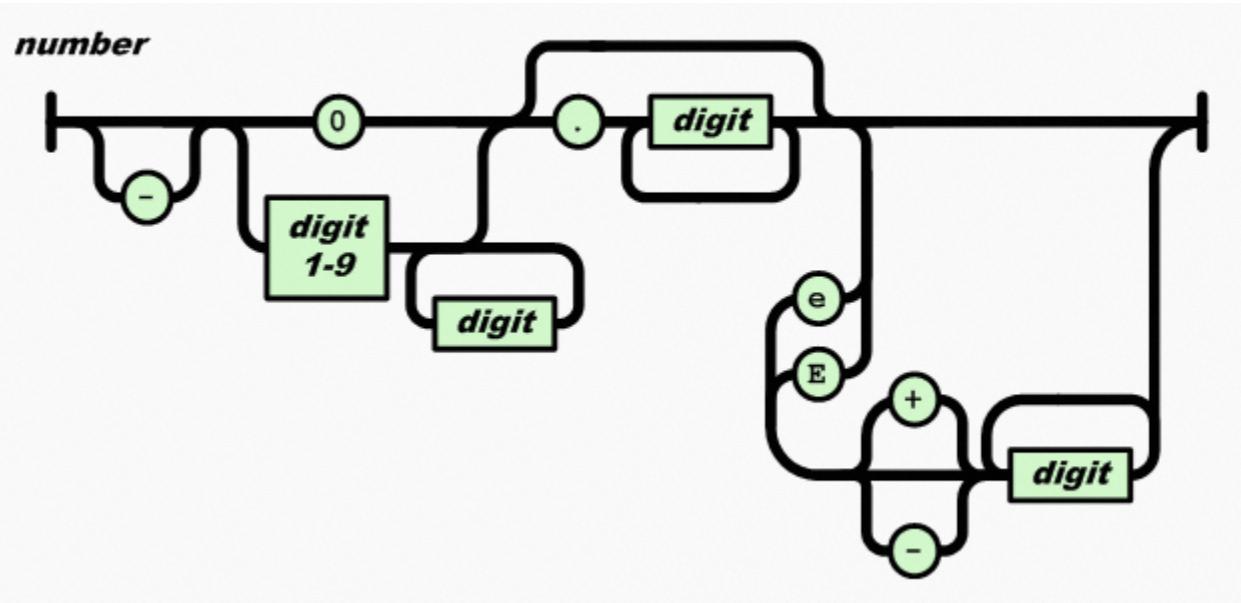
# Arrays

array 由一对 `[]` 进行包裹，其中包含  $0+$  个 JSON 值，值之间通过 `,` 进行分隔，JSON 语法对值的顺序没有要求。



# Numbers

`number` 是一系列一个不包含先导 `0` 的十进制数字，可能在数字前包含 `-` 号，随后可能包含以小数点为前缀的小数部分，随后也可能包含指数 `e/E-/+` 组成。



正确的例子

- -0.5
- 1.5
- 2.1e5
- 2.1E-3
- 34

错误的例子：

- 03：包含前导零
- .6：小数点前要有数字

# String

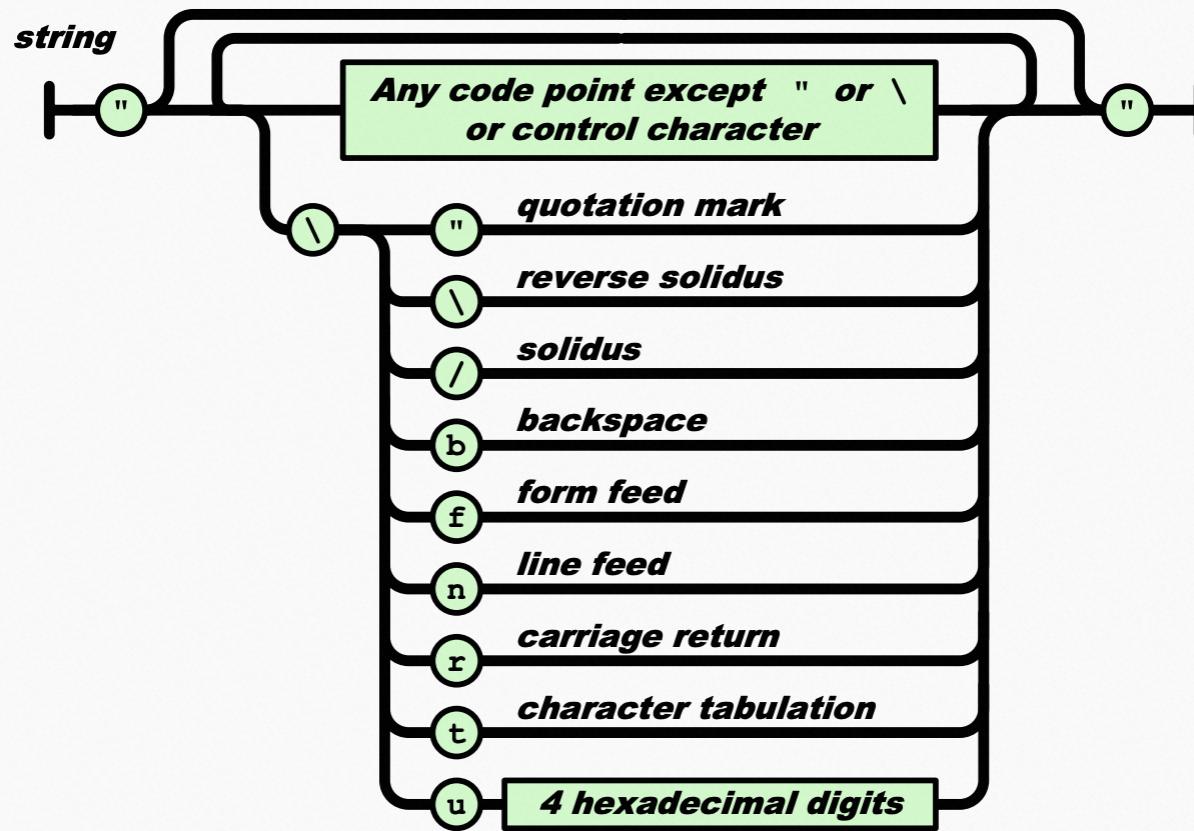
`string` 由一对 `""` 进行包裹，除了一些需要转义的特殊字符，其中可以放任何的 `Unicode` 字符，需要转义字符包括：`" \` 以及控制字符 `U+0000 - U+001F`。

通过两个字符的转义序列来表示一些字符：

- `\"`
- `\\`
- `\/`
- `\b`
- `\n`
- ... ...

还可以直接通过 `Unicode` 码点直接表示字符（包括转义字符）

- `\u002F`
- `\u002f`
- `\/`



# ECMA-414

规定了哪些规范是和 ECMAScript 有关的。

目前内部就包含了 262, 402 和 404。