阮一峰

规格文件（specification）是计算机语言的官方标准，详细描述语法规则和实现方法。

阅读规格的好处:

了解编译器的编译过程；

查询疑难的语法问题（解决语法问题的最可靠、最权威的终极方法）

<https://v8.dev/blog/understanding-ecmascript-part-1>

<https://zhuanlan.zhihu.com/p/360497030>（中文译文 李松峰）

# Understanding the ECMAScript spec, part 1

Published 03 February 2020 · Tagged with [ECMAScript](https://v8.dev/blog/tags/ecmascript) [Understanding ECMAScript](https://v8.dev/blog/tags/understanding-ecmascript)

[All episodes](https://v8.dev/blog/tags/understanding-ecmascript)

In this article, we take a simple function in the spec and try to understand the notation(**符号**). Let’s go!

## Preface

Even if you know JavaScript, reading its language specification, [ECMAScript Language specification, or the ECMAScript spec for short](https://tc39.es/ecma262/), can be pretty daunting. At least that’s how I felt when I started reading it for the first time.

Let’s start with a concrete example and walk through the spec to understand it. The following code demonstrates usage of Object.prototype.hasOwnProperty:

const o = { foo: 1 };  
o.hasOwnProperty('foo'); // true  
o.hasOwnProperty('bar'); // false

In the example, o doesn’t have a property called hasOwnProperty, so we walk up the prototype chain and look for it. We find it in o’s prototype, which is Object.prototype.

To describe how Object.prototype.hasOwnProperty works, the spec uses pseudocode-like descriptions:

[Object.prototype.hasOwnProperty(V)](https://tc39.es/ecma262" \l "sec-object.prototype.hasownproperty)

When the hasOwnProperty method is called with argument V, the following steps are taken:

1. Let P be ? ToPropertyKey(V).
2. Let O be ? ToObject(this value).
3. Return ? HasOwnProperty(O, P).

…and…**(以及)**

[HasOwnProperty(O, P)](https://tc39.es/ecma262" \l "sec-hasownproperty)

The abstract operation HasOwnProperty is used to determine whether an object has an own property with the specified property key. A Boolean value is returned. The operation is called with arguments O and P where O is the object and P is the property key. This abstract operation performs the following steps:

1. Assert: Type(O) is Object.
2. Assert: IsPropertyKey(P) is true.
3. Let desc be ? O.[[GetOwnProperty]](P).
4. If desc is undefined, return false.
5. Return true.

But what’s an “abstract operation”? What are the things inside [[ ]]? Why is there a ? in front of a function? What do the asserts mean?

Let’s find out!

## Language types and specification types

Let’s start with something that looks familiar. The spec uses values such as undefined, true, and false, which we already know from JavaScript. They are all [language values](https://tc39.es/ecma262/" \l "sec-ecmascript-language-types), values of **language types** which the spec also defines.

The spec also uses language values internally, for example, an internal data type might contain a field whose possible values are true and false. In contrast(**相对而言**), JavaScript engines don’t typically use language values internally. For example, if the JavaScript engine is written in C++, it would typically use the C++ true and false (and not its internal representations(**内部表示**) of the JavaScript true and false).

In addition to language types, the spec also uses [specification types](https://tc39.es/ecma262/" \l "sec-ecmascript-specification-types), which are types that occur only in the spec, but not in the JavaScript language. The JavaScript engine does not need to (but is free to) implement(**实现;完成**) them. In this blog post(**博客条目**), we'll get to know the specification type Record (and its subtype Completion Record).

## Abstract operations

[Abstract operations](https://tc39.es/ecma262/" \l "sec-abstract-operations) are functions defined in the ECMAScript spec; they are defined for the purpose of writing the spec **concisely(简洁地)**. A JavaScript engine doesn’t have to implement them as separate functions inside the engine. They cannot be directly called from JavaScript.

## Internal slots(插槽) and internal methods

[Internal slots and internal methods](https://tc39.es/ecma262/" \l "sec-object-internal-methods-and-internal-slots) use names enclosed in [[ ]].

Internal slots are **data members(数据成员)** of a JavaScript object or a specification type. They are used for storing the state of the object. Internal methods are **member functions(内部成员函数)** of a JavaScript object.

For example, every JavaScript object has an internal slot [[Prototype]] and an internal method [[GetOwnProperty]].

Internal slots and methods are not accessible from JavaScript. For example, you cannot access o.[[Prototype]] or call o.[[GetOwnProperty]](). A JavaScript engine can implement them for their own **internal use(内部使用)**, but doesn’t have to.

Sometimes internal methods **delegate** to similarly-named abstract operations, such as in the case of ordinary objects' [[GetOwnProperty]]:

[[[GetOwnProperty]](P)](https://tc39.es/ecma262/" \l "sec-ordinary-object-internal-methods-and-internal-slots-getownproperty-p)

When the [[GetOwnProperty]] internal method of O is called with property key P, the following steps are taken:

1. Return ! OrdinaryGetOwnProperty(O, P).

(We’ll **find out(查明，搞清楚)** what the **exclamation mark** means in the next chapter.)

OrdinaryGetOwnProperty is not an internal method, **since(因为)** it’s not associated with any object; instead, the object it operates on is passed as a parameter.

OrdinaryGetOwnProperty is called “ordinary” since it operates on ordinary objects. ECMAScript objects can be either **ordinary** or **exotic**. Ordinary objects must have the default behavior for **a set of(一套；一组)** methods called **essential internal methods(基本内部方法)**. If an object **deviates(脱离；越轨)** from the default behavior, it’s exotic.

The most well-known **exotic(奇异的)** object is the Array, since its length property behaves in a non-default way: setting **the length property** can remove elements from the Array.

Essential internal methods are the methods listed [here](https://tc39.es/ecma262/" \l "table-5).

## Completion records

What about the question marks and exclamation marks? To understand them, we need to **look into** [Completion Records](https://tc39.es/ecma262/" \l "sec-completion-record-specification-type)!

Completion Record is a **specification type(规范类型)** (only defined for spec purposes). A JavaScript engine doesn’t have to have **a corresponding internal data type**.

A Completion Record is a “record” — a data type which has **a fixed(固定的) set of named fields**. A Completion Record has three fields:

| **Name** | **Description** |
| --- | --- |
| [[Type]] | One of: normal, break, continue, return, or throw. All  other types except normal are **abrupt completions**. |
| [[Value]] | The value that was produced when the completion occurred,  for example, the return value of a function or the  **exception(异常)** (if one is thrown**(v)**). |
| [[Target]] | Used for **directed control transfers(转移)** (not **relevant**  for this blog post).  用于定向转移控制（本文不讨论） |

Every abstract operation **implicitly(暗中地)** returns a Completion Record. Even if it looks like an abstract operation would return a simple type such as Boolean, it’s implicitly **wrapped(包装)** into a Completion Record with the type normal (see [Implicit Completion Values](https://tc39.es/ecma262/" \l "sec-implicit-completion-values)).

Note 1: The spec is not fully **consistent(一致的)** in this regard(**就这一点而言**); there are some **helper functions(辅助函数)** which return **bare values(裸值)** and whose return values are used as is, without extracting the value from the Completion Record. This is usually clear from the context.

Note 2: The spec editors are **looking into(致力于,着手于)** making the Completion Record handling more explicit.

If an algorithm throws an exception, it means **returning(归还，[计]返回，回复)** a Completion Record with [[Type]] throw whose [[Value]] is the exception object. We’ll ignore the break, continue and return types for now.

[ReturnIfAbrupt(argument)](https://tc39.es/ecma262/" \l "sec-returnifabrupt) means taking the following steps:

1. If argument is abrupt, return argument
2. Set argument to argument.[[Value]].

**That is(就是说；换言之)**, we inspect a Completion Record; if it’s an abrupt completion, we return immediately. Otherwise, we **extract(提取)** the value from the Completion Record.

ReturnIfAbrupt might look like a function call, but it’s not. It **causes(引起,导致)** the function where ReturnIfAbrupt() occurs to return, not the ReturnIfAbrupt function itself. It behaves more like a **macro(宏)** in C-like languages.

ReturnIfAbrupt can be used like this:

1. Let obj be Foo(). (obj is a Completion Record.)
2. ReturnIfAbrupt(obj).
3. Bar(obj). (If **we’re still here**, obj is the value extracted from the Completion Record.)

And now [the question mark](https://tc39.es/ecma262/" \l "sec-returnifabrupt-shorthands) comes into play: ? Foo() is equivalent to ReturnIfAbrupt(Foo()). Using a shorthand is practical: we don’t need to write the error handling code explicitly each time.

Similarly, Let val be ! Foo() is equivalent to:

1. Let val be Foo().
2. Assert: val is not an abrupt completion.
3. Set val to val.[[Value]].

Using this knowledge, we can rewrite Object.prototype.hasOwnProperty like this:

Object.prototype.hasOwnProperty(V)

1. Let P be ToPropertyKey(V).
2. If P is an abrupt completion, return P
3. Set P to P.[[Value]]
4. Let O be ToObject(this value).
5. If O is an abrupt completion, return O
6. Set O to O.[[Value]]
7. Let temp be HasOwnProperty(O, P).
8. If temp is an abrupt completion, return temp
9. Let temp be temp.[[Value]]
10. Return NormalCompletion(temp)

…and we can rewrite HasOwnProperty like this:

HasOwnProperty(O, P)

1. Assert: Type(O) is Object.
2. Assert: IsPropertyKey(P) is true.
3. Let desc be O.[[GetOwnProperty]](P).
4. If desc is an abrupt completion, return desc
5. Set desc to desc.[[Value]]
6. If desc is undefined, return NormalCompletion(false).
7. Return NormalCompletion(true).

We can also rewrite the [[GetOwnProperty]] internal method without the exclamation mark:

O.[[GetOwnProperty]]

1. Let temp be OrdinaryGetOwnProperty(O, P).
2. Assert: temp is not an abrupt completion.
3. Let temp be temp.[[Value]].
4. Return NormalCompletion(temp).

Here we **assume(假设)** that temp is a **brand new(全新的；崭新的)** temporary variable which doesn’t **collide(冲突)** with anything else.

We’ve also used the knowledge that when **a return statement** returns something else than a Completion Record, it’s implicitly wrapped inside a NormalCompletion.

### Side track: Return ? Foo()

The spec uses the notation Return ? Foo() — why the question mark?

Return ? Foo() expands to:

1. Let temp be Foo().
2. If temp is an abrupt completion, return temp.
3. Set temp to temp.[[Value]].
4. Return NormalCompletion(temp).

Which is the same as Return Foo(); it behaves the same way for both abrupt and normal completions.

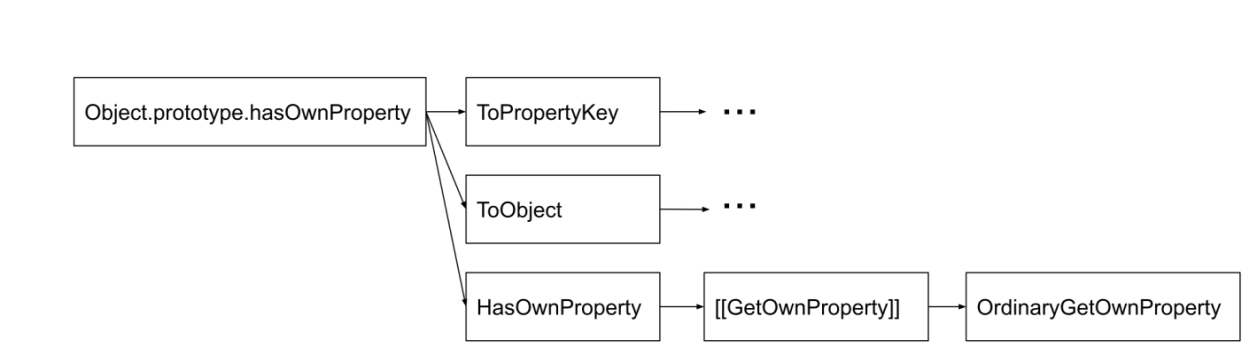
Return ? Foo() is only used for editorial reasons, to make it more explicit that Foo returns a Completion Record.

## Asserts

Asserts in the spec assert invariant conditions of the **algorithms(算法)**. They are added for **clarity**, but don't add any requirements to the implementation — the implementation doesn’t need to check them.

## Moving on

The abstract operations **delegate(委托)** to other abstract operations (see picture below), but based on this blog post we should be able to **figure out** what they do. We’ll **encounter(碰到)** Property Descriptors, which is just another specification type.

Function call graph starting from Object.prototype.hasOwnProperty

## Summary

We read through a simple method — Object.prototype.hasOwnProperty — and **abstract operations** it invokes. We **familiarized(使熟悉)** ourselves with the **shorthands** ? and ! related to error handling. We **encountered(遇到了,了解了)** **language types**, **specification types**, **internal slots**, and **internal methods**.

## Useful links

[How to Read the ECMAScript Specification](https://timothygu.me/es-howto/): a tutorial which covers much of the material covered in this post, from a slightly different angle.



**Posted by** [Marja Hölttä](https://twitter.com/marjakh), **speculative specification spectator**.

**[Retweet this article!](https://twitter.com/v8js/status/1224363301146189824)**

参考资料

[1]

语言值: [https://tc39.es/ecma262/#sec-ecmascript-language-types](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-ecmascript-language-types" \t "https://zhuanlan.zhihu.com/p/_blank)

[2]

规范类型: [https://tc39.es/ecma262/#sec-ecmascript-specification-types](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-ecmascript-specification-types" \t "https://zhuanlan.zhihu.com/p/_blank)

[3]

抽象操作: [https://tc39.es/ecma262/#sec-abstract-operations](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-abstract-operations" \t "https://zhuanlan.zhihu.com/p/_blank)

1. 内部栏位（slot）和内部方法:

[https://tc39.es/ecma262/#sec-object-internal-methods-and-internal-slots](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-object-internal-methods-and-internal-slots" \t "https://zhuanlan.zhihu.com/p/_blank)

[5] 基本内部方法: [https://tc39.es/ecma262/#table-5](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23table-5" \t "https://zhuanlan.zhihu.com/p/_blank)

[6]

完成记录: [https://tc39.es/ecma262/#sec-completion-record-specification-type](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-completion-record-specification-type" \t "https://zhuanlan.zhihu.com/p/_blank)

[7]

隐式完成值: [https://tc39.es/ecma262/#sec-implicit-completion-values](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-implicit-completion-values" \t "https://zhuanlan.zhihu.com/p/_blank)

[8]

问号: [https://tc39.es/ecma262/#sec-returnifabrupt-shorthands](https://link.zhihu.com/?target=https://tc39.es/ecma262/%23sec-returnifabrupt-shorthands" \t "https://zhuanlan.zhihu.com/p/_blank)

[9]

原文链接: [https://v8.dev/blog/understandi](https://link.zhihu.com/?target=https://v8.dev/blog/understanding-ecmascript-part-1" \t "https://zhuanlan.zhihu.com/p/_blank)