[ECMAScript 2015 Language Specification – ECMA-262 6th Edition (ecma-international.org)](https://262.ecma-international.org/6.0/" \l "sec-7.2.12)

[ECMAScript 2015 Language Specification – ECMA-262 6th Edition (ecma-international.org)](https://262.ecma-international.org/6.0/" \l "sec-7.2.12)

https://262.ecma-international.org/6.0/#sec-7.2.12

语法行为：0 == null

# [7.2.12](https://262.ecma-international.org/6.0/" \l "sec-abstract-equality-comparison" \o "link to this section) Abstract Equality Comparison

The comparison x == y, where x and y are values, produces **true** or **false**. Such a comparison is performed as follows:

1. [ReturnIfAbrupt](https://262.ecma-international.org/6.0/" \l "sec-returnifabrupt)(*x*).
2. [ReturnIfAbrupt](https://262.ecma-international.org/6.0/" \l "sec-returnifabrupt)(*y*).
3. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is the same as [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*), then
   1. Return the result of performing Strict Equality Comparison *x* === *y*.
4. If *x* is **null** and *y* is **undefined**, return **true**.
5. If *x* is **undefined** and *y* is **null**, return **true**.
6. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is Number and [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*) is String,  
   return the result of the comparison *x* == [ToNumber](https://262.ecma-international.org/6.0/" \l "sec-tonumber)(*y*).
7. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is String and [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*) is Number,  
   return the result of the comparison [ToNumber](https://262.ecma-international.org/6.0/" \l "sec-tonumber)(*x*) == *y*.
8. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is Boolean, return the result of the comparison [ToNumber](https://262.ecma-international.org/6.0/" \l "sec-tonumber)(*x*) == *y*.
9. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*) is Boolean, return the result of the comparison *x* == [ToNumber](https://262.ecma-international.org/6.0/" \l "sec-tonumber)(*y*).
10. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is either String, Number, or Symbol and [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*) is Object, then  
    return the result of the comparison *x* == [ToPrimitive](https://262.ecma-international.org/6.0/" \l "sec-toprimitive)(*y*).
11. If [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*x*) is Object and [Type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-data-types-and-values)(*y*) is either String, Number, or Symbol, then  
    return the result of the comparison [ToPrimitive](https://262.ecma-international.org/6.0/" \l "sec-toprimitive)(*x*) == *y*.
12. Return **false**.

伪代码（Pseudocode）：用于描述模块结构图的语言。

[解读闭包，这次从ECMAScript词法环境，执行上下文说起](https://www.cnblogs.com/wenbinjiang/p/13476342.html)

<https://www.cnblogs.com/wenbinjiang/p/13476342.html>

JavaScript的闭包实现细节

<https://zhuanlan.zhihu.com/p/25296587>

**下面的官网中文译文：参考<http://www.xiuyetang.com/view/249.html>**

# [8](https://262.ecma-international.org/6.0/" \l "sec-executable-code-and-execution-contexts" \o "link to this section) Executable Code and Execution Contexts

[8.1](https://262.ecma-international.org/6.0/" \l "sec-lexical-environments" \o "link to this section) Lexical Environments

A *Lexical Environment* is a specification type used to define the association of *Identifiers* to **specific(具体的；特定的)** variables and functions based upon the lexical **nesting structure(嵌套结构)** of ECMAScript code. A Lexical Environment consists of an [Environment Record](https://262.ecma-international.org/6.0/" \l "sec-environment-records) and a possibly null reference(**空引用**) to an *outer* Lexical Environment. Usually a Lexical Environment is associated with some specific **syntactic structure(句法结构)** of ECMAScript code such as a *FunctionDeclaration*, a *BlockStatement(****块语句****)*, or a *Catch* **clause(子句，从句，分句)** of a *TryStatement* and a new Lexical Environment is created each time such code is **evaluated**.

An [Environment Record](https://262.ecma-international.org/6.0/" \l "sec-environment-records) records the identifier bindings that are created within the scope of its associated Lexical Environment. It is referred to as the Lexical Environment’s EnvironmentRecord

The outer environment reference is used to model the logical nesting of Lexical Environment values. The outer reference of a (inner) Lexical Environment is a reference to the Lexical Environment that logically surrounds the inner Lexical Environment. An outer Lexical Environment may, of course, have its own outer Lexical Environment. A Lexical Environment may serve as the outer environment for multiple inner Lexical Environments. For example, if a *FunctionDeclaration* contains two nested *FunctionDeclarations* then the Lexical Environments of each of the nested functions will have as their outer Lexical Environment the Lexical Environment of the current evaluation of the surrounding function.

A *global environment* is a Lexical Environment which does not have an outer environment. The global environment’s outer environment reference is **null**. A global environment’s EnvironmentRecord may be prepopulated with identifier bindings and includes an associated *global object* whose properties provide some of [the global environment](https://262.ecma-international.org/6.0/" \l "sec-global-environment-records)’s identifier bindings. This global object is the value of a global environment’s **this** binding. As ECMAScript code is executed, additional properties may be added to the global object and the initial properties may be modified.

A *module environment* is a Lexical Environment that contains the bindings for the top level declarations of a *Module*. It also contains the bindings that are explicitly imported by the *Module*. The outer environment of a module environment is a global environment.

A *function environment* is a Lexical Environment that corresponds to the invocation of an [ECMAScript function object](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-function-objects). A function environment may establish a new **this** binding. A function environment also captures the state necessary to support **super** method invocations.

Lexical Environments and [Environment Record](https://262.ecma-international.org/6.0/" \l "sec-environment-records) values are purely specification mechanisms and need not correspond to any specific artefact of an ECMAScript implementation. It is impossible for an ECMAScript program to directly access or manipulate such values.

[8.1.1](https://262.ecma-international.org/6.0/" \l "sec-environment-records" \o "link to this section) Environment Records

There are two primary kinds of Environment Record values used in this specification: *declarative Environment Records* and *object Environment Records*. Declarative Environment Records are used to define the effect of ECMAScript language syntactic elements such as *FunctionDeclarations*, *VariableDeclarations*, and *Catch* clauses that directly associate identifier bindings with [ECMAScript language values](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-language-types). Object Environment Records are used to define the effect of ECMAScript elements such as *WithStatement* that associate identifier bindings with the properties of some object. [Global Environment Records](https://262.ecma-international.org/6.0/" \l "sec-global-environment-records) and function Environment Records are specializations that are used for specifically for *Script* global declarations and for top-level declarations within functions.

For specification purposes Environment Record values are values of the Record specification type and can be thought of as existing in a simple object-oriented hierarchy where Environment Record is an abstract class with three concrete subclasses, declarative Environment Record, object Environment Record, and global Environment Record. [Function Environment Records](https://262.ecma-international.org/6.0/" \l "sec-function-environment-records) and module Environment Records are subclasses of declarative Environment Record. The abstract class includes the abstract specification methods defined in [Table 15](https://262.ecma-international.org/6.0/" \l "table-15). These abstract methods have distinct concrete algorithms for each of the concrete subclasses.

Table 15 — Abstract Methods of Environment Records

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| HasBinding(N) | Determine if an Environment Record has a binding for the String value N. Return **true** if it does and **false** if it does not |
| CreateMutableBinding(N, D) | Create a new but uninitialized mutable binding in an Environment Record. The String value N is the text of the bound name. If the optional Boolean argument D is **true** the binding is may be subsequently deleted. |
| CreateImmutableBinding(N, S) | Create a new but uninitialized immutable binding in an Environment Record. The String value N is the text of the bound name. If S is **true** then attempts to access the value of the binding before it is initialized or set it after it has been initialized will always throw an exception, regardless of the strict mode setting of operations that reference that binding. S is an optional parameter that defaults to **false**. |
| InitializeBinding(N,V) | Set the value of an already existing but uninitialized binding in an Environment Record. The String value N is the text of the bound name. V is the value for the binding and is a value of any [ECMAScript language type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-language-types). |
| SetMutableBinding(N,V, S) | Set the value of an already existing mutable binding in an Environment Record. The String value N is the text of the bound name. V is the value for the binding and may be a value of any [ECMAScript language type](https://262.ecma-international.org/6.0/" \l "sec-ecmascript-language-types). S is a Boolean flag. If S is **true** and the binding cannot be set throw a **TypeError** exception. |
| GetBindingValue(N,S) | Returns the value of an already existing binding from an Environment Record. The String value N is the text of the bound name. S is used to identify references originating in [strict mode code](https://262.ecma-international.org/6.0/" \l "sec-strict-mode-code) or that otherwise require strict mode reference semantics. If S is **true** and the binding does not exist throw a **ReferenceError** exception. If the binding exists but is uninitialized a **ReferenceError** is thrown, regardless of the value of *S.* |
| DeleteBinding(N) | Delete a binding from an Environment Record. The String value N is the text of the bound name. If a binding for N exists, remove the binding and return **true**. If the binding exists but cannot be removed return **false**. If the binding does not exist return **true**. |
| HasThisBinding() | Determine if an Environment Record establishes a **this** binding. Return **true** if it does and **false** if it does not. |
| HasSuperBinding() | Determine if an Environment Record establishes a **super** method binding. Return **true** if it does and **false** if it does not. |
| WithBaseObject () | If this Environment Record is associated with a **with** statement, return the with object. Otherwise, return **undefined**. |

[8.1.1.1](https://262.ecma-international.org/6.0/" \l "sec-declarative-environment-records" \o "link to this section) Declarative Environment Records

Each declarative [Environment Record](https://262.ecma-international.org/6.0/" \l "sec-environment-records) is associated with an ECMAScript program scope containing variable, constant, let, class, module, import, and/or function declarations. A declarative [Environment Record](https://262.ecma-international.org/6.0/" \l "sec-environment-records) binds the set of identifiers defined by the declarations contained within its scope.

The behaviour of the concrete specification methods for declarative Environment Records is defined by the following algorithms.