# **Javascript之 Symbol 类型**

根据规范，对象属性键只能是string类型或symbol类型,不能是number、boolean，只有string和symbol两种类型。   
我们一般比较属性string，现在看看symbols类型给我们带来的优势。

## **Symbols**

“Symbol”值代表用给定名称作为唯一标识。这种类型的值可以这样创建：Symbol(name):

// id is a symbol with the name "id"

let id = Symbol("id");

Symbols确保唯一，即使我们使用相同的名称，也会产生不同的值。举例，这里有两个symbol使用相同的名称，两种不相等：

let id1 = Symbol("id");

let id2 = Symbol("id");

alert(id1 == id2); // false

如果你熟悉Ruby或其他语言，也有相同类型的symbol，不要被误导，Javascript的symbol是不同的。

## **隐藏属性**

symbol允许创建对象的隐藏属性，这样其他代码不会偶尔访问或覆盖。   
举例，如果我们想存储对象user的“identifier”，我们可以创建id的symbol：   
let user = { name: “John” };   
let id = Symbol(“id”);

user[id] = "ID Value";

alert( user[id] ); // we can access the data using the symbol as the key

现在让我们想像假设另外的脚本想给user对象添加自己的“id”属性，为其自有的目的。可能是另外其他的Javascript库，所以完全相互不清楚彼此。   
没有问题，能够创建自己的Symbol("id")。

代码如下：

// ...

let id = Symbol("id");

user[id] = "Their id value";

这是没有冲突的，因为symbol总是不同的，即使有相同的名称。注意如果我们使用字符串“id”代替symbol实现相同目的，那么会有冲突：

let user = { name: "John" };

// our script uses "id" property

user.id = "ID Value";

// ...if later another script the uses "id" for its purposes...

user.id = "Their id value"

// boom! overwritten! it did not mean to harm the colleague, but did it!

## **直接量使用symbol**

如果我们在一个直接量对象中使用symbol，我们需要使用方括号：

let id = Symbol("id");

let user = {

name: "John",

[id]: 123 // not just "id: 123"

};

因为我们需要名称为“id”的symbol变量值，不是字符串“id”。

## **Symbol 被 for…in 忽略**

symbol属性不参与for..in循环，举例：

let id = Symbol("id");

let user = {

name: "John",

age: 30,

[id]: 123

};

for(let key in user) alert(key); // name, age (no symbols)

// the direct access by the symbol works

alert( "Direct: " + user[id] );

这时一般隐藏概念的一部分，如果其他脚本或库，也不期望访问symbol属性。   
相反，Object.assign同时拷贝字符属性和symbol属性。

let id = Symbol("id");

let user = {

[id]: 123

};

let clone = Object.assign({}, user);

alert( clone[id] ); // 123

这两者没有矛盾，规范就是这么设计的，其思想是当我们克隆对象或合并对象，通常希望symbol属性也被拷贝。

其他类型的属性键被强制转换成字符串：

对象中的键只能使用字符串或symbol，其他类型强制被转成字符串。

let obj = {

0: "test" // same as "0": "test"

}

// both alerts access the same property (the number 0 is converted to string "0")

alert( obj["0"] ); // test

alert( obj[0] ); // test (same property)

## **全局symbol**

通常，所有symbol是不同的，但有时我们想相同名词的symbol是相同的。举例，我们应用不同部分想访问名词为”id“的symbol，当然需要是相同的。   
可以通过全局symbol注册实现，我们能先创建，然后访问他们，并且保证通过相同名词重复访问获得相同的symbol。

在注册中创建或读取，使用语法为：symbol.for(name),举例：

// read from the global registry

let name = Symbol.for("name"); // if the symbol did not exist, it is created

// read it again

let nameAgain = Symbol.for("name");

// the same symbol

alert( name === nameAgain ); // true

在注册中symbol称为全局symbol，如果我们有应用范围的symbol，代码都可以访问，可以使用全局symbol。

其他编程语言，想Ruby，每个名词有单个symbol，在javascript中，我们知道，正是全局symbol。

## **Symbol.keyFor**

对全局symbol，不仅有Symbol.for(name)，根据名称返回symbol，也有相反的调用：Symbol.keyFor(name),功能相反：根据全局symbol返回名称。   
举例：

let sym = Symbol.for("name");

let sym2 = Symbol.for("id");

// get name from symbol

alert( Symbol.keyFor(sym) ); // name

alert( Symbol.keyFor(sym2) ); // id

Symbol.keyfor内部实现，使用全局symbol注册，根据symbol查找symbol名称。   
所以对非全局的symbol不其作用，如果不是全局symbol，则找不到，返回undefined。

举例：

alert( Symbol.keyFor(Symbol.for("name")) ); // name, global symbol

alert( Symbol.keyFor(Symbol("name2")) ); // undefined, non-global symbol

非全局symbol，名称仅用于调试目的。

## **系统symbol**

在Javascript内部存在很多系统symbol，我们能使用他们调整对象的各个方面。   
这里列出一些常用的symbol：

* Symbol.hasInstance
* Symbol.isConcatSpreadable
* Symbol.iterator
* Symbol.toPrimitive
* …

举例：Symbol.toPrimitive用于对象转基本类型时描述对象，如果你继续深入学习Javascript，其他的你也会逐步熟悉。

## **总结**

* symbol是基本类型，实现唯一标识
* 通过调用symbol(name)创建symbol
* 我们创建一个字段，仅为知道对应symbol的人能访问，使用symbol很有用
* symbol不会出现在for..in结果中
* 使用symbol(name）创建的symbol，总是不同，即使name相同。如果希望相同名称的symbol相等，则使用全局注册
* symbol.for(name)返回给定名称的全局symbol，多次调用返回相同symbol
* Javascript有系统symbol，通过Symbol.\*访问。我们能使用他们去修改一些内置行为。

技术上，symbol并不是100%隐藏，有内置方法Object.getOwnPropertySymbols(obj)可以获得所有的symbol。   
也有一个方法Reflect.ownKeys(obj)返回对象所有的键，包括symbol。

所以并不是真正隐藏。但大多数库内置方法和语法结构遵循通用约定他们是隐藏的，如果某人显示调用上述方法，可能完全理解他正在做什么。