CodeWithHarry

HTML CSS JS C C++ JAVA PYTHON PHP

JavaScript Symbols | JavaScript Tutorial In Hindi #59

Overview Q&A Downloads Announcements

JavaScript Symbols | JavaScript Tutorial In Hindi #59

In today's tutorial, we will study about the **symbols** which are the newest JavaScript primitive. It brings a few benefits to the JavaScript language and is very useful when used as object properties. Before symbols were introduced in ES6, JavaScript used seven types of data, which is grouped into two categories. Primitives, including the string, number, boolean, null, and undefined data types and another is the Objects, which includes arrays, functions, and regular JS objects.

What is the symbol?

A symbol is a primitive which cannot be recreated. It was introduced in ECMAScript 2015. It is a very peculiar data type. Once we create a symbol, its value is kept private and for internal use. All that remains after the creation of the symbol is the symbol reference. We can create a symbol by calling the Symbol() global factory function:

```
const mySymbol = Symbol()
```

Every time we invoke Symbol(), we get a new and unique symbol, which is different from all other symbols:

Copy

```
const sym1 = Symbol();
const sym2 = Symbol();
console.log(s1 === s2); // false
```

We can pass a parameter to Symbol(), and that is used as the symbol *description*, useful just for debugging purposes:

```
console.log(Symbol()) //Symbol()
console.log(Symbol('Some Test')) //Symbol(Some Test)
```

Examples:

```
const mysymbol = Symbol()
const person = {
  [mysymbol]: 'John'
}
person[mysymbol]
const mysymbol2 = Symbol()
person[mysymbol2] = () => 'Hello World'
console.log(person[mysymbol2]())
//Output: Hello World
```

Purpose of the symbol:-

In ES6, symbols were added to the primitives group, just like all other primitive, they are also immutable and does not have methods of their own. The main purpose of symbols was to provide globally unique values that were kept private and for internal use only.

Use Case: Symbols as keys of non-public properties

When there are inheritance hierarchies in JavaScript, we have two kinds of properties. First one is created via classes, and the second is a purely prototypal approach:

- **Public properties**: They are seen by clients of the code.
- **Private properties**: They are used internally within the pieces that make up the inheritance hierarchy such as classes, objects.

For usability, public properties usually have string keys. However, in the case of private properties with string keys, accidental name clashes can become a problem. Therefore, symbols are the right choice. Here is an example:

Suppose that there are two companies, A and B, developed by two different people. Both of them need to add a property to work on an object, but, they both end up naming their property id by coincidence. This coincidence leads to one of the companies overwriting the data stored in id. Before ES6, when the object key could only be of the string type, this scenario was a real possibility. String type object keys posed the danger of name collision and led to the overwriting of data/values. In such a scenario, symbols have provides a solution to this problem.

```
let student = { name: "Harry" };
let id_companyA = Symbol("id");
student[id_companyA] = "ID assigned by company A";
let id_companyB = Symbol("id");
student[id_companyB] = "ID assigned by company B";
console.log(student)
```

```
//Output: {name: "Harry", Symbol(id): "ID assigned by company A", Symbol(id): "ID assigned by company B"}
```

Conclusion:-

Symbols in JavaScript can provide uniqueness to objects. It is worthwhile for all developer to have a basic understanding of them and their various use-cases.

Code as described/written in the video

```
console.log("This is tutorial 59")

// Symbols
const sym1 =  Symbol('My identifier');
const sym2 =  Symbol('My identifier');
// console.log('Symbol is ', sym1);
// console.log('Type of Symbol is ', typeof sym1);
console.log(sym1 === sym2);

const a = "this is";
const b = "this is";

console.log(a === b);
console.log(null === null);
console.log(undefined === undefined);

const k1 = Symbol('identifier for k1');
const k2 = Symbol('for k2');

myObj = {};
myObj[k1] = "Harry";
myObj[k2] = "Rohan";
myObj["name"] = "Good boy"
myObj[4] = "Good int"


console.log(myObj);
console.log(myObj[k1]);
console.log(myObj[k2]);
console.log(myObj.k2); // !! ALERT !!: cannot do this to get Rohan because it is same as myObj["k2"]
```

```
// Symbols are ignored in for in loop
for(key in myObj){
    console.log(key, myObj[key])
}

console.log(JSON.stringify(myObj));
```

Previous Next