UT03.7: Symbol

Contenido

1. Introducción

Symbol es un objeto global cuyos valores son **únicos e inmutables.** Es tipo nos permite crear valores para ser utilizados como identificadores que. pueden ser utilizados en las propiedades de los objetos. Cada valor `Symbol` tiene asociado un *string* o *undefined* que describen el símbolo.

2. Crear un `Symbol`

Para crear un valor `Symbol` tenemos que utilizar la función factoría `Symbol()`, es decir, no se crean por medio del operador `new()`. Podemos pasarle opcionalmente un *string*, pero solo se utiliza a modo descriptivo para labores de depuración. En el siguiente ejemplo vemos la creación de varios valores. Aunque utilicemos el mismo texto descriptivos, dos valores *symbol* no pueden ser iguales. Los valores *symbol* no pueden ser convertidos a *string* automáticamente, hemos de utilizar el método `toString()`.

```
function createSymbol() {
  // Función factoria para crear un Symbol
  const sym1 = Symbol();
  const sym2 = Symbol('foo');
  const sym3 = Symbol('foo');

  $$result.log(sym1.toString()); // Symbol()
  $$result.log(sym2.toString()); // Symbol(foo)
  $$result.log(sym3.toString()); // Symbol(foo)
  $$result.log(sym2.description); // foo
  $$result.log(typeof sym2); // symbol
  $$result.log(sym2 === sym3); // false
}
```

3. Uso de `Symbol` como propiedades de objetos

Para utilizar un *symbol* como propiedad de un objeto debemos utilizar los corchetes. En este ejemplo podemos ver como definimos un variable de tipo *symbol* utilizando la función factoría del tipo. En un objeto literal podemos utilizar la variable como nombre de propiedad, haciendo que el nombre sea único.

Las propiedades definidas con *symbol* no se pueden enumerar. En el ejemplo vemos un bucle *for/in* para recorrer las propiedades, pero las definidas con *symbol* no son enumeradas. Para poder acceder directamente a la propiedad es necesario hacerlo explícitamente con la notación de corchetes.

```
function propertySymbol() {
  const id = Symbol('id');
  const computer = {
    [id]: 1, // Nombre de propiedad definido con symbol
    brand: 'HP',
    model: 'EliteBook',
    memory: 16,
  };
  $$result.logBold('Un Symbol no es enumerable');
  // Las propiedades symbol no se pueden enumerar.
  for (const property in computer) {
    $$result.log(computer[property]);
  }

  $$result.logBold('Las propiedades Symbol accedemos con la notación [
]');
  $$result.log(computer[id]); // 1
}
```

4. Prevenir colisiones en nombres de propiedades

El tipo de datos *symbol* puede utilizarse para prevenir colisiones entre propiedades de un objeto. En un proyecto es habitual utilizar diversos framework desarrollados por terceros, lo cuales podrían utilizar los mismos nombres de propiedades provocando colisiones entre ambos. Los *symbol* evitan este tipo de colisiones al ser valores únicos.

En esta ocasión estamos emulando el uso de dos librerías externas, *lib1* y *lib2*, que necesitan utilizar una propiedad denominada *id*. Para evitar colisiones utilizamos dos constantes *symbol* para definir la propiedad para cada librería, evitando con ello la colisión entre los nombres.

```
function propertyNameCollision() {
  const library1property = Symbol('lib1.id');
  const library2property = Symbol('lib2.id');
  function lib1tag(obj) {
    obj[library1property] = 1234;
  }
  function lib2tag(obj) {
    obj[library2property] = 'abcd';
  }
  const computer = {
    brand: 'HP',
    model: 'EliteBook',
    memory: 16,
  };
  lib1tag(computer);
  lib2tag(computer);
  $$result.logBold('Evitan colisiones entre nombres');
  $$result.log(computer[library1property]); // 1234
  $$result.log(computer[library2property]); // abcd
}
```

5. Registro global de `Symbol`

Podemos necesitar utilizar el mismo *symbol* en diferentes partes de nuestra aplicación, lo que implica poder acceder al mismo objeto *symbol*. Para evitar tener que utilizar constantes globales, disponemos de un registro de valores *symbol* localizados a partir de una clave. El método `Symbol.for()` crea un un *symbol* disponible en el registro global de *symbols*, por lo que podremos acceder a él posteriormente.

En este último ejemplo creamos un *symbol* en el registro con la clave *"lib1.id"*. Es una buena práctica utilizar un prefijo como clave en el registro para evitar colisiones. Utilizando la clave en el método `Symbol.for()` podemos recuperar el valor *symbol* y reutilizarlo, ya que obtendremos el mismo valor a diferencia de la función factoría que siempre devuelve valores diferentes. Como vemos, cada función que necesite acceder a la propiedad recupera el nombre del registro de *symbol*.

```
function symbolGlobalRegister() {
  // Registramos los symbol para su uso
  Symbol.for('lib1.id');

  function func1(obj) {
    const s = Symbol.for('lib1.id');
    $$result.log(obj[s]); // 1234
  }

  function func2(obj) {
    const s = Symbol.for('lib1.id');
    $$result.log(obj[s]); // 1234
  }

  const computer = {
    brand: 'HP',
    model: 'EliteBook',
    memory: 16,
  };
  $$result.logBold('Registro global de Symbol');
  computer[Symbol.for('lib1.id')] = 1234;
  $$result.log(Symbol.for('lib1.id') === Symbol.for('lib1.id')); // true
  func1(computer);
  func2(computer);
}
```