JS JavaScript TUTORIAL

The Beginner's Guide to JavaScript Set Type in ES6

Summary: in this tutorial, you will learn about the JavaScript `Set` object that allows you to manage a collection of unique values of any type effectively.

Introduction to the JavaScript Set object

ES6 provides a new type named `Set` that stores a collection of unique values of any type. To create a new empty `Set`, you use the following syntax:

```
let setObject = new Set();
```

The `Set` constructor also accepts an optional iterable object (https://www.javascripttutorial.net/es6/javascript-iterator/) . If you pass an iterable object to the `Set` constructor, all the elements of the iterable object will be added to the new set:

```
let setObject = new Set(iterableObject);
```

Useful Set methods

The `Set` object provides the following useful methods:

- `add(value)` – appends a new element with a specified value to the set. It returns the `Set` object, therefore, you can chain this method with another `Set` method.
- `clear()` – removes all elements from the `Set` object.
- `delete(value)` – deletes an element specified by the value.
- `entries()` – returns a new `Iterator` that contains an array of `[value, value]` .
- `forEach(callback [, thisArg])` – invokes a callback (https://www.javascripttutorial.net/javascript-callback/) on each element of the `Set` with the `this` value sets to `thisArg` in each call.

- `has(value)` – returns `true` if an element with a given value is in the set, or `false` if it is not.
- `keys()` – is the same as `values()` function.
- `[@@iterator]` – returns a new `Iterator (https://www.javascripttutorial.net/es6/javascript-iterator/)` object that contains values of all elements stored in the insertion order.

JavaScript Set examples

Create a new Set from an Array

The following example shows how to create a new Set from an array (https://www.javascripttutorial.net/javascript-array/) .

```
let chars = new Set(['a', 'a', 'b', 'c', 'c']);
```

All elements in the set must be unique therefore the `chars` only contains 3 distinct elements `a` , `b` and `c` .

```
console.log(chars);
```

Output:

```
Set { 'a', 'b', 'c' }
```

When you use the `typeof` operator to the `chars`, it returns `object` .

```
console.log(typeof(chars));
```

Output:

```
object
```

The `chars` set is an instance of the `Set` type so the following statement returns `true` .

```
let result = chars instanceof Set;
console.log(result);
```

Get the size of a Set

To get the number of elements that the set holds, you use the `size` property of the `Set` object:

```
let size = chars.size;
console.log(size);//  3
```

Add elements to a Set

To add an element to the set, you use the `add()` method:

```
chars.add('d');
console.log(chars);
```

Output:

```
Set { 'a', 'b', 'c', 'd' }
```

Since the `add()` method is chainable, you can add multiple items to a set using a chain statement:

```
chars.add('e')
     .add('f');
```

Check if a value is in the Set

To check if a set has a specific element, you use the `has()` method. The `has()` method returns `true` if the set contains the element, otherwise, it returns `false`. Since the `chars` set contains `'a'`, the following statement returns `true`:

```
let exist = chars.has('a');
console.log(exist);// true
```

The following statement returns `false` because the `chars` set does not contain the `'z'` value.

```
exist = chars.has('z');
console.log(exist); // false
```

Remove elements from a set

To delete a specified element from a set, you use the `delete()` method. The following statement deletes the `'f'` value from the `chars` set.

```
chars.delete('f');
console.log(chars); // Set {"a", "b", "c", "d", "e"}
```

Output:

```
Set { 'a', 'b', 'c', 'd', 'e' }
```

The `delete()` method returns `true` indicating that the element has been removed successfully. To delete all elements of a set, you use the `clear()` method:

```
chars.clear();
console.log(chars); // Set{}
```

Looping the elements of a JavaScript Set

A Set object maintains the insertion order of its elements, therefore, when you iterate over its elements, the order of the elements is the same as the inserted order. Suppose you have a set of user roles as follows.

```
let roles = new Set();
roles.add('admin')
    .add('editor')
    .add('subscriber');
```

The following example uses the for...of loop (https://www.javascripttutorial.net/es6/javascript-for-of/) to iterate over the chars set.

```
for (let role of roles) {
    console.log(role);
}
```

Output:

```
admin
editor
subscriber
```

The `Set` also provides the `keys(),` `values(),` and `entries()` methods like the Map (https://www.javascripttutorial.net/es6/javascript-map/) . However, keys and values in the `Set` are identical. For example:

```
for (let [key, value] of roles.entries()) {
    console.log(key === value);
}
```

Output

```
true
true
true
```

Invoke a callback function on each element of a set

If you want to invoke a callback (https://www.javascripttutorial.net/javascript-callback/) on every element of a set, you can use the `forEach()` method.

```
roles.forEach(role => console.log(role.toUpperCase()));
```

WeakSets

A `WeakSet` is similar to a `Set` except that it contains only objects. Since objects in a `WeakSet` may be automatically garbage-collected, a `WeakSet` does not have `size` property. Like a `WeakMap`, you cannot iterate elements of a `WeakSet`, therefore, you will find that WeakSet is rarely used in practice. In fact, you only use a `WeakSet` to check if a specified value is in the set. Here is an example:

```
let computer = {type: 'laptop'};
let server = {type: 'server'};
let equipment = new WeakSet([computer, server]);

if (equipment.has(server)) {
    console.log('We have a server');
}
```

Output

```
We have a server
```

In this tutorial, you have learned about the JavaScript `Set` object and how to manipulate its elements.