28th October 2019

Object.keys, values, entries

Let’s step away from the individual data structures and talk about the iterations over them.

In the previous chapter we saw methods `map.keys()` , `map.values()` , `map.entries()` .

These methods are generic, there is a common agreement to use them for data structures. If we ever create a data structure of our own, we should implement them too.

They are supported for:

- `Map`
- `Set`
- `Array`

Plain objects also support similar methods, but the syntax is a bit different.

Object.keys, values, entries

For plain objects, the following methods are available:

- Object.keys(obj) – returns an array of keys.
- Object.values(obj) – returns an array of values.
- Object.entries(obj) – returns an array of `[key, value]` pairs.

Please note the distinctions (compared to map for example):

	Map	**Object**
Call syntax	`map.keys()`	`Object.keys(obj)` , but not `obj.keys()`
Returns	iterable	“real” Array

The first difference is that we have to call `Object.keys(obj)` , and not `obj.keys()` .

Why so? The main reason is flexibility. Remember, objects are a base of all complex structures in JavaScript. So we may have an object of our own like `data` that implements its own `data.values()` method. And we still can call `Object.values(data)` on it.

The second difference is that `Object.*` methods return “real” array objects, not just an iterable. That’s mainly for historical reasons.

For instance:

```
let user = {
  name: "John",
  age: 30
};
```

- `Object.keys(user) = ["name", "age"]`
- `Object.values(user) = ["John", 30]`
- `Object.entries(user) = [ ["name","John"], ["age",30] ]`

Here's an example of using `Object.values` to loop over property values:

```
let user = {
  name: "John",
  age: 30
};

// loop over values
for (let value of Object.values(user)) {
  alert(value); // John, then 30
}
```

Object.keys/values/entries ignore symbolic properties

Just like a `for..in` loop, these methods ignore properties that use `Symbol(...)` as keys.

Usually that's convenient. But if we want symbolic keys too, then there's a separate method Object.getOwnPropertySymbols that returns an array of only symbolic keys. Also, there exist a method Reflect.ownKeys(obj) that returns *all* keys.

Transforming objects

Objects lack many methods that exist for arrays, e.g. `map` , `filter` and others.

If we'd like to apply them, then we can use `Object.entries` followed `Object.fromEntries` :

1. Use `Object.entries(obj)` to get an array of key/value pairs from `obj` .
2. Use array methods on that array, e.g. `map` .
3. Use `Object.fromEntries(array)` on the resulting array to turn it back into an object.

For example, we have an object with prices, and would like to double them:

```
let prices = {
  banana: 1,
  orange: 2,
  meat: 4,
};

let doublePrices = Object.fromEntries(
  // convert to array, map, and then fromEntries gives back the object
  Object.entries(prices).map(([key, value]) => [key, value * 2])
);

alert(doublePrices.meat); // 8
```

It may look difficult from the first sight, but becomes easy to understand after you use it once or twice. We can make powerful chains of transforms this way.

Tasks

Sum the properties

importance: 5

There is a `salaries` object with arbitrary number of salaries.

Write the function `sumSalaries(salaries)` that returns the sum of all salaries using `Object.values` and the `for..of` loop.

If `salaries` is empty, then the result must be `0`.

For instance:

```
let salaries = {
  "John": 100,
  "Pete": 300,
  "Mary": 250
};

alert( sumSalaries(salaries) ); // 650
```

Open a sandbox with tests.

solution

Count properties

importance: 5

Write a function `count(obj)` that returns the number of properties in the object:

```
let user = {
  name: 'John',
  age: 30
};

alert( count(user) ); // 2
```

Try to make the code as short as possible.

P.S. Ignore symbolic properties, count only “regular” ones.

Open a sandbox with tests.

solution

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)