23rd October 2019

Extending built-in classes

Built-in classes like Array, Map and others are extendable also.

For instance, here `PowerArray` inherits from the native `Array` :

```
// add one more method to it (can do more)
class PowerArray extends Array {
  isEmpty() {
    return this.length === 0;
  }
}

let arr = new PowerArray(1, 2, 5, 10, 50);
alert(arr.isEmpty()); // false

let filteredArr = arr.filter(item => item >= 10);
alert(filteredArr); // 10, 50
alert(filteredArr.isEmpty()); // false
```

Please note a very interesting thing. Built-in methods like `filter` , `map` and others – return new objects of exactly the inherited type `PowerArray` . Their internal implementation uses the object's `constructor` property for that.

In the example above,

```
arr.constructor === PowerArray
```

When `arr.filter()` is called, it internally creates the new array of results using exactly `arr.constructor` , not basic `Array` . That's actually very cool, because we can keep using `PowerArray` methods further on the result.

Even more, we can customize that behavior.

We can add a special static getter `Symbol.species` to the class. If it exists, it should return the constructor that JavaScript will use internally to create new entities in `map` , `filter` and so on.

If we'd like built-in methods like `map` or `filter` to return regular arrays, we can return `Array` in `Symbol.species` , like here:

```
class PowerArray extends Array {
  isEmpty() {
    return this.length === 0;
  }

  // built-in methods will use this as the constructor
  static get [Symbol.species]() {
    return Array;
  }
}

let arr = new PowerArray(1, 2, 5, 10, 50);
alert(arr.isEmpty()); // false

// filter creates new array using arr.constructor[Symbol.species] as construc
let filteredArr = arr.filter(item => item >= 10);

// filteredArr is not PowerArray, but Array
alert(filteredArr.isEmpty()); // Error: filteredArr.isEmpty is not a function
```

As you can see, now `.filter` returns `Array`. So the extended functionality is not passed any further.

Other collections work similarly

Other collections, such as `Map` and `Set`, work alike. They also use `Symbol.species`.

No static inheritance in built-ins

Built-in objects have their own static methods, for instance `Object.keys`, `Array.isArray` etc.

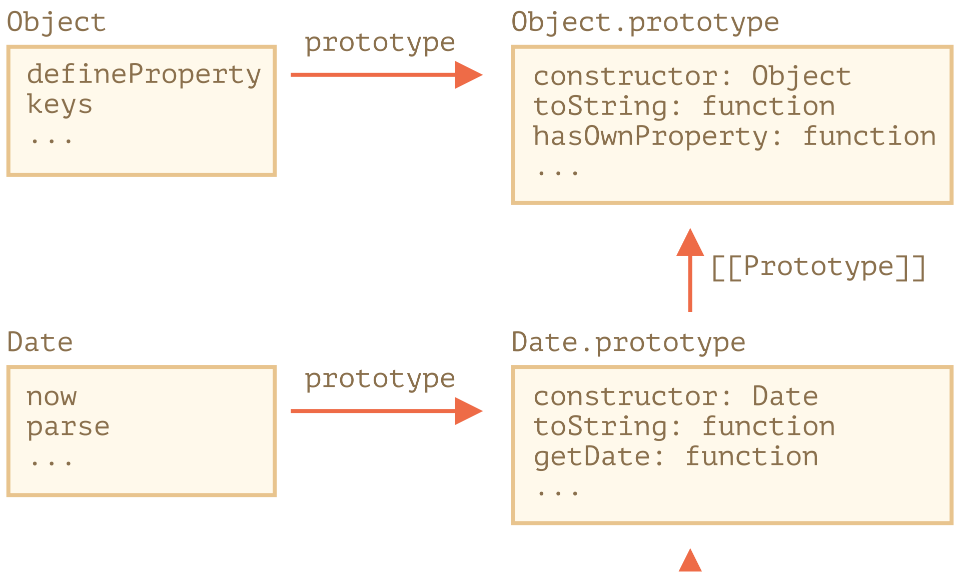
As we already know, native classes extend each other. For instance, `Array` extends `Object`.

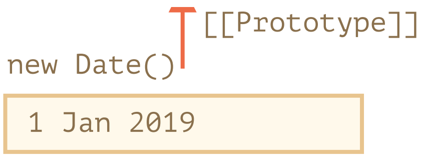
Normally, when one class extends another, both static and non-static methods are inherited. That was thoroughly explained in the article Static properties and methods.

But built-in classes are an exception. They don't inherit statics from each other.

For example, both `Array` and `Date` inherit from `Object`, so their instances have methods from `Object.prototype`. But `Array.[[Prototype]]` does not reference `Object`, so there's no, for instance, `Array.keys()` (or `Date.keys()`) static method.

Here's the picture structure for `Date` and `Object`:





As you can see, there’s no link between `Date` and `Object`. They are independent, only `Date.prototype` inherits from `Object.prototype`.

That’s an important difference of inheritance between built-in objects compared to what we get with `extends`.

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…)