Расширение встроенных классов

От встроенных классов, таких как `Array`, `Map` и других, тоже можно наследовать.

Например, в этом примере `PowerArray` наследуется от встроенного `Array`:

```
// добавим один метод (можно более одного)
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
```

Обратите внимание на интересный момент: встроенные методы, такие как `filter`, `map` и другие возвращают новые объекты унаследованного класса `PowerArray`. Их внутренняя реализация такова, что для этого они используют свойство объекта `constructor`.

В примере выше,

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

Поэтому при вызове метода `arr.filter()` он внутри создаёт массив результатов, именно используя `arr.constructor`, а не обычный массив. Это замечательно, поскольку можно продолжать использовать методы `PowerArray` далее на результатах.

Более того, мы можем настроить это поведение.

При помощи специального статического геттера `Symbol.species` можно вернуть конструктор, который JavaScript будет использовать в `filter`, `map` и других методах для создания новых объектов.

Если бы мы хотели, чтобы методы `map`, `filter` и т. д. возвращали обычные массивы, мы могли бы вернуть `Array` в `Symbol.species`, вот так:

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

  // встроенные методы массива будут использовать этот
  static get [Symbol.species]() {
    return Array;
  }
}

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

// filter создаст новый массив, используя arr.construct
let filteredArr = arr.filter(item => item >= 10);
```

```
17
18  // filteredArr не является PowerArray, это Array
19  alert(filteredArr.isEmpty()); // Error: filteredArr.isE
```

Как вы видите, теперь `.filter` возвращает `Array`. Расширенная функциональность не будет передаваться далее.

> **Аналогично работают другие коллекции**
>
> Другие коллекции, такие как `Map`, `Set`, работают аналогично. Они также используют `Symbol.species`.

Отсутствие статического наследования встроенных классов

У встроенных объектов есть собственные статические методы, например `Object.keys`, `Array.isArray` и т. д.

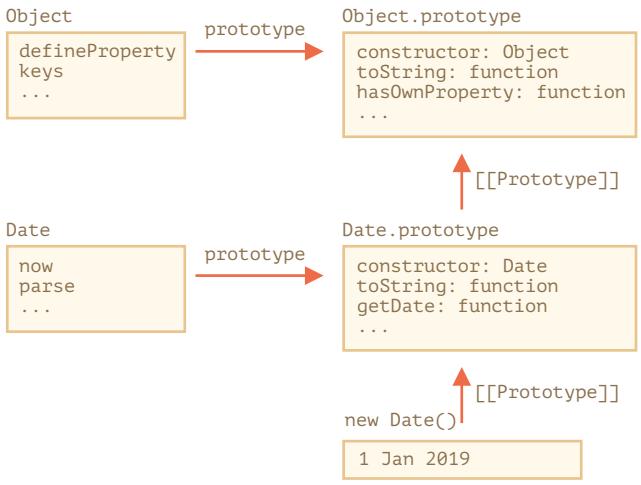
Как мы уже знаем, встроенные классы расширяют друг друга.

Обычно, когда один класс наследует другой, то наследуются и статические методы. Это было подробно разъяснено в главе Статические свойства и методы.

Но встроенные классы – исключение. Они не наследуют статические методы друг друга.

Например, и `Array`, и `Date` наследуют от `Object`, так что в их экземплярах доступны методы из `Object.prototype`. Но `Array.[[Prototype]]` не ссылается на `Object`, поэтому нет методов `Array.keys()` или `Date.keys()`.

Ниже вы видите структуру `Date` и `Object`:



Как видите, нет связи между `Date` и `Object`. Они независимы, только `Date.prototype` наследует от `Object.prototype`.

В этом важное отличие наследования встроенных объектов от того, что мы получаем с использованием `extends`.

Комментарии

перед тем как писать...