

Раздел

[Прототипы, наследование](#)

Навигация по уроку

Object.prototype

Другие встроенные прототипы

Примитивы

Изменение встроенных прототипов


Заимствование у прототипов

Итого

Задачи (2)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠](#) → [Язык программирования JavaScript](#)  
→ [Прототипы, наследование](#) 18-го февраля 2020

## Встроенные прототипы

Свойство "prototype" широко используется внутри самого языка JavaScript. Все встроенные функции-конструкторы используют его.

Сначала мы рассмотрим детали, а затем используем "prototype" для добавления встроенным объектам новой функциональности.

### Object.prototype

Давайте выведем пустой объект:

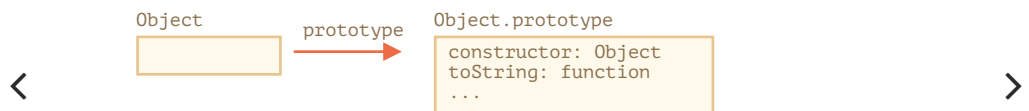
```
1 let obj = {};  
2 alert( obj ); // "[object Object]" ?
```



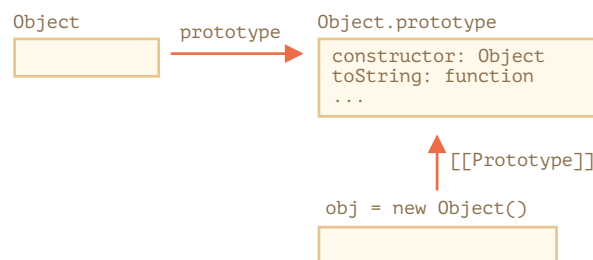
Где код, который генерирует строку "[object Object]" ? Это встроенный метод toString, но где он? obj ведь пуст!

...Но краткая нотация obj = {} – это то же самое, что и obj = new Object(), где Object – встроенная функция-конструктор для объектов с собственным свойством prototype, которое ссылается на огромный объект с методом toString и другими.

Вот что происходит:



Когда вызывается new Object() (или создаётся объект с помощью литерала {...}), свойство [[Prototype]] этого объекта устанавливается на Object.prototype по правилам, которые мы обсуждали в предыдущей главе:



Таким образом, когда вызывается obj.toString(), метод берётся из Object.prototype.

Мы можем проверить это так:

```
1 let obj = {};  
2  
3 alert(obj.__proto__ === Object.prototype); // true  
4 // obj.toString === obj.__proto__.toString == Object.pr
```



Обратите внимание, что по цепочке прототипов выше Object.prototype больше нет свойства [[Prototype]]:

```
1 alert(Object.prototype.__proto__); // null
```



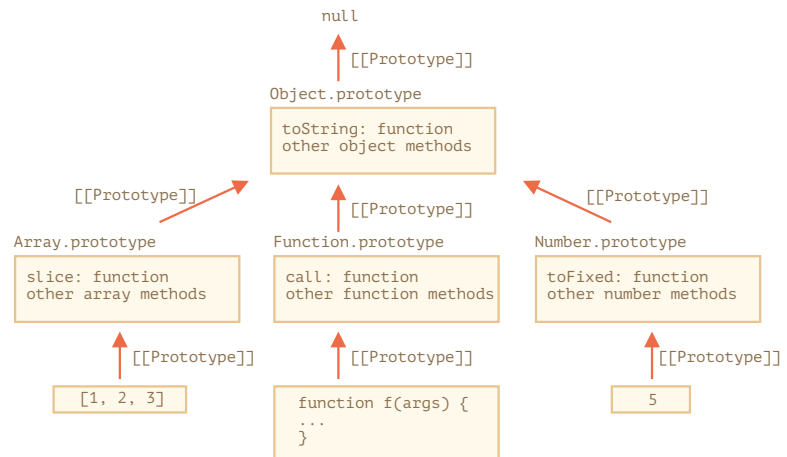
## Другие встроенные прототипы

Другие встроенные объекты, такие как `Array`, `Date`, `Function` и другие, также хранят свои методы в прототипах.

Например, при создании массива `[1, 2, 3]` внутренне используется конструктор массива `Array`. Поэтому прототипом массива становится `Array.prototype`, предоставляя ему свои методы. Это позволяет эффективно использовать память.

Согласно спецификации, наверху иерархии встроенных прототипов находится `Object.prototype`. Поэтому иногда говорят, что «всё наследует от объектов».

Вот более полная картина (для трёх встроенных объектов):



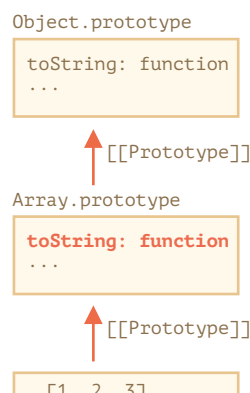
Давайте проверим прототипы:

```
1 let arr = [1, 2, 3];
2
3 // наследует ли от Array.prototype?
4 alert( arr.__proto__ === Array.prototype ); // true
5
6 // затем наследует ли от Object.prototype?
7 alert( arr.__proto__.__proto__ === Object.prototype );
8
9 // и null на вершине иерархии
10 alert( arr.__proto__.__proto__.__proto__ ); // null
```

Некоторые методы в прототипах могут пересекаться, например, у `Array.prototype` есть свой метод `toString`, который выводит элементы массива через запятую:

```
1 let arr = [1, 2, 3]
2 alert(arr); // 1,2,3 <-- результат Array.prototype.toSt
```

Как мы видели ранее, у `Object.prototype` есть свой метод `toString`, но так как `Array.prototype` ближе в цепочке прототипов, то берётся именно вариант для массивов:



Раздел

Прототипы, наследование

Навигация по уроку

Object.prototype

Другие встроенные прототипы

Примитивы

Изменение встроенных прототипов

Заимствование у прототипов

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



В браузерных инструментах, таких как консоль разработчика, можно посмотреть цепочку наследования (возможно, потребуется использовать `console.dir` для встроенных объектов):

```
> console.dir([1,2,3])
▼ Array[3] ⓘ
  0: 1
  1: 2
  2: 3
  length: 3
  __proto__: Array.prototype
  ▶ concat: function concat() { [native code] }
  ▶ ...
  ▶ unshift: function unshift() { [native code] }
  ▼ __proto__: Object.prototype
    ▶ ...
    ▶ constructor: function Object() { [native code] }
    ▶ hasOwnProperty: function hasOwnProperty() { [native code] }
    ▶ isPrototypeOf: function isPrototypeOf() { [native code] }
    ▶ ...
```

Другие встроенные объекты устроены аналогично. Даже функции – они объекты встроенного конструктора `Function`, и все их методы (`call` / `apply` и другие) берутся из `Function.prototype`. Также у функций есть свой метод `toString`.

```
1 function f() {}
2
3 alert(f.__proto__ == Function.prototype); // true
4 alert(f.__proto__.__proto__ == Object.prototype); // tr
```

## Примитивы

Самое сложное происходит со строками, числами и булевыми значениями.

Как мы помним, они не объекты. Но если мы попытаемся получить доступ к их свойствам, то тогда будет создан временный объект-обёртка с использованием встроенных конструкторов `String`, `Number` и `Boolean`, который предоставит методы и после этого исчезнет.

Эти объекты создаются невидимо для нас, и большая часть движков оптимизирует этот процесс, но спецификация описывает это именно таким образом. Методы этих объектов также находятся в прототипах, доступных как `String.prototype`, `Number.prototype` и `Boolean.prototype`.

### ⚠ Значения `null` и `undefined` не имеют объектов-обёрток

Специальные значения `null` и `undefined` стоят особняком. У них нет объектов-обёрток, так что методы и свойства им недоступны. Также у них нет соответствующих прототипов.

## Изменение встроенных прототипов

Встроенные прототипы можно изменять. Например, если добавить метод к `String.prototype`, метод становится доступен для всех строк:

```
1 String.prototype.show = function() {
2   alert(this);
3 };
4
5 "BOOM!".show(); // BOOM!
```

В течение процесса разработки у нас могут возникнуть идеи о новых встроенных методах, которые нам хотелось бы иметь, и искушение добавить их во встроенные прототипы. Это плохая идея.

Раздел

[Прототипы, наследование](#)

Навигация по уроку

Object.prototype

Другие встроенные прототипы

Примитивы

Изменение встроенных прототипов

Заимствование у прототипов

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



#### ⚠ Важно:

Прототипы глобальны, поэтому очень легко могут возникнуть конфликты. Если две библиотеки добавляют метод `String.prototype.show`, то одна из них перепишет метод другой.

Так что, в общем, изменение встроенных прототипов считается плохой идеей.

**В современном программировании есть только один случай, в котором одобряется изменение встроенных прототипов. Это создание полифилов.**

Полифил – это термин, который означает эмуляцию метода, который существует в спецификации JavaScript, но ещё не поддерживается текущим движком JavaScript.

Тогда мы можем реализовать его сами и добавить во встроенный прототип.

Например:

```
1 if (!String.prototype.repeat) { // Если такого метода н
2   // добавляем его в прототип
3
4   String.prototype.repeat = function(n) {
5     // повторить строку n раз
6
7     // на самом деле код должен быть немного более слож
8     // (полный алгоритм можно найти в спецификации)
9     // но даже неполный полифил зачастую достаточно хор
10    return new Array(n + 1).join(this);
11  };
12 }
13
14 alert( "La".repeat(3) ); // LaLaLa
```



## Заимствование у прототипов

В главе [Декораторы и переадресация вызова, call/apply](#) мы говорили о заимствовании методов.

Это когда мы берём метод из одного объекта и копируем его в другой.

Некоторые методы встроенных прототипов часто одалживают.

Например, если мы создаём объект, похожий на массив (псевдомассив), мы можем скопировать некоторые методы из `Array` в этот объект.

Пример:

```
1 let obj = {
2   0: "Hello",
3   1: "world!",
4   length: 2,
5 };
6
7 obj.join = Array.prototype.join;
8
9 alert( obj.join(',') ); // Hello,world!
```

Это работает, потому что для внутреннего алгоритма встроенного метода `join` важны только корректность индексов и свойство `length`, он не проверяет, является ли объект на самом деле массивом. И многие встроенные методы работают так же.

Альтернативная возможность – мы можем унаследовать от массива, установив `obj.__proto__` как `Array.prototype`, таким образом все методы `Array` станут автоматически доступны в `obj`.

Но это будет невозможно, если `obj` уже наследует от другого объекта. Помните, мы можем наследовать только от одного объекта одновременно.

Заимствование методов – гибкий способ, позволяющий смешивать функциональность разных объектов по необходимости.

## Итого

- Все встроенные объекты следуют одному шаблону:
  - Методы хранятся в прототипах (`Array.prototype`, `Object.prototype`, `Date.prototype` и т.д.).
  - Сами объекты хранят только данные (элементы массивов, свойства объектов, даты).
- Прimitives также хранят свои методы в прототипах объектов-обёрток: `Number.prototype`, `String.prototype`, `Boolean.prototype`. Только у значений `undefined` и `null` нет объектов-обёрток.
- Встроенные прототипы могут быть изменены или дополнены новыми методами. Но не рекомендуется менять их. Единственная допустимая причина – это добавление нового метода из стандарта, который ещё не поддерживается движком JavaScript.

## ✓ Задачи

### Добавить функциям метод "f.defer(ms)"

важность: 5

Добавьте всем функциям в прототип метод `defer(ms)`, который вызывает функции через `ms` миллисекунд.

После этого должен работать такой код:

```
1 function f() {
2   alert("Hello!");
3 }
4
5 f.defer(1000); // выведет "Hello!" через 1 секунду
```

решение

### Добавьте функциям декорирующий метод "defer()"

важность: 4

Добавьте всем функциям в прототип метод `defer(ms)`, который возвращает обёртку, откладывающую вызов функции на `ms` миллисекунд.

Например, должно работать так:

```
1 function f(a, b) {
2   alert( a + b );
3 }
4
5 f.defer(1000)(1, 2); // выведет 3 через 1 секунду.
```

Пожалуйста, заметьте, что аргументы должны корректно передаваться оригинальной функции.

решение

Проводим [курсы по JavaScript и фреймворкам.](#) ✕

## 💬 Комментарии

перед тем как писать...

Раздел

[Прототипы, наследование](#)

Навигация по уроку

[Object.prototype](#)

[Другие встроенные прототипы](#)

[Прimitives](#)

[Изменение встроенных прототипов](#)

[Заимствование у прототипов](#)

[Итого](#)

[Задачи \(2\)](#)

[Комментарии](#)

[Поделиться](#)



[Редактировать на GitHub](#)

Раздел

[Прототипы, наследование](#)

Навигация по уроку

Object.prototype

Другие встроенные  
прототипы

Примитивы

Изменение встроенных  
прототипов

Заимствование у прототипов

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)

