

Раздел

[Типы данных](#)

Навигация по уроку


Примитив как объект

Итого

Задачи (1)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Типы данных](#) 30-го марта 2020

Методы у примитивов

JavaScript позволяет нам работать с примитивными типами данных – строками, числами и т.д., как будто они являются объектами. У них есть и методы. Мы изучим их позже, а сначала разберём, как это всё работает, потому что, конечно, примитивы – не объекты.

Давайте взглянем на ключевые различия между примитивами и объектами.

Примитив

- Это – значение «примитивного» типа.
- Есть 7 примитивных типов: `string`, `number`, `boolean`, `symbol`, `null`, `undefined` и `bigint`.

Объект

- Может хранить множество значений как свойства.
- Объявляется при помощи фигурных скобок `{}`, например: `{name: "Рома", age: 30}`. В JavaScript есть и другие виды объектов: например, функции тоже являются объектами.

Одна из лучших особенностей объектов – это то, что мы можем хранить функцию как одно из свойств объекта.

```
1 let roma = {  
2   name: "Рома",  
3   sayHi: function() {  
4     alert("Привет, дружище!");  
5   }  
6 };  
7  
8 roma.sayHi(); // Привет, дружище!
```

Здесь мы создали объект `roma` с методом `sayHi`.

Существует множество встроенных объектов. Например, те, которые работают с датами, ошибками, HTML-элементами и т.д. Они имеют различные свойства и методы.

Однако у этих возможностей есть обратная сторона!

Объекты «тяжелее» примитивов. Они нуждаются в дополнительных ресурсах для поддержания внутренней структуры.

Примитив как объект

Вот парадокс, с которым столкнулся создатель JavaScript:

- Есть много всего, что хотелось бы сделать с примитивами, такими как строка или число. Было бы замечательно, если бы мы могли работать с ними через вызовы методов.
- Примитивы должны быть лёгкими и быстрыми.

Выбранное решение, хотя выглядит оно немного неуклюже:

1. Примитивы остаются примитивами. Одно значение, как и хотелось.
2. Язык позволяет осуществлять доступ к методам и свойствам строк, чисел, булевых значений и символов.
3. Чтобы это работало, при таком доступе создаётся специальный «объект-обёртка», который предоставляет нужную функциональность, а после удаляется.

Раздел

[Типы данных](#)

Навигация по уроку

Примитив как объект

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Каждый примитив имеет свой собственный «объект-обёртку», которые называются: `String`, `Number`, `Boolean` и `Symbol`. Таким образом, они имеют разный набор методов.

К примеру, существует метод `str.toUpperCase()`, который возвращает строку в верхнем регистре.

Вот, как он работает:

```
1 let str = "Привет";
2
3 alert( str.toUpperCase() ); // ПРИВЕТ
```



Очень просто, не правда ли? Вот, что на самом деле происходит в `str.toUpperCase()`:

1. Строка `str` – примитив. В момент обращения к его свойству, создаётся специальный объект, который знает значение строки и имеет такие полезные методы, как `toUpperCase()`.
2. Этот метод запускается и возвращает новую строку (показывается в `alert()`).
3. Специальный объект удаляется, оставляя только примитив `str`.

Получается, что примитивы могут предоставлять методы, и в то же время оставаться «лёгкими».

Движок JavaScript сильно оптимизирует этот процесс. Он даже может пропустить создание специального объекта. Однако, он всё же должен придерживаться спецификаций и работать так, как будто он его создаёт.

Число имеет собственный набор методов. Например, `toFixed(n)` округляет число до `n` знаков после запятой.

```
1 let n = 1.23456;
2
3 alert( n.toFixed(2) ); // 1.23
```



Более подробно с различными свойствами и методами мы познакомимся в главах [Числа](#) и [Строки](#).

Раздел

[Типы данных](#)

Навигация по уроку

Примитив как объект

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



⚠ Конструкторы String/Number/Boolean предназначены только для внутреннего пользования

Некоторые языки, такие как Java, позволяют явное создание «объектов-обёрток» для примитивов при помощи такого синтаксиса как `new Number(1)` или `new Boolean(false)`.

В JavaScript, это тоже возможно по историческим причинам, но очень **не рекомендуется**. В некоторых местах последствия могут быть катастрофическими.

Например:

```
1 alert( typeof 0 ); // "число"
2
3 alert( typeof new Number(0) ); // "object"!
```

Объекты в `if` всегда дают `true`, так что в нижеприведённом примере будет показан `alert`:

```
1 let zero = new Number(0);
2
3 if (zero) {
4   // zero возвращает "true", так как является объе
5   alert( "zero имеет «истинное» значение?!?" );
6 }
```

С другой стороны, использование функций `String/Number/Boolean` без оператора `new` – вполне разумно и полезно. Они превращают значение в соответствующий примитивный тип: в строку, в число, в булевый тип.

К примеру, следующее вполне допустимо:

```
1 let num = Number("123"); // превращает строку в чи
```

⚠ `null/undefined` не имеют методов

Особенные примитивы `null` и `undefined` являются исключениями. У них нет соответствующих «объектов-обёрток», и они не имеют никаких методов. В некотором смысле, они «самые примитивные».

Попытка доступа к свойствам такого значения возвратит ошибку:

```
1 alert(null.test); // ошибка
```

Итого

- Все примитивы, кроме `null` и `undefined`, предоставляют множество полезных методов. Мы познакомимся с ними поближе в следующих главах.
- Формально эти методы работают с помощью временных объектов, но движки JavaScript внутренне очень хорошо оптимизируют этот процесс, так что их вызов не требует много ресурсов.

✓ Задачи

Можно ли добавить свойство строке? [↗](#)

важность: 5

Раздел

[Типы данных](#)

Навигация по уроку

Примитив как объект

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



Взгляните на следующий код:

```
1 let str = "Привет";
2
3 str.test = 5;
4
5 alert(str.test);
```

Как вы думаете, это сработает? Что выведется на экран?

решение

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



Комментарии

перед тем как писать...

