



```
👚 → Язык JavaScript → Свойства объекта, их конфигурация
```

Свойства - геттеры и сеттеры

Есть два типа свойств объекта.

Первый тип это *свойства-данные* (data properties). Мы уже знаем, как работать с ними. Все свойства, которые мы использовали до текущего момента, были свойствами-данными.

Второй тип свойств мы ещё не рассматривали. Это *свойства-аксессоры* (accessor properties). По своей сути это функции, которые используются для присвоения и получения значения, но во внешнем коде они выглядят как обычные свойства объекта.

Геттеры и сеттеры

Свойства-аксессоры представлены методами: «геттер» – для чтения и «сеттер» – для записи. При литеральном объявлении объекта они обозначаются get и set:

```
1 let obj = {
2  get propName() {
3    // геттер, срабатывает при чтении obj.propName
4  },
5
6  set propName(value) {
7    // сеттер, срабатывает при записи obj.propName = value
8  }
9 };
```

Геттер срабатывает, когда оbj.propName читается, сеттер – когда значение присваивается.

Например, у нас есть объект user со свойствами name и surname:

```
1 let user = {
2   name: "John",
3   surname: "Smith"
4 };
```

А теперь добавим свойство объекта fullName для полного имени, которое в нашем случае "John Smith". Само собой, мы не хотим дублировать уже имеющуюся информацию, так что реализуем его при помощи аксессора:

```
1 let user = {
2   name: "John",
3   surname: "Smith",
4
```



```
5  get fullName() {
6   return `${this.name} ${this.surname}`;
7  }
8  };
9
10 alert(user.fullName); // John Smith
```

Снаружи свойство-аксессор выглядит как обычное свойство. В этом и заключается смысл свойств-аксессоров. Мы не *вызываем user.fullName* как функцию, а *читаем* как обычное свойство: геттер выполнит всю работу за кулисами.

На данный момент у fullName есть только геттер. Если мы попытаемся назначить user.fullName=, произойдёт ошибка:

```
1 let user = {
2   get fullName() {
3    return `...`;
4  }
5 };
6
7 user.fullName = "Тест"; // Ошибка (у свойства есть только геттер)
```

Давайте исправим это, добавив сеттер для user.fullName:

```
1 let user = {
2
    name: "John",
3
     surname: "Smith",
4
  get fullName() {
5
6
      return `${this.name} ${this.surname}`;
7
    },
8
     set fullName(value) {
9
      [this.name, this.surname] = value.split(" ");
10
11
     }
12
  };
13
  // set fullName запустится с данным значением
15 user.fullName = "Alice Cooper";
16
17 alert(user.name); // Alice
18 alert(user.surname); // Cooper
```

В итоге мы получили «виртуальное» свойство fullName. Его можно прочитать и изменить.

Дескрипторы свойств доступа

Дескрипторы свойств-аксессоров отличаются от «обычных» свойств-данных.

Свойства-аксессоры не имеют value и writable, но взамен предлагают функции get и set.

То есть, дескриптор аксессора может иметь:

- get функция без аргументов, которая сработает при чтении свойства,
- set функция, принимающая один аргумент, вызываемая при присвоении свойства,
- enumerable то же самое, что и для свойств-данных,
- configurable то же самое, что и для свойств-данных.

Например, для создания аксессора fullName при помощи defineProperty мы можем передать дескриптор с использованием get и set:

```
1 let user = {
   name: "John",
2
     surname: "Smith"
3
4
  };
5
6 Object.defineProperty(user, 'fullName', {
7
     get() {
8
       return `${this.name} ${this.surname}`;
9
     },
10
11
     set(value) {
       [this.name, this.surname] = value.split(" ");
12
13
14
   });
15
16
   alert(user.fullName); // John Smith
17
18
   for(let key in user) alert(key); // name, surname
```

Ещё раз заметим, что свойство объекта может быть либо свойством-аксессором (с методами get/set), либо свойством-данным (со значением value).

При попытке указать и get, и value в одном дескрипторе будет ошибка:

```
1 // Error: Invalid property descriptor.
2 Object.defineProperty({}, 'prop', {
3    get() {
4      return 1
5    },
6
7   value: 2
8 });
```

Умные геттеры/сеттеры

Геттеры/сеттеры можно использовать как обёртки над «реальными» значениями свойств, чтобы получить больше контроля над операциями с ними.

Например, если мы хотим запретить устанавливать короткое имя для user, мы можем использовать сеттер name для проверки, а само значение хранить в отдельном свойстве $_name$:

```
1 let user = {
2  get name() {
```

```
3
       return this. name;
4
     },
 5
 6
     set name(value) {
7
        if (value.length < 4) {</pre>
8
          alert("Имя слишком короткое, должно быть более 4 символов");
9
          return:
10
11
        this. name = value;
12
     }.
13
   };
14
15
   user.name = "Pete";
16 alert(user.name); // Pete
17
18 user.name = ""; // Имя слишком короткое...
```

Таким образом, само имя хранится в _name , доступ к которому производится через геттер и сеттер.

Технически, внешний код всё ещё может получить доступ к имени напрямую с помощью user._name, но существует широко известное соглашение о том, что свойства, которые начинаются с символа "_", являются внутренними, и к ним не следует обращаться из-за пределов объекта.

Использование для совместимости

У аксессоров есть интересная область применения – они позволяют в любой момент взять «обычное» свойство и изменить его поведение, поменяв на геттер и сеттер.

Например, представим, что мы начали реализовывать объект user , используя свойства-данные имя name и возраст age :

```
1 function User(name, age) {
2   this.name = name;
3   this.age = age;
4 }
5   
6 let john = new User("John", 25);
7   
8 alert( john.age ); // 25
```

...Но рано или поздно всё может измениться. Взамен возраста age мы можем решить хранить дату рождения birthday, потому что так более точно и удобно:

```
1 function User(name, birthday) {
2   this.name = name;
3   this.birthday = birthday;
4 }
5
6 let john = new User("John", new Date(1992, 6, 1));
```

Что нам делать со старым кодом, который использует свойство age?

Мы можем попытаться найти все такие места и изменить их, но это отнимает время и может быть невыполнимо, если код используется другими людьми. И кроме того, age - это отличное свойство для user, верно?

Давайте его сохраним.

Добавление геттера для age решит проблему:

```
function User(name, birthday) {
1
2
     this.name = name;
3
     this.birthday = birthday;
4
5
     // возраст рассчитывается из текущей даты и дня рождения
6
     Object.defineProperty(this, "age", {
       get() {
7
8
         let todayYear = new Date().getFullYear();
9
         return todayYear - this.birthday.getFullYear();
10
11
     });
   }
12
13
   let john = new User("John", new Date(1992, 6, 1));
14
15
16 alert( john.birthday ); // доступен как день рождения
17 alert( john.age ); // ...так и возраст
```

Теперь старый код тоже работает, и у нас есть отличное дополнительное свойство!



Поделиться









Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег <code>, для нескольких строк кода тег , если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье пишите, что именно и с какого места.



Присоединиться к обсуждению.... войти с помощью или через disqus ? Имя

Орий месяц назад

Лучшие Новые Старые
— №

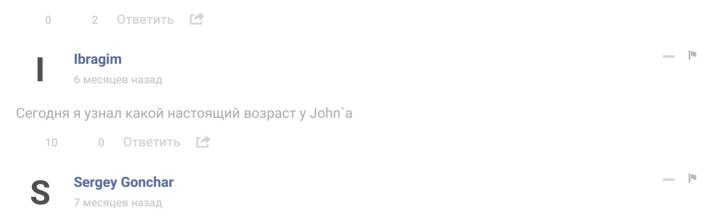
Хочешь разобраться зачем нужны геттеры и сеттеры?

Тогда переходи в ТГ-блог «Джун на фронте»!

ABTOP - системный администратор, который с декабря 2021 года освоил HTML, CSS, JS, Vue, Nuxt, React Native, MongoDB и Node.js.

Следи за моим путем в мир разработки: от новичка до создателя 🔖 Телеграм-бота для автоматической отправки откликов!

Вбивайте «Джун на фронте» и присоединяйтесь к нам!



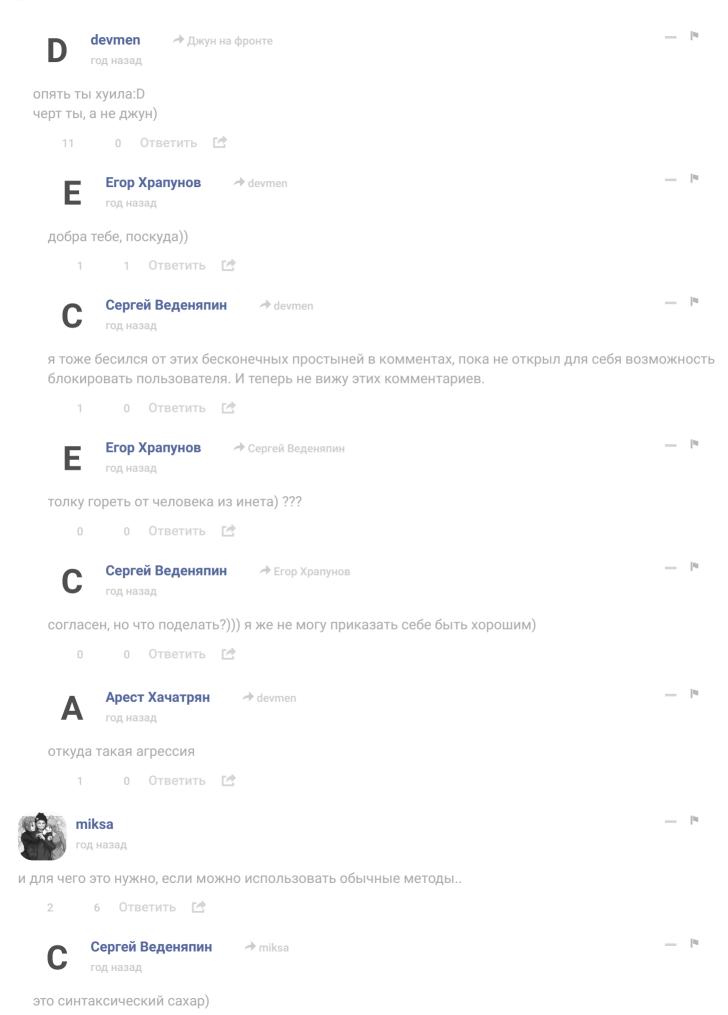
Возможно я слишком стар для всего этого дерьма плохо соображаю, но мне остается неясным, зачем функции маскировать под свойства? Разве это не создает путаницу в чтении кода?

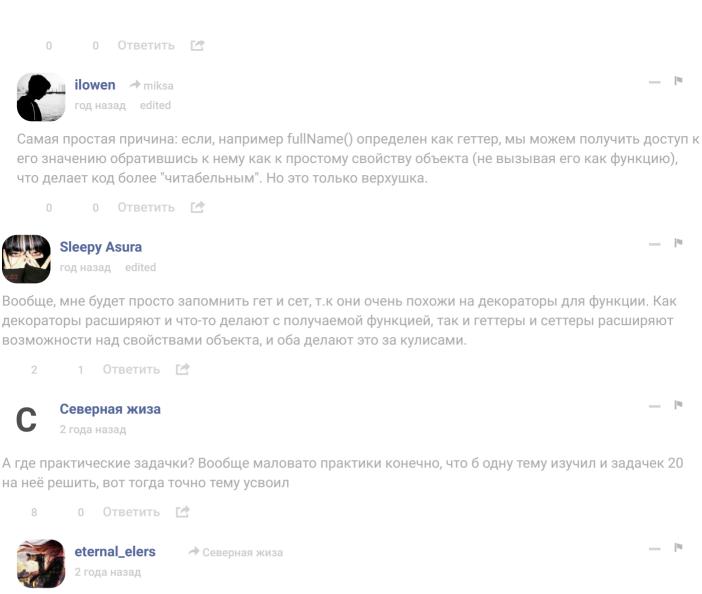


Не стоит потокать своему СПГС. Решили вот такую фичу добавить в язык, по аналогии со всеми ООП языками, вот и все.

0 Ответить 🖆





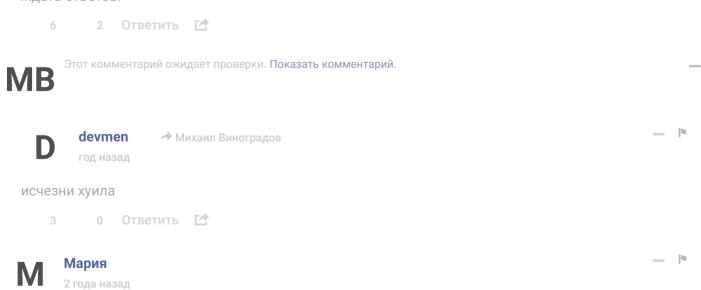


на неё решить, вот тогда точно тему усвоил

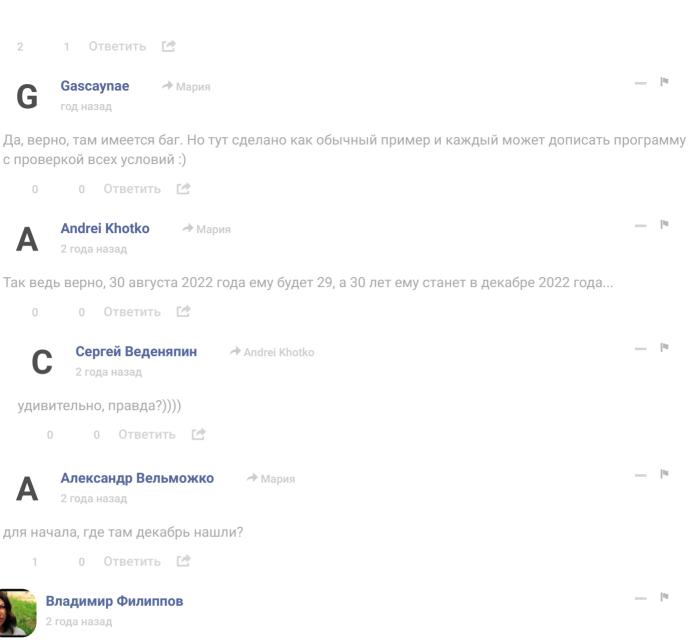
Ну здесь тема крайне легкая в усвоении и, уж темболее, понимании. А вот без задачек в Замыканиях или Регулярных выражениях было бы посложнее.

Главное - всегда пишите от самостоятельно код из примеров. Так вы будете блок за блоком писать код и смотреть на него как на отдельные элементы, попутно анализируя их работу, абстрагируясь от конечного варианта.

А задачки - тут Вам нужны полет фантазии и немного гугления. Последнему нужно учиться сразу, а не ждать ответов.



Хм, последний пример про возраст выходит с небольшим багом) если бы Джон родился в декабре 1992, то сегодня 30 августа, ответ был бы не "30")

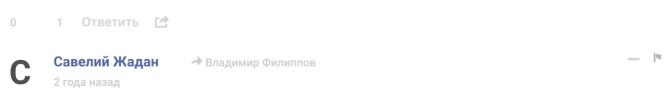




сеттер name для проверки, а само значение хранить в отдельном свойстве _name:

Это зачем?

Правильно ли я понял, что Pete из примера будет хранится, но в сам объект не попадёт? Не понимаю зачем здесь _



Pete из примера будет храниться в объекте в свойстве _name, так как мы задаем его в set(value) в строчке this._name = value

Идея состоит в том, что к таким свойствам не принято обращаться напрямую и нужны они просто, чтобы хранить значение



Ulluuriou.

0 0 Ответить 🗠

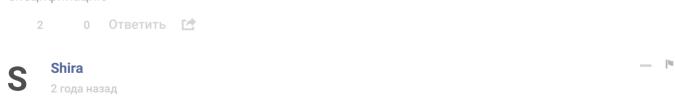


Почему дескриптор свойства-аксессора (name) отображается как свойство-данные?

```
'name': {
   'value': 'Pete',
   'writable': true,
   'enumerable': true,
   'configurable': true
},
 '_name': {
   'value': 'Pete'.
   'writable': true,
   'enumerable': true,
   'configurable': true
let user = {
  get name() {
                                             показать больше
          Ответить 🔼
        Aleksandr Volkov
                              → Кирилл
        2 года назад
```

100

Да, почему-то при использовании метода Object.assign копируется как свойство-данные. При копировании через метод из прошлого урока let clone = Object.defineProperties({}, Object.getOwnPropertyDescriptors(user) такого не происходит и свойство-аксессор копируется как свойство-аксессор. Судя по всему, у методов заложено разное поведение, нужно углубляться в спецификацию



Такс, почитал вроде сегодня немного можно и отдохнуть, лето же! Ребят не забрасывайте, но и не забывайте что нельзя сидеть целыми днями на этом сайте, саморазвитие это хорошо, но и отдых тоже



отдых? если для тебя тяжело читать текст который всем приносит удовольствие то тебе нужно обернуться на свой путь жизни и посмотреть где ты свернул не туда ,чел. Всем интересно читать как школьникам смотреть тик ток



Мне приносит удовольствие чтение, но у меня нагорало от дз, которое иногда сделать сложно. Многим не интересно читать, а интереснее делать практику, как и мне))

4 0 Ответить



Не совсем понял фразу "При попытке указать и get, и value в одном дескрипторе будет ошибка". Имеется в виду, что мы не можем в одном defineProperty и добавить геттер/сеттер и установить значение данного свойства?



Что здесь может быть непонятного? При создании нового свойства с помощью defineProperty может быть созданы либо свойства-данные, либо свойства-аксессоры. Либо то, либо другое, но не оба вместе.



Смотри, свойства-аксессоры не имеют стандартного value, это по сути функции, обернутые в свойство. Ты можешь их прописывать так, что при обращении к "свойству" get будет что-то возвращать, а при присваивании set - что-то устанавливать



Это я понимаю. Только не понимаю, как читать эту мудреную фразу.

Геттер в любом случае не позволяет устанавливать значения, а только получать их из свойства (не важно, "реального" или аксессора).



Обрати внимание, что ты еще и writable не можешь поставить вместе с get/set.

А все потому что у тебя свойство может быть одним из двух вариантов:

- 1)Свойство-значение: просто обычное свойство с value
- 2)Свойство-аксессор: оно по сути работает "посредником", не имеет своего value, его соответственно невозможно перезаписать (ведь все присваивания идут через сеттер), поэтому для него и writable не работает
 - 2 0 Ответить ГА



Понимаю, что не в тему, да и мало кому это будет полезным, но я рад, что не я один слушаю Элиса Купера



некоторые люди радуются жизни,некоторые богатству,но этот человек обошел всех их стороной,он рад что слушает Элиса Купера



Я же написал, что радуюсь, что слушаю его не один. Всегда же приятно, когда кто-то ещё кроме тебя слушает Элиса Купера.

3 0 Ответить 🗠

2 года назад

Загрузить ещё комментарии

Подписаться О защите персональных данных

Не продавайте мои данные

 Канторо проектес	связаться с нами	ипользовательс	ское соглашение	э политика конфи	и денциальност