

Раздел

[Свойства объекта, их конфигурация](#)

Навигация по уроку

Геттеры и сеттеры


Дескрипторы свойств доступа

Умные геттеры/сеттеры

Использование для совместимости

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Свойства объекта, их конфигурация](#) 29-го ноября 2019

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 = va
8   }
9 };
```

Геттер срабатывает, когда `obj.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.

Раздел

Свойства объекта, их конфигурация

Навигация по уроку

Геттеры и сеттеры

Дескрипторы свойств доступа

Умные геттеры/сеттеры

Использование для совместимости

Комментарии

Поделиться



Редактировать на GitHub



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

Давайте исправим это, добавив сеттер для `user.fullName`:

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

Раздел

[Свойства объекта, их конфигурация](#)

Навигация по уроку

Геттеры и сеттеры

Дескрипторы свойств доступа

Умные геттеры/сеттеры

Использование для совместимости

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
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) {
8       alert("Имя слишком короткое, должно быть более 4");
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 }
```

Раздел

[Свойства объекта, их конфигурация](#)

Навигация по уроку

Геттеры и сеттеры

Дескрипторы свойств доступа

Умные геттеры/сеттеры

Использование для совместимости

Комментарии

Поделиться



Редактировать на GitHub



```
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:

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

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

Комментарии

перед тем как писать...