**Дескрипторы, геттеры и сеттеры свойств**

В этой главе мы рассмотрим возможности, которые позволяют очень гибко и мощно управлять всеми свойствами объекта, включая их аспекты – изменяемость, видимость в цикле for..in и даже незаметно делать их функциями.

Они поддерживаются всеми современными браузерами, но не IE8-. Впрочем, даже IE8 их поддерживает, но только для DOM-объектов (используются при работе со страницей, это сейчас вне нашего рассмотрения).

**[Дескрипторы в примерах](http://learn.javascript.ru/descriptors-getters-setters" \l "deskriptory-v-primerah)**

Основной метод для управления свойствами – [Object.defineProperty](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperty).

Он позволяет объявить свойство объекта и, что самое главное, тонко настроить его особые аспекты, которые никак иначе не изменить.

Синтаксис:

Object.defineProperty(obj, prop, descriptor)

Аргументы:

**obj**

Объект, в котором объявляется свойство.

**prop**

Имя свойства, которое нужно объявить или модифицировать.

**descriptor**

Дескриптор – объект, который описывает поведение свойства.

В нём могут быть следующие поля:

* value – значение свойства, по умолчанию undefined
* writable – значение свойства можно менять, если true. По умолчанию false.
* configurable – если true, то свойство можно удалять, а также менять его в дальнейшем при помощи новых вызовов defineProperty. По умолчанию false.
* enumerable – если true, то свойство просматривается в цикле for..in и методе Object.keys(). По умолчанию false.
* get – функция, которая возвращает значение свойства. По умолчанию undefined.
* set – функция, которая записывает значение свойства. По умолчанию undefined.

Чтобы избежать конфликта, запрещено одновременно указывать значение value и функции get/set. Либо значение, либо функции для его чтения-записи, одно из двух. Также запрещено и не имеет смысла указывать writable при наличии get/set-функций.

Далее мы подробно разберём эти свойства на примерах.

**[Обычное свойство](http://learn.javascript.ru/descriptors-getters-setters" \l "obychnoe-svoystvo)**

Два таких вызова работают одинаково:

var user = {};

// 1. простое присваивание

user.name = "Вася";

// 2. указание значения через дескриптор

Object.defineProperty(user, "name", { value: "Вася", configurable: true, writable: true, enumerable: true });

Оба вызова выше добавляют в объект user обычное (удаляемое, изменяемое, перечисляемое) свойство.

**[Свойство-константа](http://learn.javascript.ru/descriptors-getters-setters" \l "svoystvo-konstanta)**

Для того, чтобы сделать свойство неизменяемым, изменим его флаги writable и configurable:

"use strict";

var user = {};

Object.defineProperty(user, "name", {

value: "Вася",

writable: false, // запретить присвоение "user.name="

configurable: false // запретить удаление "delete user.name"

});

// Теперь попытаемся изменить это свойство.

// в strict mode присвоение "user.name=" вызовет ошибку

user.name = "Петя";

Заметим, что без use strict операция записи «молча» не сработает. Лишь если установлен режим use strict, то дополнительно сгенерируется ошибка.

**[Свойство, скрытое для for…in](http://learn.javascript.ru/descriptors-getters-setters" \l "svoystvo-skrytoe-dlya-for-in)**

Встроенный метод toString, как и большинство встроенных методов, не участвует в цикле for..in. Это удобно, так как обычно такое свойство является «служебным».

К сожалению, свойство toString, объявленное обычным способом, будет видно в цикле for..in, например:

var user = {

name: "Вася",

toString: function() { return this.name; }

};

for(var key in user) alert(key); // name, toString

Мы бы хотели, чтобы поведение нашего метода toString было таким же, как и стандартного.

Object.defineProperty может исключить toString из списка итерации, поставив ему флаг enumerable: false. По стандарту, у встроенного toString этот флаг уже стоит.

var user = {

name: "Вася",

toString: function() { return this.name; }

};

// помечаем toString как не подлежащий перебору в for..in

Object.defineProperty(user, "toString", {enumerable: false});

for(var key in user) alert(key); // name

Обратим внимание, вызов defineProperty не перезаписал свойство, а просто модифицировал настройки у существующего toString.

**[Свойство-функция](http://learn.javascript.ru/descriptors-getters-setters" \l "svoystvo-funktsiya)**

Дескриптор позволяет задать свойство, которое на самом деле работает как функция. Для этого в нём нужно указать эту функцию в get.

Например, у объекта user есть обычные свойства: имя firstName и фамилия surname.

Создадим свойство fullName, которое на самом деле является функцией:

var user = {

firstName: "Вася",

surname: "Петров"

}

Object.defineProperty(user, "fullName", {

get: function() {

return this.firstName + ' ' + this.surname;

}

});

alert(user.fullName); // Вася Петров

Обратим внимание, снаружи fullName – это обычное свойство user.fullName. Но дескриптор указывает, что на самом деле его значение возвращается функцией.

Также можно указать функцию, которая используется для записи значения, при помощи дескриптора set.

Например, добавим возможность присвоения user.fullName к примеру выше:

var user = {

firstName: "Вася",

surname: "Петров"

}

Object.defineProperty(user, "fullName", {

get: function() {

return this.firstName + ' ' + this.surname;

},

set: function(value) {

var split = value.split(' ');

this.firstName = split[0];

this.surname = split[1];

}

});

user.fullName = "Петя Иванов";

alert( user.firstName ); // Петя

alert( user.surname ); // Иванов

**[Указание get/set в литералах](http://learn.javascript.ru/descriptors-getters-setters" \l "ukazanie-get-set-v-literalah)**

Если мы создаём объект при помощи синтаксиса { ... }, то задать свойства-функции можно прямо в его определении.

Для этого используется особый синтаксис: get свойство или set свойство.

Например, ниже объявлен геттер-сеттер fullName:

var user = {

firstName: "Вася",

surname: "Петров",

get fullName() {

return this.firstName + ' ' + this.surname;

},

set fullName(value) {

var split = value.split(' ');

this.firstName = split[0];

this.surname = split[1];

}

};

alert( user.fullName ); // Вася Петров (из геттера)

user.fullName = "Петя Иванов";

alert( user.firstName ); // Петя (поставил сеттер)

alert( user.surname ); // Иванов (поставил сеттер)

**[Да здравствуют get/set!](http://learn.javascript.ru/descriptors-getters-setters" \l "da-zdravstvuyut-get-set)**

Казалось бы, зачем нам назначать get/set для свойства через всякие хитрые вызовы, когда можно сделать просто функции с самого начала? Например, getFullName, setFullName…

Конечно, в ряде случаев свойства выглядят короче, такое решение просто может быть красивым. Но основной бонус – это гибкость, возможность получить контроль над свойством в любой момент!

Например, в начале разработки мы используем обычные свойства, например у User будет имя name и возраст age:

function User(name, age) {

this.name = name;

this.age = age;

}

var pete = new User("Петя", 25);

alert( pete.age ); // 25

С обычными свойствами в коде меньше букв, они удобны, причины использовать функции пока нет.

…Но рано или поздно могут произойти изменения. Например, в User может стать более целесообразно вместо возраста age хранить дату рождения birthday:

function User(name, birthday) {

this.name = name;

this.birthday = birthday;

}

var pete = new User("Петя", new Date(1987, 6, 1));

Что теперь делать со старым кодом, который выводит свойство age?

Можно, конечно, найти все места и поправить их, но это долго, а иногда и невозможно, скажем, если вы взаимодействуете со сторонней библиотекой, код в которой – чужой и влезать в него нежелательно.

Добавление get-функции age позволяет обойти проблему легко и непринуждённо:

function User(name, birthday) {

this.name = name;

this.birthday = birthday;

// age будет высчитывать возраст по birthday

Object.defineProperty(this, "age", {

get: function() {

var today = new Date();

var yearDelta = today.getFullYear() - this.birthday.getFullYear();

if (today.getMonth() > this.birthday.getMonth() ||

(today.getMonth() === this.birthday.getMonth() && today.getDate() >= this.birthday.getDate())) {

return yearDelta;

}

return yearDelta - 1;

}

});

}

var pete = new User("Петя", new Date(1987, 6, 1));

alert( pete.birthday ); // и дата рождения доступна

alert( pete.age ); // и возраст

Заметим, что pete.age снаружи как было свойством, так и осталось. То есть, переписывать внешний код на вызов функции pete.age() не нужно.

Таким образом, defineProperty позволяет нам начать с обычных свойств, а в будущем, при необходимости, можно в любой момент заменить их на функции, реализующие более сложную логику.

**[Другие методы работы со свойствами](http://learn.javascript.ru/descriptors-getters-setters" \l "drugie-metody-raboty-so-svoystvami)**

[**Object.defineProperties(obj, descriptors)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperties)

Позволяет объявить несколько свойств сразу:

var user = {}

Object.defineProperties(user, {

firstName: {

value: "Петя"

},

surname: {

value: "Иванов"

},

fullName: {

get: function() {

return this.firstName + ' ' + this.surname;

}

}

});

alert( user.fullName ); // Петя Иванов

[**Object.keys(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/keys)**, [Object.getOwnPropertyNames(obj)](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames)**

Возвращают массив – список свойств объекта.

Object.keys возвращает только enumerable-свойства.

Object.getOwnPropertyNames – возвращает все:

var obj = {

a: 1,

b: 2,

internal: 3

};

Object.defineProperty(obj, "internal", {

enumerable: false

});

alert( Object.keys(obj) ); // a,b

alert( Object.getOwnPropertyNames(obj) ); // a, b, internal

[**Object.getOwnPropertyDescriptor(obj, prop)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor)

Возвращает дескриптор для свойства obj[prop].

Полученный дескриптор можно изменить и использовать defineProperty для сохранения изменений, например:

var obj = {

test: 5

};

var descriptor = Object.getOwnPropertyDescriptor(obj, 'test');

// заменим value на геттер, для этого...

delete descriptor.value; // ..нужно убрать value/writable

delete descriptor.writable;

descriptor.get = function() { // и поставить get

alert( "Preved :)" );

};

// поставим новое свойство вместо старого

// если не удалить - defineProperty объединит старый дескриптор с новым

delete obj.test;

Object.defineProperty(obj, 'test', descriptor);

obj.test; // Preved :)

…И несколько методов, которые используются очень редко:

[**Object.preventExtensions(obj)**](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/preventExtensions)

Запрещает добавление свойств в объект.

[**Object.seal(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/seal)

Запрещает добавление и удаление свойств, все текущие свойства делает configurable: false.

[**Object.freeze(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/freeze)

Запрещает добавление, удаление и изменение свойств, все текущие свойства делает configurable: false, writable: false.

[**Object.isExtensible(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/isExtensible)

Возвращает false, если добавление свойств объекта было запрещено вызовом метода Object.preventExtensions.

[**Object.isSealed(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/isSealed)

Возвращает true, если добавление и удаление свойств объекта запрещено, и все текущие свойства являются configurable: false.

[**Object.isFrozen(obj)**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/isFrozen)

Возвращает true, если добавление, удаление и изменение свойств объекта запрещено, и все текущие свойства являются configurable: false, writable: false.