**Методы объектов, this**

До этого мы говорили об объекте лишь как о хранилище значений. Теперь пойдём дальше и поговорим об объектах как о сущностях со своими функциями («методами»).

**[Методы у объектов](http://learn.javascript.ru/object-methods" \l "metody-u-obektov)**

При объявлении объекта можно указать свойство-функцию, например:

var user = {

name: 'Василий',

// метод

sayHi: function() {

alert( 'Привет!' );

}

};

// Вызов

user.sayHi();

Свойства-функции называют «методами» объектов. Их можно добавлять и удалять в любой момент, в том числе и явным присваиванием:

var user = {

name: 'Василий'

};

user.sayHi = function() { // присвоили метод после создания объекта

alert('Привет!');

};

// Вызов метода:

user.sayHi();

**[Доступ к объекту через this](http://learn.javascript.ru/object-methods" \l "dostup-k-obektu-cherez-this)**

Для полноценной работы метод должен иметь доступ к данным объекта. В частности, вызов user.sayHi() может захотеть вывести имя пользователя.

**Для доступа к текущему объекту из метода используется ключевое слово this**.

Значением this является объект перед «точкой», в контексте которого вызван метод, например:

var user = {

name: 'Василий',

sayHi: function() {

alert( this.name );

}

};

user.sayHi(); // sayHi в контексте user

Здесь при выполнении функции user.sayHi() в this будет храниться ссылка на текущий объект user.

Вместо this внутри sayHi можно было бы обратиться к объекту, используя переменную user:

...

sayHi: function() {

alert( user.name );

}

...

…Однако, такое решение нестабильно. Если мы решим скопировать объект в другую переменную, например admin = user, а в переменную user записать что-то другое – обращение будет совсем не по адресу:

var user = {

name: 'Василий',

sayHi: function() {

alert( user.name ); // приведёт к ошибке

}

};

var admin = user;

user = null;

admin.sayHi(); // упс! внутри sayHi обращение по старому имени, ошибка!

Использование this гарантирует, что функция работает именно с тем объектом, в контексте которого вызвана.

Через this метод может не только обратиться к любому свойству объекта, но и передать куда-то ссылку на сам объект целиком:

var user = {

name: 'Василий',

sayHi: function() {

showName(this); // передать текущий объект в showName

}

};

function showName(namedObj) {

alert( namedObj.name );

}

user.sayHi(); // Василий

**[Подробнее про this](http://learn.javascript.ru/object-methods" \l "podrobnee-pro-this)**

Любая функция может иметь в себе this. Совершенно неважно, объявлена ли она в объекте или отдельно от него.

Значение this называется *контекстом вызова* и будет определено в момент вызова функции.

Например, такая функция, объявленная без объекта, вполне допустима:

function sayHi() {

alert( this.firstName );

}

Эта функция ещё не знает, каким будет this. Это выяснится при выполнении программы.

**Если одну и ту же функцию запускать в контексте разных объектов, она будет получать разный this:**

var user = { firstName: "Вася" };

var admin = { firstName: "Админ" };

function func() {

alert( this.firstName );

}

user.f = func;

admin.g = func;

// this равен объекту перед точкой:

user.f(); // Вася

admin.g(); // Админ

admin['g'](); // Админ (не важно, доступ к объекту через точку или квадратные скобки)

Итак, значение this не зависит от того, как функция была создана, оно определяется исключительно в момент вызова.

**[Значение this при вызове без контекста](http://learn.javascript.ru/object-methods" \l "znachenie-this-pri-vyzove-bez-konteksta)**

Если функция использует this – это подразумевает работу с объектом. Но и прямой вызов func() технически возможен.

Как правило, такая ситуация возникает при ошибке в разработке.

При этом this получает значение window, глобального объекта:

function func() {

alert( this ); // выведет [object Window] или [object global]

}

func();

Таково поведение в старом стандарте.

А в режиме use strict вместо глобального объекта this будет undefined:

function func() {

"use strict";

alert( this ); // выведет undefined (кроме IE9-)

}

func();

Обычно если в функции используется this, то она, всё же, служит для вызова в контексте объекта, так что такая ситуация – скорее исключение.

**[Ссылочный тип](http://learn.javascript.ru/object-methods" \l "ssylochnyy-tip)**

Контекст this никак не привязан к функции, даже если она создана в объявлении объекта. Чтобы this передался, нужно вызвать функцию именно через точку (или квадратные скобки).

Любой более хитрый вызов приведёт к потере контекста, например:

var user = {

name: "Вася",

hi: function() { alert(this.name); },

bye: function() { alert("Пока"); }

};

user.hi(); // Вася (простой вызов работает)

// а теперь вызовем user.hi или user.bye в зависимости от имени

(user.name == "Вася" ? user.hi : user.bye)(); // undefined

В последней строке примера метод получен в результате выполнения тернарного оператора и тут же вызван. Но this при этом теряется.

Если хочется понять, почему, то причина кроется в деталях работы вызова obj.method().

Он ведь, на самом деле, состоит из двух независимых операций: точка . – получение свойства и скобки () – его вызов (предполагается, что это функция).

Функция, как мы говорили раньше, сама по себе не запоминает контекст. Чтобы «донести его» до скобок, JavaScript применяет «финт ушами» – точка возвращает не функцию, а значение специального «ссылочного» типа [Reference Type](https://tc39.github.io/ecma262/" \l "sec-reference-specification-type).

Этот тип представляет собой связку «base-name-strict», где:

* *base* – как раз объект,
* *name* – имя свойства,
* *strict* – вспомогательный флаг для передачи use strict.

То есть, ссылочный тип (Reference Type) – это своеобразное «три-в-одном». Он существует исключительно для целей спецификации, мы его не видим, поскольку любой оператор тут же от него избавляется:

* Скобки () получают из base значение свойства name и вызывают в контексте base.
* Другие операторы получают из base значение свойства name и используют, а остальные компоненты игнорируют.

Поэтому любая операция над результатом операции получения свойства, кроме вызова, приводит к потере контекста.

Аналогично работает и получение свойства через квадратные скобки obj[method].