# Методы объекта, "this"

Объекты обычно создаются, чтобы представлять сущности реального мира, будь то пользователи, заказы и так далее:

// Объект пользователя

let user = {

name: "Джон",

age: 30

};

И так же, как и в реальном мире, пользователь может *совершать действия*: выбирать что-то из корзины покупок, авторизовываться, выходить из системы, оплачивать и т.п.

Такие действия в JavaScript представлены свойствами-функциями объекта.

# Примеры методов

Для начала давайте научим нашего пользователя user здороваться:

let user = {

name: "Джон",

age: 30

};

user.sayHi = function() {

alert("Привет!");

};

user.sayHi(); // Привет!

Здесь мы просто использовали Function Expression (функциональное выражение), чтобы создать функцию для приветствия, и присвоили её свойству user.sayHi нашего объекта.

Затем мы вызвали её. Теперь пользователь может говорить!

Функцию, которая является свойством объекта, называют *методом* этого объекта.

Итак, мы получили метод sayHi объекта user.

Конечно, мы могли бы заранее объявить функцию и использовать её в качестве метода, примерно так:

let user = {

// ...

};

// сначала объявляем

function sayHi() {

alert("Привет!");

};

// затем добавляем в качестве метода

user.sayHi = sayHi;

user.sayHi(); // Привет!

Объектно-ориентированное программирование

Когда мы пишем наш код, используя объекты для представления сущностей реального мира, – это называется объектно-ориентированное программирование или сокращённо: «ООП».

ООП является большой предметной областью и интересной наукой само по себе. Как выбрать правильные сущности? Как организовать взаимодействие между ними? Это – создание архитектуры, и есть хорошие книги по этой теме, такие как «Приёмы объектно-ориентированного проектирования. Паттерны проектирования» авторов Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес или «Объектно-ориентированный анализ и проектирование с примерами приложений» Гради Буча, а также ещё множество других книг.

# Сокращённая запись метода

Существует более короткий синтаксис для методов в литерале объекта:

// эти объекты делают одно и то же (одинаковые методы)

user = {

sayHi: function() {

alert("Привет");

}

};

// сокращённая запись выглядит лучше, не так ли?

user = {

sayHi() { // то же самое, что и "sayHi: function()"

alert("Привет");

}

};

Как было показано, мы можем пропустить ключевое слово "function" и просто написать sayHi().

Нужно отметить, что эти две записи не полностью эквивалентны. Есть тонкие различия, связанные с наследованием объектов (что будет рассмотрено позже), но на данном этапе изучения это неважно. В большинстве случаев сокращённый синтаксис предпочтителен.

# Ключевое слово «this» в методах

Как правило, методу объекта необходим доступ к информации, которая хранится в объекте, чтобы выполнить с ней какие-либо действия (в соответствии с назначением метода).

Например, коду внутри user.sayHi() может понадобиться имя пользователя, которое хранится в объекте user.

**Для доступа к информации внутри объекта метод может использовать ключевое слово this.**

Значение this – это объект «перед точкой», который использовался для вызова метода.

Например:

let user = {

name: "Джон",

age: 30,

sayHi() {

// this - это "текущий объект"

alert(this.name);

}

};

user.sayHi(); // Джон

Здесь во время выполнения кода user.sayHi() значением this будет являться user (ссылка на объект user).

Технически также возможно получить доступ к объекту без ключевого слова this, ссылаясь на него через внешнюю переменную (в которой хранится ссылка на этот объект):

let user = {

name: "Джон",

age: 30,

sayHi() {

alert(user.name); // используем переменную "user" вместо ключевого слова "this"

}

};

…Но такой код будет ненадёжным. Если мы решим скопировать ссылку на объект user в другую переменную, например, admin = user, и перезапишем переменную user чем-то другим, тогда будет осуществлён доступ к неправильному объекту при вызове метода из admin.

Это показано ниже:

let user = {

name: "Джон",

age: 30,

sayHi() {

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

}

};

let admin = user;

user = null; // обнулим переменную для наглядности, теперь она не хранит ссылку на объект.

admin.sayHi(); // Ошибка! Внутри sayHi() используется user, которая больше не ссылается на объект!

Если мы используем this.name вместо user.name внутри alert, тогда этот код будет работать.

# «this» не является фиксированным

В JavaScript ключевое слово «this» ведёт себя иначе, чем в большинстве других языков программирования. Оно может использоваться в любой функции.

В этом коде нет синтаксической ошибки:

function sayHi() {

alert( this.name );

}

Значение this вычисляется во время выполнения кода и зависит от контекста.

Например, здесь одна и та же функция назначена двум разным объектам и имеет различное значение «this» при вызовах:

let user = { name: "Джон" };

let admin = { name: "Админ" };

function sayHi() {

alert( this.name );

}

// используем одну и ту же функцию в двух объектах

user.f = sayHi;

admin.f = sayHi;

// вызовы функции, приведённые ниже, имеют разное значение this

// "this" внутри функции является ссылкой на объект, который указан "перед точкой"

user.f(); // Джон (this == user)

admin.f(); // Админ (this == admin)

admin['f'](); // Админ (неважен способ доступа к методу - через точку или квадратные скобки)

Правило простое: при вызове obj.f() значение this внутри f равно obj. Так что, в приведённом примере это user или admin.

Вызов без объекта: this == undefined

Мы даже можем вызвать функцию вовсе без использования объекта:

function sayHi() {

alert(this);

}

sayHi(); // undefined

В строгом режиме ("use strict") в таком коде значением this будет являться undefined. Если мы попытаемся получить доступ к name, используя this.name – это вызовет ошибку.

В нестрогом режиме значением this в таком случае будет *глобальный объект* (window для браузера, мы вернёмся к этому позже в главе Глобальный объект). Это – исторически сложившееся поведение this, которое исправляется использованием строгого режима ("use strict").

Обычно подобный вызов является ошибкой программирования. Если внутри функции используется this, тогда ожидается, что она будет вызываться в контексте какого-либо объекта.

Последствия свободного this

Если вы до этого изучали другие языки программирования, тогда вы, скорее всего, привыкли к идее "фиксированного this" – когда методы, определённые внутри объекта, всегда сохраняют в качестве значения this ссылку на свой объект (в котором был определён метод).

В JavaScript this является «свободным», его значение вычисляется в момент вызова метода и не зависит от того, где этот метод был объявлен, а зависит от того, какой объект вызывает метод (какой объект стоит «перед точкой»).

Эта идея вычисления this в момент исполнения имеет как свои плюсы, так и минусы. С одной стороны, функция может быть повторно использована в качестве метода у различных объектов (что повышает гибкость). С другой стороны, большая гибкость увеличивает вероятность ошибок.

Здесь мы не будем судить о том, является ли это решение в языке хорошим или плохим. Мы должны понимать, как с этим работать, чтобы получать выгоды и избегать проблем.

# У стрелочных функций нет «this»

Стрелочные функции особенные: у них нет своего «собственного» this. Если мы используем this внутри стрелочной функции, то его значение берётся из внешней «нормальной» функции.

Например, здесь arrow() использует значение this из внешнего метода user.sayHi():

let user = {

firstName: "Илья",

sayHi() {

let arrow = () => alert(this.firstName);

arrow();

}

};

user.sayHi(); // Илья

Это является особенностью стрелочных функций. Они полезны, когда мы на самом деле не хотим иметь отдельное значение this, а хотим брать его из внешнего контекста. Позднее в главе [Повторяем стрелочные функции](https://learn.javascript.ru/arrow-functions) мы увидим больше примеров на эту тему.

**[Итого](https://learn.javascript.ru/object-methods" \l "itogo)**

* Функции, которые находятся в объекте в качестве его свойств, называются «методами».
* Методы позволяют объектам «действовать»: object.doSomething().
* Методы могут ссылаться на объект через this.

Значение this определяется во время исполнения кода.

* При объявлении любой функции в ней можно использовать this, но этот this не имеет значения до тех пор, пока функция не будет вызвана.
* Эта функция может быть скопирована между объектами (из одного объекта в другой).
* Когда функция вызывается синтаксисом «метода» – object.method(), значением this во время вызова является объект перед точкой.

Также ещё раз заметим, что стрелочные функции являются особенными – у них нет this. Когда внутри стрелочной функции обращаются к this, то его значение берётся снаружи.