**Явное указание this: "call", "apply"**

Итак, мы знаем, что this – это текущий объект при вызове «через точку» и новый объект при конструировании через new.

В этой главе наша цель получить окончательное и полное понимание this в JavaScript. Для этого не хватает всего одного элемента: способа явно указать this при помощи методов call и apply.

**[Метод call](http://learn.javascript.ru/call-apply" \l "metod-call)**

Синтаксис метода call:

func.call(context, arg1, arg2, ...)

При этом вызывается функция func, первый аргумент call становится её this, а остальные передаются «как есть».

**Вызов func.call(context, a, b...) – то же, что обычный вызов func(a, b...), но с явно указанным this(=context).**

Например, у нас есть функция showFullName, которая работает с this:

function showFullName() {

alert( this.firstName + " " + this.lastName );

}

Пока объекта нет, но это нормально, ведь JavaScript позволяет использовать this везде. Любая функция может в своём коде упомянуть this, каким будет это значение – выяснится в момент запуска.

Вызов showFullName.call(user) запустит функцию, установив this = user, вот так:

function showFullName() {

alert( this.firstName + " " + this.lastName );

}

var user = {

firstName: "Василий",

lastName: "Петров"

};

// функция вызовется с this=user

showFullName.call(user) // "Василий Петров"

После контекста в call можно передать аргументы для функции. Вот пример с более сложным вариантом showFullName, который конструирует ответ из указанных свойств объекта:

var user = {

firstName: "Василий",

surname: "Петров",

patronym: "Иванович"

};

function showFullName(firstPart, lastPart) {

alert( this[firstPart] + " " + this[lastPart] );

}

// f.call(контекст, аргумент1, аргумент2, ...)

showFullName.call(user, 'firstName', 'surname') // "Василий Петров"

showFullName.call(user, 'firstName', 'patronym') // "Василий Иванович"

**[«Одалживание метода»](http://learn.javascript.ru/call-apply" \l "odalzhivanie-metoda)**

При помощи call можно легко взять метод одного объекта, в том числе встроенного, и вызвать в контексте другого.

Это называется «одалживание метода» (на англ. *method borrowing*).

**Используем эту технику для упрощения манипуляций с arguments.**

Как мы знаем, arguments не массив, а обычный объект, поэтому таких полезных методов как push, pop, join и других у него нет. Но иногда так хочется, чтобы были…

Нет ничего проще! Давайте скопируем метод join из обычного массива:

function printArgs() {

arguments.join = [].join; // одолжили метод (1)

var argStr = arguments.join(':'); // (2)

alert( argStr ); // сработает и выведет 1:2:3

}

printArgs(1, 2, 3);

1. В строке (1) объявлен пустой массив [] и скопирован его метод [].join. Обратим внимание, мы не вызываем его, а просто копируем. Функция, в том числе встроенная – обычное значение, мы можем скопировать любое свойство любого объекта, и [].join здесь не исключение.
2. В строке (2) запустили join в контексте arguments, как будто он всегда там был.

**Почему вызов сработает?**

Здесь метод [join](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/join) массива скопирован и вызван в контексте arguments. Не произойдёт ли что-то плохое от того, что arguments – не массив? Почему он, вообще, сработал?

Ответ на эти вопросы простой. В соответствии [со спецификацией](http://es5.github.com/x15.4.html#x15.4.4.5), внутри join реализован примерно так:

function join(separator) {

if (!this.length) return '';

var str = this[0];

for (var i = 1; i < this.length; i++) {

str += separator + this[i];

}

return str;

}

Как видно, используется this, числовые индексы и свойство length. Если эти свойства есть, то все в порядке. А больше ничего и не нужно.

В качестве this подойдёт даже обычный объект:

var obj = { // обычный объект с числовыми индексами и length

0: "А",

1: "Б",

2: "В",

length: 3

};

obj.join = [].join;

alert( obj.join(';') ); // "A;Б;В"

…Однако, копирование метода из одного объекта в другой не всегда приемлемо!

Представим на минуту, что вместо arguments у нас – произвольный объект. У него тоже есть числовые индексы, length и мы хотим вызвать в его контексте метод [].join. То есть, ситуация похожа на arguments, но (!) вполне возможно, что у объекта есть *свой* метод join.

Поэтому копировать [].join, как сделано выше, нельзя: если он перезапишет собственный join объекта, то будет страшный бардак и путаница.

Безопасно вызвать метод нам поможет call:

function printArgs() {

var join = [].join; // скопируем ссылку на функцию в переменную

// вызовем join с this=arguments,

// этот вызов эквивалентен arguments.join(':') из примера выше

var argStr = join.call(arguments, ':');

alert( argStr ); // сработает и выведет 1:2:3

}

printArgs(1, 2, 3);

Мы вызвали метод без копирования. Чисто, безопасно.

**[Ещё пример: [].slice.call(arguments)](http://learn.javascript.ru/call-apply" \l "eschyo-primer-slice-call-arguments)**

В JavaScript есть очень простой способ сделать из arguments настоящий массив. Для этого возьмём метод массива: [slice](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/slice).

По стандарту вызов arr.slice(start, end) создаёт новый массив и копирует в него элементы массива arr от start до end. А если start и end не указаны, то копирует весь массив.

Вызовем его в контексте arguments:

function printArgs() {

// вызов arr.slice() скопирует все элементы из this в новый массив

var args = [].slice.call(arguments);

alert( args.join(', ') ); // args - полноценный массив из аргументов

}

printArgs('Привет', 'мой', 'мир'); // Привет, мой, мир

Как и в случае с join, такой вызов технически возможен потому, что slice для работы требует только нумерованные свойства и length. Всё это в arguments есть.

**[Метод apply](http://learn.javascript.ru/call-apply" \l "metod-apply)**

Если нам неизвестно, с каким количеством аргументов понадобится вызвать функцию, можно использовать более мощный метод: apply.

**Вызов функции при помощи func.apply работает аналогично func.call, но принимает массив аргументов вместо списка.**

func.call(context, arg1, arg2);

// идентичен вызову

func.apply(context, [arg1, arg2]);

В частности, эти две строчки сработают одинаково:

showFullName.call(user, 'firstName', 'surname');

showFullName.apply(user, ['firstName', 'surname']);

Преимущество apply перед call отчётливо видно, когда мы формируем массив аргументов динамически.

Например, в JavaScript есть встроенная функция Math.max(a, b, c...), которая возвращает максимальное значение из аргументов:

alert( Math.max(1, 5, 2) ); // 5

При помощи apply мы могли бы найти максимум в произвольном массиве, вот так:

var arr = [];

arr.push(1);

arr.push(5);

arr.push(2);

// получить максимум из элементов arr

alert( Math.max.apply(null, arr) ); // 5

В примере выше мы передали аргументы через массив – второй параметр apply… Но вы, наверное, заметили небольшую странность? В качестве контекста this был передан null.

Строго говоря, полным эквивалентом вызову Math.max(1,2,3) был бы вызов Math.max.apply(Math, [1,2,3]). В обоих этих вызовах контекстом будет объект Math.

Но в данном случае в качестве контекста можно передавать что угодно, поскольку в своей внутренней реализации метод Math.max не использует this. Действительно, зачем this, если нужно всего лишь выбрать максимальный из аргументов? Вот так, при помощи apply мы получили короткий и элегантный способ вычислить максимальное значение в массиве!

**Вызов call/apply с null или undefined**

В современном стандарте call/apply передают this «как есть». А в старом, без use strict, при указании первого аргумента null или undefined в call/apply, функция получает this = window, например:

Современный стандарт:

function f() {

"use strict";

alert( this ); // null

}

f.call(null);

Без use strict:

function f() {

alert( this ); // window

}

f.call(null);

**[Итого про this](http://learn.javascript.ru/call-apply" \l "itogo-pro-this)**

Значение this устанавливается в зависимости от того, как вызвана функция:

* При вызове функции как метода:

obj.func(...) // this = obj

obj["func"](...)

* При обычном вызове:

func(...) // this = window (ES3) /undefined (ES5)

* В new:

new func() // this = {} (новый объект)

* Явное указание:

func.apply(context, args) // this = context (явная передача)

func.call(context, arg1, arg2, ...)