

Call, Apply, Bind

Занятие 9

План занятия

- Явное указание this: «call», «apply»
- Привязка контекста и карринг: «bind»
- Функции-обёртки, декораторы



Явное указание this

Метод call

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

```
function showFullName() {  
    console.log( this.firstName + ' ' + this.lastName );  
}
```

Метод call

```
function showFullName() {  
  console.log( this.firstName + ' ' + this.lastName );  
}
```

```
var user = {  
  firstName: 'Василий',  
  lastName: 'Петров'  
};
```

```
// функция вызовется с this=user  
showFullName.call(user) // 'Василий Петров'
```

Метод call

```
var user = {  
  firstName: 'Василий',  
  surname: 'Петров',  
  patronym: 'Иванович'  
};
```

```
function showFullName(firstPart, lastPart) {  
  console.log( this[firstPart] + ' ' + this[lastPart] );  
}
```

```
// f.call(контекст, аргумент1, аргумент2, ...)  
showFullName.call(user, 'firstName', 'surname');  
// 'Василий Петров'
```

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

Method borrowing

```
function printArgs() {  
    arguments.join = [].join; // одолжили метод (1)  
  
    var argStr = arguments.join(':'); // (2)  
  
    console.log( argStr ); // сработает и выведет 1:2:3  
}  
  
printArgs(1, 2, 3);
```

Method borrowing

```
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, ':');  
  
    console.log( argStr ); // сработает и выведет 1:2:3  
}  
  
printArgs(1, 2, 3);
```


Method borrowing

```
function printArgs() {  
  // вызов arr.slice() скопирует  
  // все элементы из this в новый массив  
  var args = [].slice.call(arguments);  
  console.log( args.join(', ') );  
  // args - полноценный массив из аргументов  
}  
  
printArgs('Привет', 'мой', 'мир'); // Привет, мой, мир
```

Метод apply

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

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

Метод apply

```
console.log( Math.max(1, 5, 2) ); // 5
```

```
var arr = [];  
arr.push(1);  
arr.push(5);  
arr.push(2);
```

// получить максимум из элементов arr

```
console.log( Math.max.apply(null, arr) ); // 5
```

Метод apply

```
console.log( Math.max(1, 5, 2) ); // 5
```

```
var arr = [];  
arr.push(1);  
arr.push(5);  
arr.push(2);
```

// получить максимум из элементов arr

```
console.log( Math.max.apply(null, arr) ); // 5
```


Итого про this

// 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, ...)`



Привязка контекста и карринг: «bind»

Пример потери контекста

```
setTimeout(function() {  
    alert( 'Привет' );  
}, 1000);
```

```
var user = {  
    firstName: 'Вася',  
    sayHi: function() {  
        alert( this.firstName );  
    }  
};
```

```
setTimeout(user.sayHi, 1000);  
// undefined (не Вася)
```


Пример потери контекста

```
setTimeout(user.sayHi, 1000);  
// undefined (не Вася!)
```

```
var f = user.sayHi;  
setTimeout(f, 1000); // контекст user потеряли
```


Решение 1: сделать обёртку

```
var user = {  
  firstName: "Вася",  
  sayHi: function() {  
    alert( this.firstName );  
  }  
};
```

```
setTimeout(function() {  
  user.sayHi(); // Вася  
, 1000);
```

Решение 2: bind

```
function bind(func, context) {  
  return function() { // (*)  
    return func.apply(context, arguments);  
  };  
}
```

```
function f() {  
  alert( this );  
}
```

```
var g = bind(f, 'Context');  
g(); // Context
```

Решение 2: bind

```
function f(a, b) {  
  alert( this );  
  alert( a + b );  
}
```

```
var g = bind(f, 'Context');  
g(1, 2); // Context, затем 3
```

Решение 2: bind

```
function bind(func, context) {  
  return function() {  
    return func.apply(context, arguments);  
  };  
}
```

```
var user = {  
  firstName: 'Вася',  
  sayHi: function() {  
    alert( this.firstName );  
  }  
};
```

```
setTimeout(bind(user.sayHi, user), 1000);
```


Решение 2: bind

```
var user = {  
  firstName: 'Вася',  
  sayHi: function(who) { // здесь у sayHi есть один аргумент  
    alert( this.firstName + ': Привет, ' + who );  
  }  
};
```

```
var sayHi = bind(user.sayHi, user);
```

// контекст Вася, а аргумент передаётся 'как есть'

```
sayHi('Петя'); // Вася: Привет, Петя
```

```
sayHi('Маша'); // Вася: Привет, Маша
```

Решение 3: встроенный метод bind

```
function f(a, b) {  
  alert( this );  
  alert( a + b );  
}
```

// вместо

// var g = bind(f, 'Context');

```
var g = f.bind('Context');
```

```
g(1, 2); // Context, затем 3
```

Решение 3: встроенный метод bind

```
var user = {  
  firstName: 'Вася',  
  sayHi: function() {  
    alert( this.firstName );  
  }  
};
```

```
// setTimeout( bind(user.sayHi, user), 1000 );  
setTimeout(user.sayHi.bind(user), 1000);  
// аналог через встроенный метод
```

Currying (каррирование)

```
function mul(a, b) {  
  return a * b;  
}
```

// double умножает только на два

// контекст фиксируем null, он не используется

```
var double = mul.bind(null, 2);
```

```
alert( double(3) ); // = mul(2, 3) = 6
```

```
alert( double(4) ); // = mul(2, 4) = 8
```

```
alert( double(5) ); // = mul(2, 5) = 10
```


Currying (каррирование)

```
var triple = mul.bind(null, 3);
```

// контекст фиксируем null, он не используется

```
alert( triple(3) ); // = mul(3, 3) = 9
```

```
alert( triple(4) ); // = mul(3, 4) = 12
```

```
alert( triple(5) ); // = mul(3, 5) = 15
```

Задача 9.1

устно

Что выведет этот код?

```
function f() {  
    alert( this );  
}  
  
var user = {  
    g: f.bind('Hello')  
};  
  
user.g();
```

Задача 9.2

устно

Что выведет этот код?

```
function f() {  
  alert(this.name);  
}
```

```
f = f.bind( {name: 'Вася'} ).bind( {name: 'Петя'} );
```

```
f();
```

Задача 9.3

устно

В свойство функции записано значение. Изменится ли оно после применения bind? Почему?

```
function sayHi() {  
  alert( this.name );  
}  
sayHi.test = 5;  
alert( sayHi.test ); // 5
```

```
var bound = sayHi.bind({  
  name: 'Вася'  
});
```

```
alert( bound.test ); // что выведет? почему?
```




Функции-обёртки, декораторы

bind — привязка контекста

```
function bind(func, context) {  
  return function() {  
    return func.apply(context, arguments);  
  };  
}
```

Декоратор-таймер

```
function f(x) {} // любая функция
```

```
var timers = {}; // объект для таймеров
```

```
// отдекорировали
```

```
f = timingDecorator(f, 'myFunc');
```

```
// запускаем
```

```
f(1);
```

```
f(2);
```

```
f(3);
```

```
// функция работает как раньше, но время подсчитывается
```

```
alert( timers.myFunc );
```

```
// общее время выполнения всех вызовов f
```


Декоратор-таймер

```
var timers = {};
```

```
// прибавит время выполнения f к таймеру timers[timer]
```

```
function timingDecorator(f, timer) {
```

```
  return function() {
```

```
    var start = performance.now();
```

```
    var result = f.apply(this, arguments); // (*)
```

```
    if (!timers[timer]) timers[timer] = 0;
```

```
    timers[timer] += performance.now() - start;
```

```
    return result;
```

```
  }
```

```
}
```

Декоратор-таймер

// функция может быть произвольной, например такой:

```
function fibonacci(n) {  
    return (n > 2) ? fibonacci(n - 1) + fibonacci(n - 2) : 1;  
}
```

// использование: завернём fibonacci в декоратор
fibonacci = timingDecorator(fibonacci, 'fibo');

// неоднократные вызовы...

```
alert( fibonacci(10) ); // 55  
alert( fibonacci(20) ); // 6765  
// ...
```

// в любой момент можно получить общее количество времени на вызовы

```
alert( timers.fibo + 'мс' );
```

Декоратор проверки доступа

```
function checkPermissionDecorator(f) {  
  return function() {  
    if (isAdmin()) {  
      return f.apply(this, arguments);  
    }  
    alert( 'Недостаточно прав' );  
  }  
}
```

//Использование декоратора:

```
function save() { ... }
```

```
save = checkPermissionDecorator(save);
```

// Теперь вызов функции save() проверяет права

Задача 9.4

Логирующий декоратор (1 аргумент)

Создайте декоратор **makeLogging**(f, log),
который берет функцию f и массив log.

Он должен возвращать обёртку вокруг f, которая при каждом вызове записывает аргументы в log, а затем передает вызов в f.

```
function work(a) {  
    /* ... */ // work - произвольная функция, один аргумент  
}  
function makeLogging(f, log) { /* ваш код */  
  
    var log = [];  
    work = makeLogging(work, log);  
  
    work(1); // 1, добавлено в log  
    work(5); // 5, добавлено в log  
  
    console.log(log); // [1,5]
```

Задача 9.4

Решение

```
function makeLogging(f, log) {  
  
    function wrapper(a) {  
        log.push(a);  
        return f.call(this, a);  
    }  
  
    return wrapper;  
}
```

План занятия

- Явное указание this: «call», «apply»
- Привязка контекста и карринг: «bind»
- Функции-обёртки, декораторы



2015