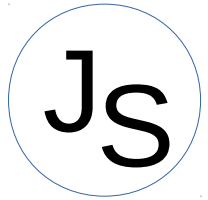
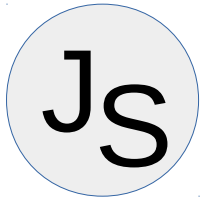
Стрелочные функции



- Роли функций
- Замыкание
- Проблема this
- Лексические переменные
- Заимствование

3 роли функции и this



Метод

```
obj.func(...)     // this = obj
obj["func"](...)
```

Конструктор

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

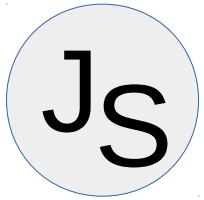
Обычный вызов

```
func(...) // this = window (ES3)
          // this = undefined (ES5 strict mode)
```

Явное указание

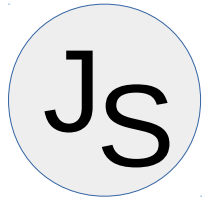
```
func.apply(context, args) // this = context
func.call (context, arg1, arg2, ...)
```

Роли функций и this



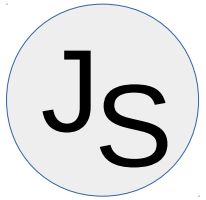

```
function Prefixer(prefix) {
    this.prefix = prefix; //this - сама функция
}

Prefixer.prototype.prefixArray =
   function (arr) { // this - Prefixer
      return arr.map(
         // this не передаётся (обычная функция)
         function (x) {return this.prefix + x;}
      );
   }
;

var pre = new Prefixer('Hi ');
pre.prefixArray(['Joe', 'Alex'])
// ["undefinedJoe", "undefinedAlex"]
```

Решение 1: that



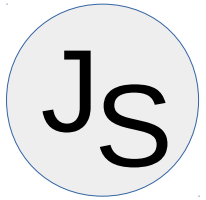

```
Prefixer.prototype.prefixArray =
   function (arr) {
      var that = this;
      return arr.map(
         function (x) {
            return that.prefix + x;
         }
      );
   }
;

var pre = new Prefixer('Hi ');
pre.prefixArray(['Joe', 'Alex'])
// [ 'Hi Joe', 'Hi Alex' ]
```

Решение 2: задать this




```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         function (x) {
            return this.prefix + x;
         },
         this
      );
   }
;
```

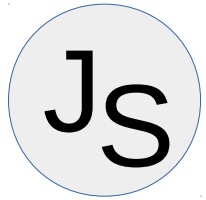
Решение 3: привязать this



bind преобразует функцию в функцию с постоянным this

```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         function (x) {
            return this.prefix + x;
         }.bind(this)
      );
   }
;
```

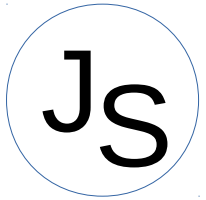
ES6: стрелочные функции



Не перекрывает this, работает как bind

```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         (x) => {
            return this.prefix + x;
         }
      );
   }
;
```

Стрелочные функции



Краткость

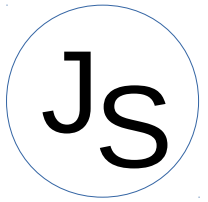
```
const squares = arr.map(x => x * x);
```

```
const squares = arr.map(function (x) { return x * x });
```

```
    () => { ... } // без параметров
     x => { ... } // 1 параметр-идентификатор
(x, y) => { ... } // несколько либо 1 не идентификатор
```

```
x => { return x * x }  // блок
x => x * x             // выражение
```

Стрелочные функции



Распространение (propagation) переменных

статическое (лексическое)

```
const x = 123;

function foo(y) {
    return x;
}
```

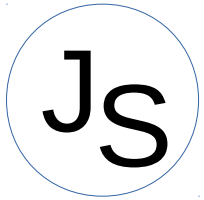
динамическое

```
function bar(arg)
{
    return arg;
}
```

Лексические переменные

- arguments
- super
- this
- new.target

Стрелочные функции



Нельзя использовать как конструктор

```
new (() => {})      // error
```

Callback

```
$button.on('click', event => {
    event.target.classList.toggle('clicked');
});
```

Объявление функции

```
function foo(arg1, arg2) {
    ...
}
```

Метод

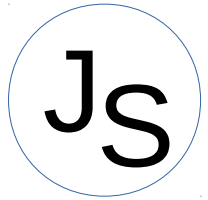
```
MyClass.prototype.foo = function (arg1, arg2) {
    ...
};
```

Замыкание

```
var cnt = 0;
function counter() {cnt += 1;}

counter(); // 1
counter(); // 2
counter(); // 3
```

```
var add = (
   ()=>{
      let cnt = 0;
      return ()=>{cnt++; return cnt;}
   }
)();

add();    // 1
add();    // 2
```

Заимствование




```
function Parent(name) {
    this.name = name || "Adam";
}

// Переиспользуемые свойства и методы
Parent.prototype.say = function () {return this.name;}

// Объект-наследник
function Child(name) {}
```

```
var child = new Child();
child.name = 'Eva';
child.say = Parent.prototype.say.apply(child, [/*par1, par2*/]);

// Заимствуем в объект child метод say из объекта parent
console.log(child.say); // Eva

// Меняем контекст вызова метода
var say = child.say;
console.log(say()); // TypeError: say is not a function
```

Заимствование

bind (ES5)

```
var child = new Child();
child.name = 'Eva';

// Заимствуем в объект child метод say из объекта parent
child.say = Parent.prototype.say.bind(child);
console.log(child.say); // function()

// Меняем контекст вызова метода
var say = child.say;
console.log(say()); // Eva
```

call, apply

```
Array.prototype.reverse.call(
    {
        0:"Martin", 1:78, 2:67, 3:["Letta", "Marieta", "Pauline"],
        length:4
    }
)
// Object {0: Array(3), 1: 67, 2: 78, 3: "Martin", length: 4}
```

Стрелочные функции

- Роли функций
- Замыкание
- Проблема this
- Лексические переменные
- Заимствование

3 роли функции и this

Метод

```
obj.func(...)    // this = obj
obj["func"](...)
```

Конструктор

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

Обычный вызов

```
func(...) // this = window (ES3)
          // this = undefined (ES5 strict mode)
```

Явное указание

```
func.apply(context, args) // this = context
func.call (context, arg1, arg2, ...)
```

Роли функций и this

```
function Prefixer(prefix) {
    this.prefix = prefix; //this - сама функция
}

Prefixer.prototype.prefixArray =
   function (arr) { // this - Prefixer
      return arr.map(
         // this не передаётся (обычная функция)
         function (x) {return this.prefix + x;}
      );
   }
;

var pre = new Prefixer('Hi ');
pre.prefixArray(['Joe', 'Alex'])
// ["undefinedJoe", "undefinedAlex"]
```

Решение 1: that

```
Prefixer.prototype.prefixArray =
   function (arr) {
      var that = this;
      return arr.map(
         function (x) {
            return that.prefix + x;
         }
      );
   }
;

var pre = new Prefixer('Hi ');
pre.prefixArray(['Joe', 'Alex'])
// [ 'Hi Joe', 'Hi Alex' ]
```

Решение 2: задать this

```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         function (x) {
            return this.prefix + x;
         },
         this
      );
   }
;
```

Решение 3: привязать this

bind преобразует функцию в функцию с постоянным this

```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         function (x) {
            return this.prefix + x;
         }.bind(this)
      );
   }
;
```

ES6: стрелочные функции

Не перекрывает this, работает как bind

```
Prefixer.prototype.prefixArray =
   function (arr) {
      return arr.map(
         (x) => {
            return this.prefix + x;
         }
      );
   }
;
```

Стрелочные функции

Краткость

```
const squares = arr.map(x => x * x);
```

```
const squares = arr.map(function (x) { return x * x });
```

```
    () => { ... } // без параметров
     x => { ... } // 1 параметр-идентификатор
(x, y) => { ... } // несколько либо 1 не идентификатор
```

```
x => { return x * x }  // блок
x => x * x             // выражение
```

Стрелочные функции

Распространение (propagation) переменных

статическое (лексическое)

```
const x = 123;

function foo(y) {
    return x;
}
```

динамическое

```
function bar(arg)
{
    return arg;
}
```

Лексические переменные

- arguments
- super
- this
- new.target

Стрелочные функции

Нельзя использовать как конструктор

```
new (() => {})      // error
```

Callback

```
$button.on('click', event => {
    event.target.classList.toggle('clicked');
});
```

Объявление функции

```
function foo(arg1, arg2) {
    ...
}
```

Метод

```
MyClass.prototype.foo = function (arg1, arg2) {
    ...
};
```

Замыкание

```
var cnt = 0;
function counter() {cnt += 1;}

counter(); // 1
counter(); // 2
counter(); // 3
```

```
var add = (
   ()=>{
      let cnt = 0;
      return ()=>{cnt++; return cnt;}
   }
)();

add();   // 1
add();   // 2
```

Заимствование

```
function Parent(name) {
    this.name = name || "Adam";
}

// Переиспользуемые свойства и методы
Parent.prototype.say = function () {return this.name;}

// Объект-наследник
function Child(name) {}
```

```
var child = new Child();
child.name = 'Eva';
child.say = Parent.prototype.say.apply(child, [/*par1, par2*/]);

// Заимствуем в объект child метод say из объекта parent
console.log(child.say); // Eva

// Меняем контекст вызова метода
var say = child.say;
console.log(say()); // TypeError: say is not a function
```

Заимствование

bind (ES5)

```
var child = new Child();
child.name = 'Eva';

// Заимствуем в объект child метод say из объекта parent
child.say = Parent.prototype.say.bind(child);
console.log(child.say); // function()

// Меняем контекст вызова метода
var say = child.say;
console.log(say()); // Eva
```

call, apply

```
Array.prototype.reverse.call(
    {
        0:"Martin", 1:78, 2:67, 3:["Letta", "Marieta", "Pauline"],
        length:4
    }
)
// Object {0: Array(3), 1: 67, 2: 78, 3: "Martin", length: 4}
```