

this

# this C#

```
class User
{
    private int age = 24;

    public void ShowAge()
    {
        Console.WriteLine(this.age);
    }
}

static void Main()
{
    User mike = new User();

    mike.ShowAge(); // 24
}
```

# this Java

```
public class User {  
    private int age = 24;  
  
    public void showAge() {  
        System.out.println(this.age);  
    }  
}  
  
public static void main(String []args){  
    User mike = new User();  
  
    mike.showAge(); // 24  
}
```

## Свойства this

- ключевое слово
- нельзя перезаписать
- указывает на текущий объект

# this JavaScript

```
function User () {  
  return {  
    age: 24,  
  
    showAge: function () {  
      console.log(this.age);  
    }  
  }  
}  
  
var mike = new User();  
  
mike.showAge(); // 24
```

## Свойства this в JavaScript

- ключевое слово
- нельзя перезаписать
- указывает на текущий объект
- МОЖНО использовать за пределом объекта\*

this за пределом объекта

```
this.innerWidth; // 1280
```

```
this.process.version; // v7.0.0
```

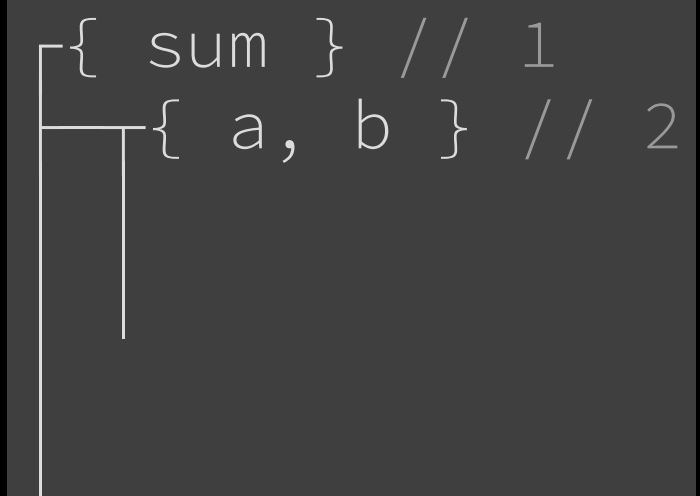
# Контекст исполнения



*Код:*

```
function sum(a, b) {  
    return a + b;  
}  
  
sum(1, 2);
```

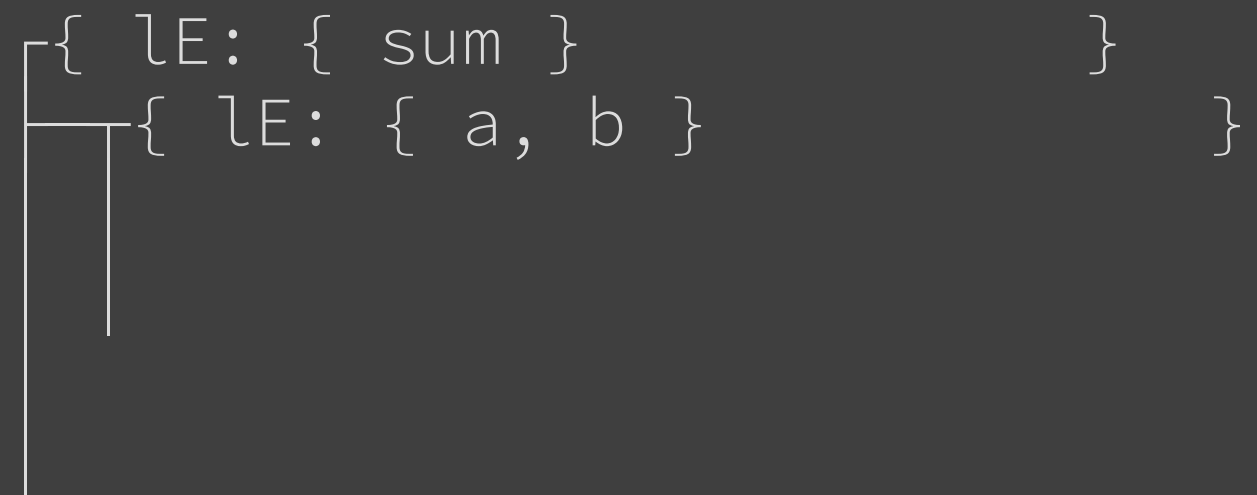
*Область видимости:*



*Код:*

```
function sum(a, b) {  
    return a + b;  
}  
  
sum(1, 2);
```

*Контекст исполнения:*




The diagram illustrates the execution context stack. It consists of two nested objects, each represented as a hash table with a 'this' property. The outer object has 'this' pointing to the global object. The inner object has 'this' pointing to the outer object. This represents the call stack where the function 'sum' is being executed with arguments '1' and '2'.

```
{ this: { this: {} }  
  this: { a: 1, b: 2 } }
```

*Код:*

```
function sum(a, b) {  
    return a + b;  
}  
  
sum(1, 2);
```

*Контекст исполнения:*



```
{ lE: { sum }, this: ??? }  
└─ { lE: { a, b }, this: ??? }
```

The diagram illustrates the execution context stack. It consists of two frames. The top frame represents the global context, with 'lE' pointing to the 'sum' function and 'this' set to '???'. The bottom frame represents the context created when 'sum' is called, with 'lE' pointing to the arguments {a, b} and 'this' set to '???'. A vertical line on the left connects the two frames, and a horizontal line separates them.

Значение `this` зависит от:

- I. Типа участка кода
- II. Как мы попали на этот участок
- III. Режимы работы интерпретатора

# I. Тип участка кода

## I. Тип участка кода. Глобальный

```
this.innerWidth;    // 1280
```

```
window.innerWidth; // 1280
```

## I. Тип участка кода. Глобальный

```
this.process.version;    // "v7.0.0"
```

```
global.process.version;  // "v7.0.0"
```

## I. Тип участка кода. Глобальный

```
console.log('Hello!');
```

```
global.console.log('Hello!');
```

```
this.console.log('Hello!');
```



I. Тип участка кода. Глобальный

```
this === global; // true
```

## I. Тип участка кода. Node.js модуль

```
// year-2016.js
```

```
module.exports.days = 366;
```

```
this.isLeapYear = true;
```

## I. Тип участка кода. Node.js модуль

```
// index.js  
  
var year2016 = require('./year-2016');  
  
year2016.days; // 366;  
year2016.isLeapYear; // true;
```

## II. Как попали на участок кода

## II. Как попали на участок кода. Простой вызов

```
function getSelf() {  
    return this;  
}  
  
getSelf(); // global
```

## II. Как попали на участок кода. Простой вызов

```
// year-2016.js  
  
module.exports.days = 366;  
  
function getSelf() {  
    return this;  
}
```

## II. Как попали на участок кода. Простой вызов

```
// year-2016.js  
  
module.exports.days = 366;  
  
function getSelf() {  
    return this;  
}  
  
getSelf(); // { days: 366 }
```

## II. Как попали на участок кода. Метод объекта

*Код:*

```
var block = {  
  innerHeight: 300,  
  
  getHeight: function () {  
    return this.innerHeight;  
  }  
}
```

*Значение this:*

```
???.innerHTML;
```



## II. Как попали на участок кода. Метод объекта

*Код:*

```
var block = {  
  innerHeight: 300,  
  
  getHeight: function () {  
    return this.innerHeight;  
  }  
}  
  
block.getHeight(); // 300
```

*Значение this:*

```
block.innerHeight;
```

## II. Как попали на участок кода. Метод объекта

*Код:*

```
var block = {  
  innerHeight: 300,  
  
  getHeight: function () {  
    return this.innerHeight;  
  }  
}  
  
var getHeight = block.getHeight;
```

*Значение this:*

???.innerHTML;

## II. Как попали на участок кода. Метод объекта

*Код:*

```
var block = {  
  innerHeight: 300,  
  
  getHeight: function () {  
    return this.innerHeight;  
  }  
}  
  
var getHeight = block.getHeight;  
getHeight(); // 1280
```

*Значение this:*

```
window.innerHeight;
```

# Заимствование метода

call

Метод `call()` вызывает  
функцию с указанным  
значением `this` и индивидуально  
предоставленными  
аргументами.

`Function.prototype.call()` - JavaScript | MDN

```
fun.call(thisArg, arg1, arg2, ...);
```

## II. Как попали на участок кода. Заимствование метода

*Код:*

```
var mike = {  
  age: 24,  
  
  getAge: function () {  
    return this.age;  
  }  
}  
  
var anna = {  
  age: 21  
}
```

*Значение this:*

???.age;

## II. Как попали на участок кода. Заимствование метода

*Код:*

```
var mike = {  
  age: 24,  
  
  getAge: function () {  
    return this.age;  
  }  
}  
  
var anna = {  
  age: 21  
}  
  
mike.getAge.call(anna); // 21
```

*Значение this:*

anna.age;



## II. Как попали на участок кода. Заимствование метода

```
function func() {  
  var args = [].slice.call(arguments);  
}
```

Метод `apply()` вызывает  
функцию с указанным  
значением `this` и аргументами,  
предоставленными в виде  
массива.

[Function.prototype.apply\(\) - JavaScript | MDN](#)

```
fun.apply(thisArg, [arg1, arg2]);
```

## apply

```
Math.min(4, 7, 2, 9); // 2
```

```
var arr = [4, 7, 2, 9];  
Math.min(arr); // NaN
```

```
Math.min.apply(Math, arr); // 2
```

```
Math.min.apply(null, arr); // 2
```

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    });  
  }  
}
```

*Значение this:*

```
???.items  
???.name
```

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    });  
  }  
}  
  
person.showItems();
```

*Значение this:*

```
person.items  
???.name
```

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    });  
  }  
}  
  
person.showItems();
```

*Значение this:*

```
person.items  
global.name
```

## Результат

```
'undefined has keys'  
'undefined has phone'  
'undefined has banana'
```

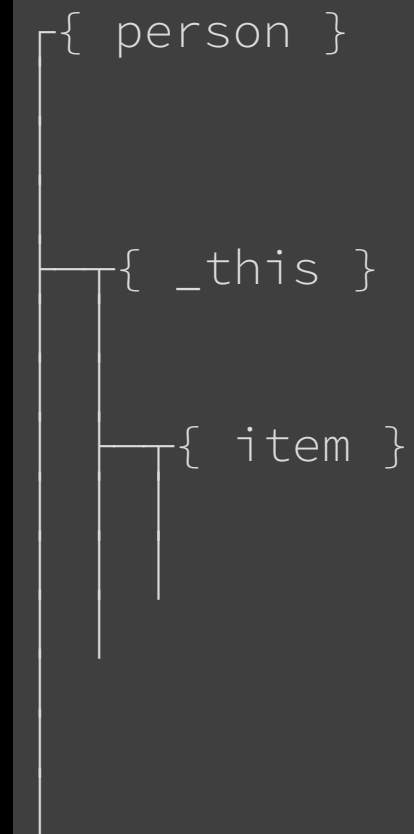


## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    var _this = this;  
  
    this.items.map(function (item) {  
      return _this.name + ' has ' + item;  
    });  
  }  
}  
  
person.showItems();
```

*Область видимости:*







Замыкание!



## Результат

```
'Sergey has keys '  
'Sergey has phone '  
'Sergey has banana '
```

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    }, this);  
  }  
}
```

*Значение this:*

```
???.items  
???.name
```

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    }, this);  
  }  
}  
  
person.showItems();
```

*Значение this:*

```
person.items  
person.name
```

## Результат

```
'Sergey has keys '  
'Sergey has phone '  
'Sergey has banana '
```

Метод `bind()` создаёт новую функцию, которая при вызове устанавливает в качестве контекста выполнения `this` предоставленное значение.

<...>

[Function.prototype.bind\(\) - JavaScript | MDN](#)

```
fun.bind(thisArg, arg1, arg2, ...);
```



## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    }).bind(this);  
  }  
}
```

*Значение this:*

???.items  
???.name

## II. Как попали на участок кода. Callback

*Код:*

```
var person = {  
  name: 'Sergey',  
  items: ['keys', 'phone', 'banana'],  
  
  showItems: function () {  
    this.items.map(function (item) {  
      return this.name + ' has ' + item;  
    }).bind(this);  
  }  
}  
  
person.showItems();
```

*Значение this:*

```
person.items  
person.name
```

## Результат

```
'Sergey has keys '  
'Sergey has phone '  
'Sergey has banana '
```

# myBind

```
Function.prototype.myBind = function(_this) {  
    var fn = this;  
    var args = [].slice.call(arguments, 1);  
  
    return function () {  
        var curArgs = [].slice.call(arguments);  
  
        return fn.apply(_this, args.concat(curArgs));  
    };  
};
```

## Частичное применение

```
Math.pow(2, 3); // 8  
Math.pow(2, 10); // 1024
```

```
var binPow = Math.pow.bind(null, 2);
```

```
binPow(3); // 8  
binPow(10); // 1024
```

## II. Как попали на участок кода. Конструктор

*Код:*

```
function User () {  
  return {  
    age: 24,  
  
    showAge: function () {  
      console.log(this.age);  
    }  
  }  
}
```

*Значение this:*

???.age

## II. Как попали на участок кода. Конструктор

*Код:*

```
function User () {  
  return {  
    age: 24,  
  
    showAge: function () {  
      console.log(this.age);  
    }  
  }  
}  
  
var mike = new User();  
mike.showAge(); // 24
```

*Значение this:*

mike.age

### III. Режим работы интерпретатора



### III. Режим работы интерпретатора. Режим СОВМЕСТИМОСТИ

```
function getSelf() {  
    return this;  
}  
  
getSelf(); // global
```

### III. Режим работы интерпретатора. Строгий режим

```
function getSelf() {  
    'use strict';  
  
    return this;  
}  
  
getSelf(); // undefined
```

eval

# eval

```
var temerature = 12;  
eval('temerature + 5'); // 17
```

# eval

*Код:*

```
var person = {  
  name: 'Sergey',  
  
  showName: function () {  
    return eval('this.name');  
  }  
}
```

*Значение this:*

???.name

# eval

*Код:*

```
var person = {  
  name: 'Sergey',  
  
  showName: function () {  
    return eval('this.name');  
  }  
}  
  
person.showName(); // Sergey
```

*Значение this:*

person.name

# eval

*Код:*

```
var person = {  
  name: 'Sergey',  
  
  showName: function () {  
    var evil = eval;  
  
    return evil('this.name');  
  }  
}
```

*Значение this:*

???.name

# eval

*Код:*

```
var person = {  
  name: 'Sergey',  
  
  showName: function () {  
    var evil = eval;  
  
    return evil('this.name');  
  }  
}  
  
person.showName(); // ''
```

*Значение this:*

general.name



# Сегодня

- Контекст исполнения
- this
  - Определяется в момент интерпретации
  - Тип участка кода
  - Как мы на него попали
  - Режим интерпретатора