**CANVAS**

**/\***

**\* AshAlom Gauge Meter. Version 2.0.0**

**\*/**





## What is this?

In JavaScript, the this keyword refers to an **object**.

**Which** object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

|  |
| --- |
| In an object method, this refers to the **object**. |
| Alone, this refers to the **global object**. |
| In a function, this refers to the **global object**. |
| In a function, in strict mode, this is undefined. |
| In an event, this refers to the **element** that received the event. |
| Methods like call(), apply(), and bind() can refer this to **any object**. |

## Explicit Function Binding

The call() and apply() methods are predefined JavaScript methods.

They can both be used to call an object method with another object as argument.

## See Also:

[The Function call() Method](https://www.w3schools.com/js/js_function_call.asp)

[The Function apply() Method](https://www.w3schools.com/js/js_function_apply.asp)

[The Function bind() Method](https://www.w3schools.com/js/js_function_bind.asp)

The example below calls person1.fullName with person2 as an argument, **this** refers to person2, even if fullName is a method of person1:

### Example

const person1 = {  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const person2 = {  
  firstName:"John",  
  lastName: "Doe",  
}  
  
// Return "John Doe":  
person1.fullName.call(person2);

It doesn't need an index, since this provides the context. As noted by the [docs](http://api.jquery.com/jQuery.each/), "The value can also be accessed through the this keyword." This is accomplished by using [call](https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Objects/Function/call). Something like:

userFunction.call(valueOfElement, indexInArray, valueOfElement);

$.fn.imgAreaSelect = function (options)

means the function is being added to the prototype. This allows it to be used with any instance of the jQuery object.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call>

# Function.prototype.call()

The **call()** method of [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) instances calls this function with a given this value and arguments provided individually.

*function Product(name, price) {*

*this.name = name;*

*this.price = price;*

*}*

*function Food(name, price) {*

*Product.call(this, name, price);*

*this.category = 'food';*

*}*

*console.log(new Food('cheese', 5).name);*

*// Expected output: "cheese"*

(function( $ ){

$.fn.createPassword = function( passBase, element, object) {

.....

return this;

};

})( jQuery );

Это расширение библиотеки jQuery методом createPassword. По виду типичный [IIFE](https://developer.mozilla.org/en-US/docs/Glossary/IIFE) с параметром jQuery

Соответственно "Password1Base", "Element", { Constructor: function(params) ...} в

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

## This Precedence

To determine which object this refers to; use the following precedence of order.

|  |
| --- |
|  |
| Precedence | Object |
| 1 | bind() |
| 2 | apply() and call() |
| 3 | Object method |
| 4 | Global scope |

// This is a function constructor:  
function myFunction(arg1, arg2) {  
  this.firstName = arg1;  
  this.lastName  = arg2;  
}  
  
// This creates a new object  
const myObj = new myFunction("John", "Doe");  
  
// This will return "John"  
myObj.firstName;

***$.fn.guageMeter = function(t)***

***{***

***}***

*$('#GaugeMeter\_1').gaugeMeter({ percent: 15 });*

t- вход парам-р

**var defaults = $.extend(**

{

id: ‘’, percent: 0, used: null, min: null, total: null, size: 100, prepend: '', append: '', theme: 'Red-Gold-Green',

color: '', back: 'RGBa(0,0,0,.06)', width: 3, style: 'Full', stripe: '0', animationstep: 1, animate\_gauge\_colors: false,

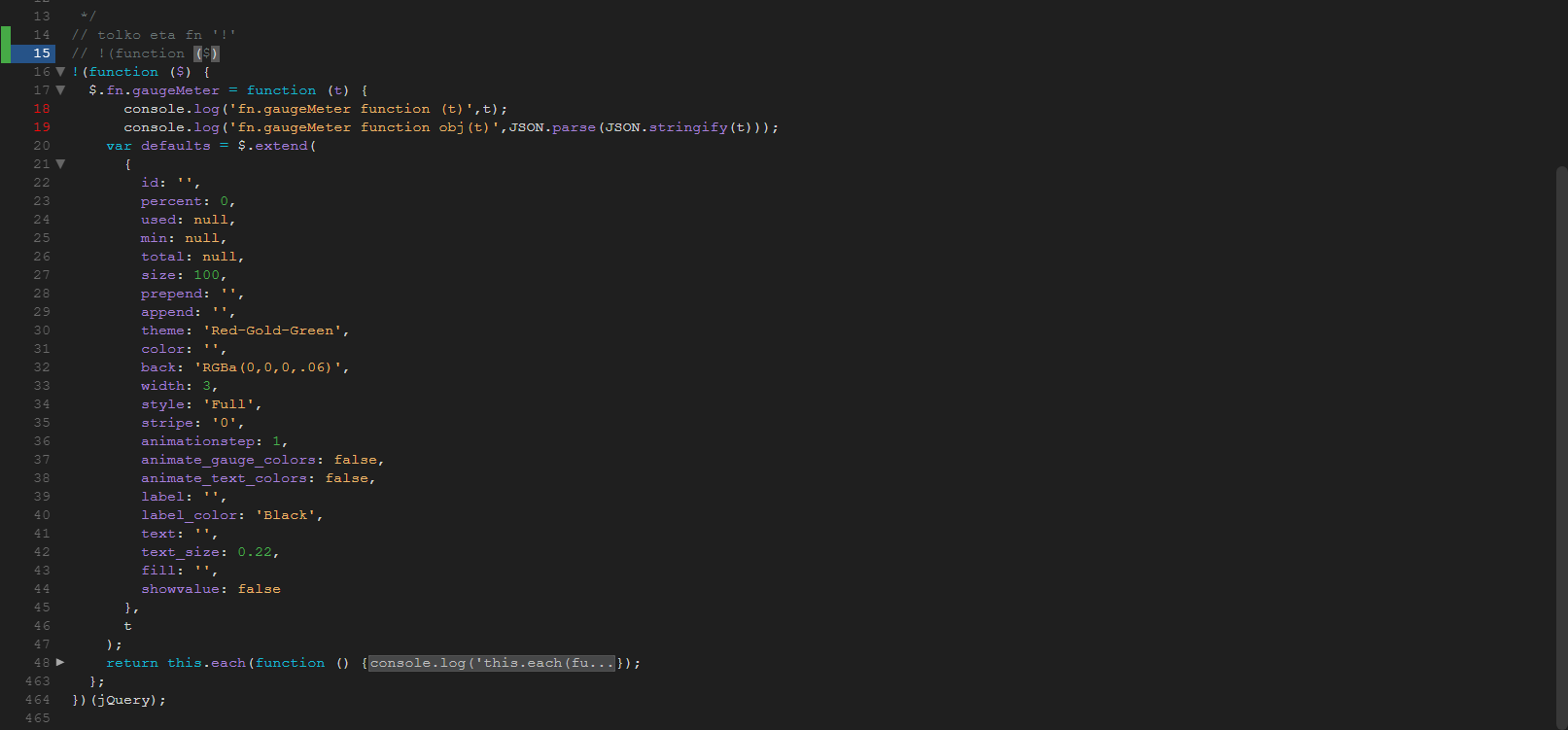
animate\_text\_colors: false, label: '', label\_color: 'Black', text: '', text\_size: 0.22, fill: '', showvalue: false

},

t

An **IIFE** (Immediately Invoked Function Expression)

https://web.archive.org/web/20171201033208/http://benalman.com/news/2010/11/immediately-invoked-function-expression/#iife



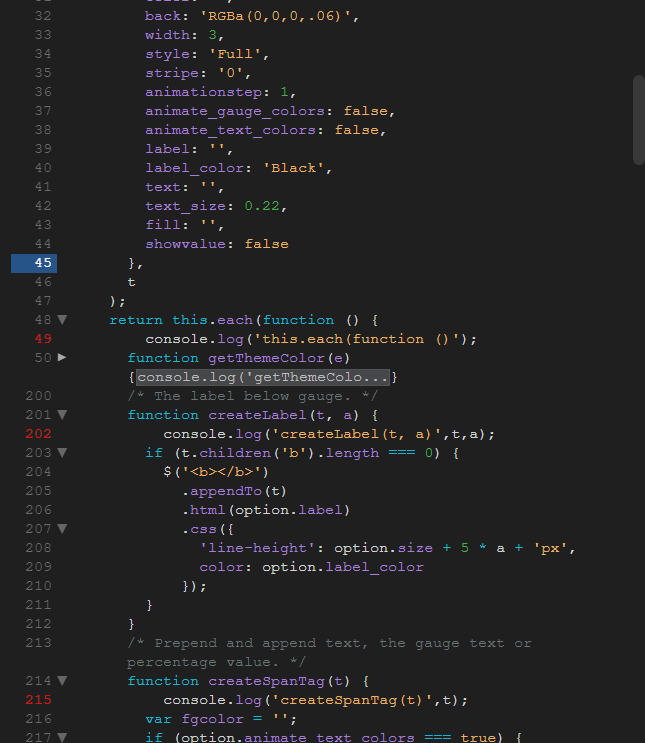
1. The first is the anonymous function with lexical scope enclosed within the [Grouping Operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Grouping) (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.
2. The second part creates the immediately invoked function expression () through which the JavaScript engine will directly interpret the function.

Эта ф-я вызывается каждый раз

**return this.each(function ()**

**getThemeColor**

эта ф-я палитра цветов для каждой цветовой гаммы



**Учавствует**

**return option.color;**

**и конструкции типа**

**'Red-Gold-Green' === option.theme &&**

**(e > 0 && (t = '#d90000'),**

**…**

**'White' === option.theme && (t = '#fff'),**

**'Black' === option.theme && (t = '#000'),**

**t**

**);**

И выбирает аргумент t

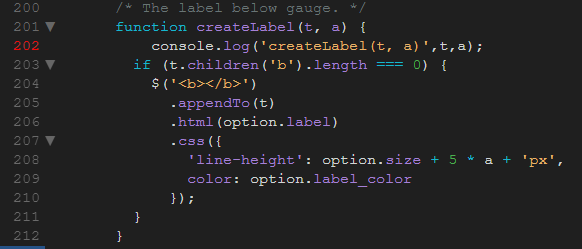
Что значит === в JS?

Оператор строгого равенства === **проверяет равенство без приведения типов**. Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования

**Созд лэбел под гагой**

t – text

a - size



до

1. "<span style=\"line-height: 100px; font-size: 22px;\">1.5<u> %</u></span><b style=\"line-height: 138.462px; color: black;\"></b><canvas width=\"100\" height=\"100\"></canvas>"

После

1. "<div class=\"GaugeMeter gaugeMeter\" id=\"GauAvPress\" data-append=\" %\" data-size=\"100\" data-width=\"15\" data-style=\"Arch\" data-animationstep=\"0\" data-id=\"GauAvPress\" style=\"width: 100px;\"><span style=\"line-height: 100px; font-size: 22px;\">1.5<u> %</u></span><b style=\"line-height: 138.462px; color: black;\"></b><canvas width=\"100\" height=\"100\"></canvas></div>"
2. outerText: "1.5 %"

/\* Prepend and append text, the gauge text or percentage value. \*/

function createSpanTag(t) {

var fgcolor = '';

if (option.animate\_text\_colors === true) {

fgcolor = option.fgcolor;

}

var child = t.children('span');

if (child.length !== 0) {

child.html(r).css({ color: fgcolor });

return;

}

if (option.text\_size <= 0.0 || Number.isNaN(option.text\_size)) {

option.text\_size = 0.22;

}

if (option.text\_size > 0.5) {

option.text\_size = 0.5;

}

$('<span></span>')

.appendTo(t)

.html(r)

.css({

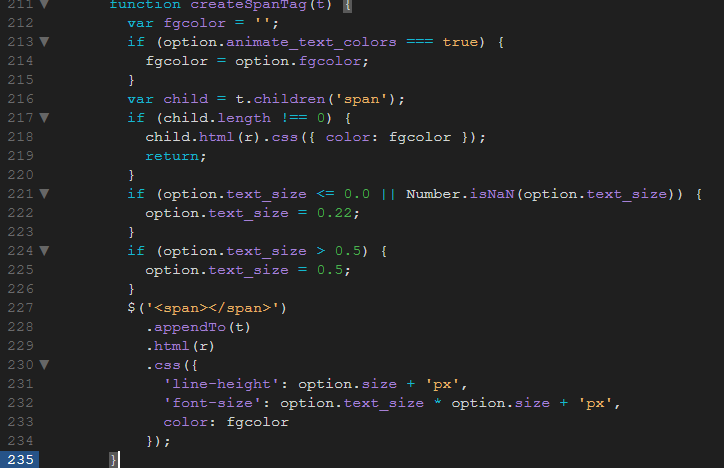
'line-height': option.size + 'px',

'font-size': option.text\_size \* option.size + 'px',

color: fgcolor

});

}



/\* Get data attributes as options from div tag. Fall back to defaults when not exists. \*/

function getDataAttr(t) {

$.each(dataAttr, function (index, element) {

if (t.data(element) !== undefined && t.data(element) !== null) {

option[element] = t.data(element);

} else {

option[element] = $(defaults).attr(element);

}

if (element === 'fill') {

s = option[element];

}

if (

(element === 'size' ||

element === 'width' ||

element === 'animationstep' ||

element === 'stripe') &&

!Number.isInteger(option[element])

) {

option[element] = parseInt(option[element]);

}

if (element === 'text\_size') {

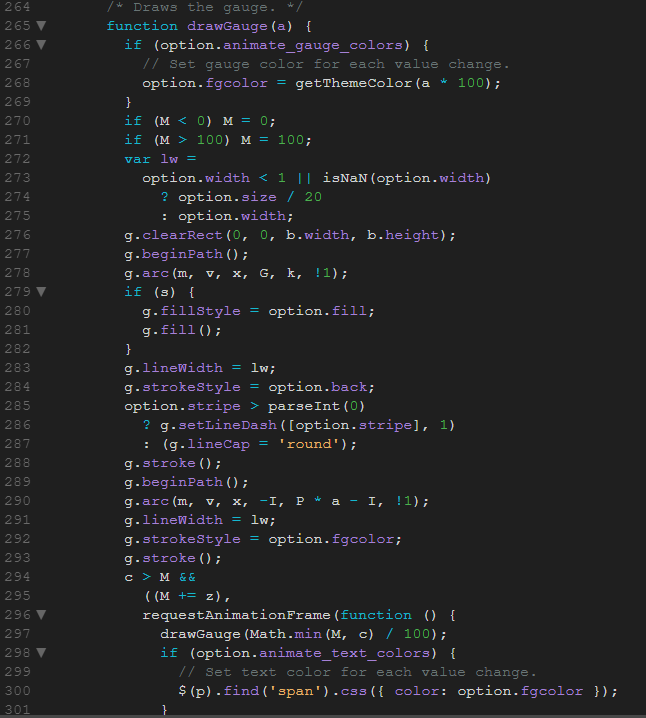
option[element] = parseFloat(option[element]);

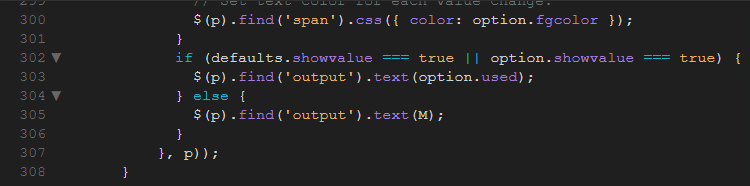
}

});

}







/\* Draws the gauge. \*/

function drawGauge(a) {

if (option.animate\_gauge\_colors) {

// Set gauge color for each value change.

option.fgcolor = getThemeColor(a \* 100);

}

if (M < 0) M = 0;

if (M > 100) M = 100;

var lw =

option.width < 1 || isNaN(option.width)

? option.size / 20

: option.width;

g.clearRect(0, 0, b.width, b.height);

g.beginPath();

g.arc(m, v, x, G, k, !1);

if (s) {

g.fillStyle = option.fill;

g.fill();

}

g.lineWidth = lw;

g.strokeStyle = option.back;

option.stripe > parseInt(0)

? g.setLineDash([option.stripe], 1)

: (g.lineCap = 'round');

g.stroke();

g.beginPath();

g.arc(m, v, x, -I, P \* a - I, !1);

g.lineWidth = lw;

g.strokeStyle = option.fgcolor;

g.stroke();

c > M &&

((M += z),

requestAnimationFrame(function () {

drawGauge(Math.min(M, c) / 100);

if (option.animate\_text\_colors) {

// Set text color for each value change.

$(p).find('span').css({ color: option.fgcolor });

}

if (defaults.showvalue === true || option.showvalue === true) {

$(p).find('output').text(option.used);

} else {

$(p).find('output').text(M);

}

}, p));

}

$(this).attr('data-id', $(this).attr('id'));

var r,

dataAttr = [

'percent',

'used',

'min',

'total',

'size',

'prepend',

'append',

'theme',

'color',

'back',

'width',

'style',

'stripe',

'animationstep',

'animate\_gauge\_colors',

'animate\_text\_colors',

'label',

'label\_color',

'text',

'text\_size',

'fill',

'showvalue',

'data'

],

option = {},

c = 0,

p = $(this),

s = false;

p.addClass('gaugeMeter');

getDataAttr(p);

if (Number.isInteger(option.used) && Number.isInteger(option.total)) {

var u = option.used;

var t = option.total;

if (Number.isInteger(option.min)) {

if (option.min < 0) {

t -= option.min;

u -= option.min;

}

}

c = u / (t / 100);

} else {

if (Number.isInteger(option.percent)) {

c = option.percent;

} else {

c = parseInt(defaults.percent);

}

}

if (c < 0) c = 0;

if (c > 100) c = 100;

if (

option.text !== '' &&

option.text !== null &&

option.text !== undefined

) {

if (

option.append !== '' &&

option.append !== null &&

option.append !== undefined

) {

r = option.text + '<u>' + option.append + '</u>';

} else {

r = option.text;

}

if (

option.prepend !== '' &&

option.prepend !== null &&

option.prepend !== undefined

) {

r = '<s>' + option.prepend + '</s>' + r;

}

} else {

if (defaults.showvalue === true || option.showvalue === true) {

r = '<output>' + option.used + '</output>';

} else {

r = '<output>' + c.toString() + '</output>';

}

if (

option.prepend !== '' &&

option.prepend !== null &&

option.prepend !== undefined

) {

r = '<s>' + option.prepend + '</s>' + r;

}

if (

option.append !== '' &&

option.append !== null &&

option.append !== undefined

) {

r = r + '<u>' + option.append + '</u>';

}

}

option.fgcolor = getThemeColor(c);

createSpanTag(p);

if (

option.style !== '' &&

option.style !== null &&

option.style !== undefined

) {

createLabel(p, option.size / 13);

}

$(this).width(option.size + 'px');

var b = $('<canvas></canvas>')

.attr({ width: option.size, height: option.size })

.get(0),

g = b.getContext('2d'),

m = b.width / 2,

v = b.height / 2,

x = (360 \* option.percent \* (Math.PI / 180), b.width / 2.5),

k = 2.3 \* Math.PI,

G = 0,

M = 0 === option.animationstep ? c : 0,

z = Math.max(option.animationstep, 0),

P = 2 \* Math.PI,

I = Math.PI / 2;

var child = $(this).children('canvas');

if (child.length !== 0) {

/\* Replace existing canvas when new percentage was written. \*/

child.replaceWith(b);

} else {

/\* Initially create canvas. \*/

$(b).appendTo($(this));

}

if ('Semi' === option.style) {

k = 2 \* Math.PI;

G = 3.13;

P = 1 \* Math.PI;

I = Math.PI / 0.996;

} else if ('Arch' === option.style) {

k = 2.195 \* Math.PI;

G = 655.99999;

P = 1.4 \* Math.PI;

I = Math.PI / 0.8335;

}

drawGauge(M / 100);

});

"fn.gaugeMeter function (t)" // [object Object]

{

"style": "Full",

"percent": 50,

"text": "1.5"

}

"fn.gaugeMeter function obj(t)" // [object Object]

{

"style": "Full",

"percent": 50,

"text": "1.5"

}

"this.each(function ()"

"getDataAttr(t)" // [object Array] (23)

["percent","used","min","total","size","prepend","append","theme","color","back","width","style","stripe","animationstep","animate\_gauge\_colors","animate\_text\_colors","label","label\_color","text","text\_size","fill","showvalue","data"]

"getThemeColor(e)" 50

"createSpanTag(t)" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

"<span></span> t r" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} "1.5<u> %</u>"

"createLabel(t, a)" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

7.6923076923076925

"drawGauge(a)" 0.5

"fn.gaugeMeter function (t)" // [object Object]

{

"style": "Semi",

"percent": 50,

"text": "1.5"

}

"fn.gaugeMeter function obj(t)" // [object Object]

{

"style": "Semi",

"percent": 50,

"text": "1.5"

}

"this.each(function ()"

"getDataAttr(t)" // [object Array] (23)

["percent","used","min","total","size","prepend","append","theme","color","back","width","style","stripe","animationstep","animate\_gauge\_colors","animate\_text\_colors","label","label\_color","text","text\_size","fill","showvalue","data"]

"getThemeColor(e)" 50

"createSpanTag(t)" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

"<span></span> t r" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} "1.5<u> %</u>"

"createLabel(t, a)" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} 7.6923076923076925

**jQuery Syntax**

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

* A $ sign to define/access jQuery
* A (*selector*) to "query (or find)" HTML elements
* A jQuery *action*() to be performed on the element(s)

**Что такое $.extend**

Соединяет 2 и более объектов

When two or more object arguments are supplied to $.extend(), properties from all of the objects are added to the target object. Arguments that are null or undefined are ignored.

If only one argument is supplied to $.extend(), this means the target argument was omitted. In this case, the jQuery object itself is assumed to be the target. By doing this, you can add new functions to the jQuery namespace. This can be useful for plugin authors wishing to add new methods to JQuery.

Keep in mind that the target object (first argument) will be modified, and will also be returned from $.extend(). If, however, you want to preserve both of the original objects, you can do so by passing an empty object as the target:

|  |  |
| --- | --- |
| 1 | **var** object = $.extend({}, object1, object2); |

(function($){$(function(){

$('select').styleThis();

})})($)

$(function(){}) — это мы запускаем действие в момент загрузки страницы. А оборачиваем в анонимную функцию (function($){ })($);, чтобы избежать конфликта имен с $.  
  
Автор плагина предусмотрел случай, когда доллар занят под другую библиотеку/функцию/переменную. Чтобы избежать конфликта имен, достаточно будет заменить доллар на jQuery, вот так (хотя сразу так и нужно было писать):

(function($){$(function(){

$('select').styleThis();

})})(jQuery);

Вызов функции $() с аргумментом в виде анонимной функции, внутри замыкания (closure) в которое передаётся $. Весь этот изврат нужен для того что бы не было проблем со сборкой мусора... на практике в JQuery это просто выполнение функции по событию onDocumentReady.

Что означает запись $.function в javascript? Пример:

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

Давайте разбирать [$ - это псевдоним](https://stackoverflow.com/questions/8667736/what-does-the-sign-mean-in-jquery-or-javascript/8667760#:%7E:text=In%20jQuery%2C%20the%20%24%20sign%20is,an%20alias%20to%20a%20function.&text=The%20jQuery%20syntax%20is%20tailor,on%20the%20element(s).&text=The%20%24%20symbol%20simply%20invokes%20the%20jQuery%20library%27s%20selector%20functionality.) для jQuery()

Допустим что есть такой код

(function( $ ){

$.fn.createPassword = function( passBase, element, object) {

.....

return this;

};

})( jQuery );

Это расширение библиотеки jQuery методом createPassword. По виду типичный [IIFE](https://developer.mozilla.org/en-US/docs/Glossary/IIFE) с параметром jQuery

Соответственно "Password1Base", "Element", { Constructor: function(params) ...} в

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

это параметры вызова $.createPassword расширения jQuery

It doesn't need an index, since this provides the context. As noted by the [docs](http://api.jquery.com/jQuery.each/), "The value can also be accessed through the this keyword." This is accomplished by using [call](https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Objects/Function/call). Something like:

userFunction.call(valueOfElement, indexInArray, valueOfElement);

$.fn.imgAreaSelect = function (options)

means the function is being added to the prototype. This allows it to be used with any instance of the jQuery object.

В JavaScript, очень распространена практика использования восклицательного знака перед функцией. Это может вызвать некоторое недоумение, особенно у тех, кто только начинает изучать язык. Чтобы разобраться в этом, рассмотрим следующий пример:

|  |  |
| --- | --- |
| 1  2  3 | !function() {      // Какой-то код...  }(); |

Вот что происходит. Здесь восклицательный знак (!) используется перед функцией для того, чтобы JavaScript интерпретировал выражение как функциональное выражение, а не как объявление функции.

В JavaScript, функциональные выражения и объявления функций обрабатываются по-разному. Объявления функций «всплывают» в верхушку своего контекста выполнения, что позволяет вызывать функцию до того, как она была объявлена. С другой стороны, функциональные выражения не «всплывают» и могут быть вызваны только после того, как они были определены.

Когда перед функцией стоит восклицательный знак, JavaScript принимает это как функциональное выражение, а не как объявление функции. Это позволяет немедленно вызвать функцию.

В этом контексте, восклицательный знак действует как логическое отрицание. Он превращает результат функции в булево значение и затем инвертирует его. Но поскольку цель здесь в основном состоит в том, чтобы вызвать функцию, а не получить булево значение, результат этого отрицания обычно игнорируется.

Важно заметить, что вместо восклицательного знака можно использовать и другие операторы, такие как +, -, ~ или void. Они все смогут превратить объявление функции в функциональное выражение, позволяя его немедленно вызвать.

В итоге, восклицательный знак перед функцией в JavaScript — это просто способ сделать функцию немедленно вызываемой.

The each function can take a function accepting an index as a parameter, but it's optional.For simplicity's sake,

.each

 was implemented to have

this

 refer to the current element.However,

.each

can accept an index as a parameter to it's callback.There's an example of that usage in the jQuery API

$('li').each(function(index) {

alert(index + ': ' + $(this).text());

});

The .each() method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery object. Each time the callback runs, it is passed the current loop iteration, beginning from 0. More importantly, the callback is fired in the context of the current DOM element, so the keyword this refers to the element.

$.each(fn) calls fn for each element contained in the current context. Each time it calls fn, it passes the "current" element as this.

So in the following example:

$("div").each(function() {

alert(this.className);

});

Will pop up one alert for each <div> in the DOM, and display the class name of each.