Basic about jQuery

This article will explain the basic concepts commonly used in jQuery such as: string (string), number (number), Boolean, array (array), function, parameter, context, ...

jQuery is a framework built on the features of JavaScript. So while developing applications using jQuery, you can use all the other functions and features that are supported in JavaScript.

This article will explain the basic concepts commonly used in jQuery such as: string (string), number (number), Boolean, array (array), function, parameter, context, .

Basic concepts in jQuery

1. Object string in jQuery
2. Object number in jQuery
3. Boolean objects in jQuery
4. Object object in jQuery
5. Object array in jQuery
6. Function in jQuery
7. The parameters in jQuery
8. Context in jQuery
9. Scope (Scope) in jQuery
10. Callback in jQuery
11. The Closure in jQuery
12. Proxy Pattern in jQuery
13. The functions are available in jQuery
14. Document Object Model (DOM)

Object string in jQuery

A string in JavaScript is a constant object containing 0, 1 or more characters.

The following is a valid example of a string in JavaScript.

```
"This is a JavaScript string"
'This is a JavaScript string'
'This "is really" a JavaScript string'
"This' is really 'a JavaScript string'
```

Object number in jQuery

The number object in JavaScript is IEEE 754 standard double-precision format (64 bit) values. They are constant, like string objects.

The following is a valid example of some in JavaScript.

```
5350
120.27
0.26
```

Boolean objects in jQuery

A Boolean in JavaScript can get true or false values. If a number is 0, its default is false. If a string is empty, the default is false.

Below are valid Boolean examples in JavaScript.

```
true // true
false // false
0 // false
1 // true
"" // false
"hello" // true
```

Object object in jQuery

JavaScript supports well the Object concept. You can create an Object by using Object Literal as follows:

```
var emp = {
name: "Zara",
age: 10
};
```

You can write and read the properties of an Object using the dot symbol (.) As follows:

```
// Get the properties of the object
emp.name // ==> Zara
emp.age // ==> 10

// Set properties for the object
emp.name = "QTM" // == QTM
emp.age = 20 // == 20
```

Object array in jQuery

You can define arrays using the following Array Literal:

```
  var x = [];
  var y = [1, 2, 3, 4, 5];
```

Arrays have length attributes, which are useful when repeating:

```
 var  x  = [ 1 , 2 , 3 , 4 , 5 ]; for ( var  i  = 0 ;  i    x . length ;  i ++) {
?ó v?i x[i] }
```

Function in jQuery

A function in JavaScript can be named or anonymous. A named function can be defined by the *function* keyword as follows:

```
 function  named (){ // làm gì ?ó ? ?ây }
```

An anonymous function can be defined in the same way as a regular function but it will not have any name.

An anonymous function can be assigned to a variable or passed to a method like this:

```
 var  handler  = function (){ // Làm gì ?ó ? ?ây }
```

jQuery uses an anonymous function very often like below:

```
 $ ( document ). ready ( function (){ // Làm gì ?ó ? ?ây });
```

The parameters in jQuery

The parameters in JavaScript are a type of Array that has a length attribute. The following example explains this:

```
 function  func ( x ){  console . log ( typeof  x ,  arguments . length ); }  fu
```

The Argument object also has a callee attribute, which refers to the function you are in. For example:

```
 function  func () { return  arguments . callee ; }  func (); // ==> func
```

Context in jQuery

Famous keywords in JavaScript are this reference to the current Context. In a function, this can change, depending on how the function is called.

```
 $ ( document ). ready ( function () { // cái này tham chi?u t?
i window.document });  $ ( "div" ). click ( function () { // tham chi?u t?
i div DOM element });
```

You can specify the Context for once by using the call () and apply () methods.

The difference between them is how they pass parameters. call () passes all parameters through parameters to the function, while apply () accepts an array as parameters.

```
function  scope () {  console . log ( this ,  arguments . length ); }  scope ()
```

Scope (Scope) in jQuery

The scope of a variable is the area in your program where that variable is defined. The variable in JavaScript will have only two ranges:

1. **Global Variables -** A Global variable has a common scope, meaning that it is defined everywhere in your JavaScript code.
2. **Local Variables -** A Local variable will only be visible inside a function where it is defined. Function parameters are always Local for that function.

In the body of a function, a Local variable has a higher priority than the Global variable that has the same name.

```
 var  myVar  = "global" ; // ==> Khai báo bi?
n global function ( ) { var  myVar  = "local" ; // ==> Khai báo bi?
n local  document . write ( myVar ); // ==> local }
```

Callback in jQuery

A callback is a pure JavaScript function that is passed some methods as a parameter or option. Some callbacks are events, called to give the user the opportunity to react when a certain state is activated.

The event system in jQuery uses these callbacks everywhere, for example:

```
$ ( "body" ). click ( function ( event ) {  console . log ( "clicked: " + event
```

Most callbacks provide parameters and a context. In the event-handler example, the callback is called with a parameter, an Event.

Some callbacks are required to return something, others return arbitrary values. To prevent form submissions, an Submit event handler can return false as follows:

```
$ ( "#myform" ). submit ( function () { return false ; });
```

The Closure in jQuery

Closure is created whenever a variable defined outside the current scope is accessed from within the internal scope.

The following example shows how the **counter** variable is visible in the functions create, increment, and print, but not visible outside them.

```
function  create () { var  counter  = 0 ; return {  increment : function () {  c
```

This pattern allows you to create objects with methods, which operate on data, but are not visible outside. You remember that **data hiding** is a very basic concept of object-oriented programs.

Proxy Pattern in jQuery

A Proxy is an object that can be used to control access to another element. It executes the same interface for this other object and passes on any method to it. This other object is often called a Real Subject.

A Proxy may be initiated at this Real Subject location and allow it to be accessed in remote mode. We can store jQuery's setArray method in a Closure and overwrite it as follows:

```
( function () { // log all calls to setArray var  proxied  =  jQuery . fn . setA
```

Example on its code **package** in a function to hide the **proxied** variable. After that, this Proxy logs all method calls and delegates that call to the original method. Using apply (this, arguments) ensures that the caller cannot pay attention to the difference between the original method and the delegated method.

The functions are available in jQuery

JavaScript comes with a set of useful functions attached to it. These methods can be used to manipulate String, Number, and Date.

The table below lists important JavaScript functions:

Method Description **charAt ()**

Returns the character at the given index.

concat ()

Connect two text strings and return a new string.

forEach ()

Call a function for each element of an array.

indexOf ()

Returns the index of the first occurrence within calling the String object with a given value, or -1 if not found.

length()

Returns the length of the string.

pop ()

Remove the last element of an array and return that element.

push ()

Add one or more elements to the end of an array and return the new length of the array.

reverse ()

Reverse the order of the elements in an array - the first element becomes the last and the last into the first.

sort ()

Sort the elements of an array.

substr ()

Returns the characters in an array starting from the given position from the number of characters specified.

toLowerCase ()

Returns the value of the calling string being converted to lower case.

toString ()

Returns the string representation of the numeric value.

toUpperCase ()

Returns the value of the calling string that is converted to uppercase.

Document Object Model (DOM)

DOM is a tree structure of various HTML elements, as follows:

```
Ví d? jQuery QTM


This is the first paragraph

 .

This is the second paragraph

 .

?ây là ?o?n v?n th? ba.
```

It will produce the following result:

```
 This is the first paragraph .
 This is the second paragraph .
 This is the third paragraph.
```

Here are some important points about the tree structure above:

1. An ancestor tag (root or ancestor) of all other elements; In other words, all other elements are descendants of the element.
2. The card is not descendant, but it is also the son.
3. Element

is the child of the element

, children of elements and, and brothers of elements other.

According to Tutorialspoint

Last post: What is jQuery?

Next article: Selector in jQuery