JavaScript Research

1. Scope là gì?

Trong Javascript chúng ta định nghĩa Scope nghĩa là phạm vi hoạt động của biến, đối tượng và function. Khi nào chúng tồn tại và khi nào có thể gọi được chúng.

Có 2 loại scope trong javascript là Global (phạm vi toàn cục) và Local (phạm phi nhất định).

1. Global Scope là gì?

Các biến khai báo ngoài các function được gọi là global. Biến này được dùng chung cho các function, các function có thể thấy được biến này và thay đổi giá trị của biến này.

|  |
| --- |
| var userName = "Bill";  function modifyUserName() {  userName = "Steve";  };  function showUserName() {  console.log(userName);  };  console.log(userName); // display Bill    modifyUserName();  showUserName();// display Steve |

Biến userName là global nên function modifyUserName và showUserName có thể thấy được và sử dụng được.

Các biến khai báo trong function nhưng không có từ khoá var, thì cũng được xem là biến có scope global.

|  |
| --- |
| function createUserName() {  userName = "Bill";  }  function modifyUserName() {  if(userName)  userName = "Steve";  };  function showUserName() {  console.log(userName);  }    createUserName();  showUserName(); // Bill  modifyUserName();  showUserName(); // Steve |

1. Local scope là gì?

Các biến được khai báo bên trong function và có từ khoá var được gọi là local. Biến local chỉ được sử dụng bên trong function. Các function khác bên ngoài không thể sử dụng được biến local này.

|  |
| --- |
| function createUserName() {  var userName = "Bill";  }  function showUserName() {  console.log(userName);  }  createUserName();  showUserName(); // throws error: userName is not defined |

Biến userName chỉ được sử dụng trong function createUserName mà thôi, các function khác không thấy được biến này.

1. Function scope

Nó cũng tương tự như local scope là đều nằm trong function. Sự khác biệt của function scope vs local scope là function scope ko dc đặt trong biểu thức

Cặp dấu {} như if, while, for trong khi khái niệm local scope thì có.

|  |
| --- |
| function A() {  var a = 1; //local & function scope  if(a == 1){  var b = 1 //local scope  }  }; |

* Có thể thấy rằng local scope có thể là function scope nhưng function scope chưa chắc đã là local scope

1. Lexical Scope

Bất cứ biến/object/ function được định nghĩa trong parent scope, đều có thể được truy cập bởi các scope con nhỏ hơn.VD:

|  |
| --- |
| var Favourite = function () {      var food = "icecream";           //parent scope        var myFavouriteFood = function () {          console.log("I love " + food);// ‘I love icream’      };      myFavouriteFood();  };  Favourite(); |

Tương tự :

|  |
| --- |
| var name = 'Xung';  var scope1 = function () {  // name is available here  var scope2 = function () {  // name is available here too  var scope3 = function () {  // name is also available here!  };  };  }; |

Nhưng ko thể truy cập ngược lại từ function con ra function cha.

|  |
| --- |
| function A() {  console.log(name);  function B() {  var name = “Xung”;  }  } |

1. Scope Chain

Khái niệm scope chain là khái niệm : khi một biến scope được gọi mà ko dc khai báo ở scope hiện tại thì js sẽ tự tìm đến biến đó ở scope cha. Nếu scope cha ko có thì sẽ báo lỗi còn có thì sẽ có 2 trường hợp: 1 là undefined hoặc là sẽ tìm dc.

VD:

|  |
| --- |
| function show() {      console.log(text); //undefined    }  show();  var text = “ashiba”; |

Mà muốn xuất ra từ ashiba thì phải khai báo ở trên

|  |
| --- |
| var text = “ashiba”;  function show() {      console.log(text); //’ashiba’    }  show(); |

? Tại sao khi khai báo biến text ở dưới function show() vs vd 1 thì nó vẫn chạy và ko báo lỗi mặc dù kq là undefined ?

* Bởi vì trong JS nó chạy theo cơ chế Hosting. Vậy cơ chế hosting là gì?

1. Hosting trong JS

**Hoisting** là một cơ chế Javascript. Các biến và hàm trong JS mà chúng ta khai báo ở thứ tự trước hoặc sau thì khi có dòng code gọi đến hàm hoặc biến đó thì cũng sẽ gọi dc kể cả khi ta viết nó sau code thực thi.

**VD:**

|  |
| --- |
| show(); // ‘ashiba’  function show() {      console.log("ashiba");  } |

Nhưng lưu ý vs biến. Ta sẽ lấy VD ở trên:

|  |  |
| --- | --- |
| var text;  function show() {      console.log(text); //undefined    }  show();  text = “ashiba”; | function show() {      console.log(text); //undefined    }  show();  var text = “ashiba”; |

Đều ra là 1 kết quả tương tự là undefined.

🡺Cho thấy rằng hàm thì có thể viết tùy ý dc nhưng đối vs biến thì phải cẩn thận thứ tự đặt biến.

Có 1 lưu ý nhỏ là: Nếu ko ghi gì hết thì mặc định kiểu là var.

|  |
| --- |
| text = "ashiba"  console.log(text)   //’ashiba’ |

Ta thử ghi ngược lại 2 dòng này vs biến kiểu var, let hoặc ko có.

|  |  |
| --- | --- |
| // Non type  console.log(text)   //Error  text = "ashiba" | //LET  console.log(text)   //Error  let text = "ashiba" |

|  |
| --- |
| //VAR  console.log(text)   //Undefined  var text = "ashiba" |

* Như ta có thể thấy, nếu ko ghi kiểu dữ liệu thì nó sẽ ngầm định là var, nhưng 2 kiểu này vẫn sẽ có sự khác biệt thông qua ví dụ trên.

1. Closure

Ta hãy xem đoạn code và giải thích sau:

// Tạo một biến toàn cục

var counter = 0;

//hàm đếm

function makeCounter() {

return counter += 1;

}

// Gọi hàm

makeCounter();

console.log(counter); // Kết quả: 1

makeCounter();

console.log(counter); // Kết quả: 2

// Thử thay đổi biến counter từ bên ngoài

counter = 10;

console.log(counter); // Kết quả: 10

\_Như ta thấy trong ví dụ trên, giá trị của biến **counter** có thể được thay đổi từ bất kỳ đâu trong chương trình mà không cần gọi hàm ***makeCounter().***

\_Nếu ta ko gọi hàm để thay đổi, mà ta thay đổi giá trị của biến counter trực tiếp bên ngoài thì biến đó cũng sẽ bị thay đổi. Điều này có thể sẽ ảnh hưởng đến việc kiểm soát dữ liệu sau này vô tình sẽ bị thay đổi giá trị của biến.

**Ta hãy thử làm bằng cách ko đặt biến global scope thì điều gì sẽ xảy ra:**

function makeCounter() {

// Tạo biến cục bộ

var counter = 0;

// Thao tác với biến counter

return counter += 1;

}

// Gọi hàm

console.log(makeCounter()); // Kết quả: 1

console.log(makeCounter()); // Kết quả: 1

\_Như ta thấy thì điều này đã làm được điều là ko cho biến counter dc thay đổi bên ngoài mà chỉ dc thay đổi khi gọi hàm makeCounter() nhưng kết quả thì lại ko nhưng mong đợi.

\_Vì mỗi khi hàm makeCounter() được gọi thì biến cục bộ bên trong hàm đó sẽ set lại bằng 0.

Vậy nếu chúng ta vẫn muốn đạt kết quả tương tự như ví dụ 1 mà lại không thay đổi biến counter từ bên ngoài thì làm thế nào?

* SỬ DỤNG CLOSURE

Về cơ bản, **Closure** là một **hàm bên trong** (**inner function**) có quyền truy cập vào phạm vi của hàm mẹ, ngay cả sau khi hàm mẹ đã thực thi xong.

Điều này được thực hiện bằng cách **tạo một hàm bên trong một hàm khác**. Hãy xem ví dụ sau để xem nó hoạt động như thế nào:

function makeCounter() {

var counter = 0; //Được hiểu như là this.counter = 0

// Hàm bên trong hàm

function make() { //return function make(){} cũng được

counter += 1; //this.counter += 1;

return counter; //return this.counter;

}

return make;

}

Khi ta gọi hàm makeCounter thì sẽ ko có chuyện gì xảy ra

makeCounter();

mà ta phải gắn cái function makeCounter cho một biến khác chứa nó và dùng biến đó như 1 function

/\* Thực thi hàm makeCounter() và lưu trữ giá trị

được trả về bằng biến myCounter \*/

var myCounter = makeCounter();

console.log(myCounter()); // Kết quả: 1

console.log(myCounter()); // Kết quả: 2

console.log(makeCounter()()); // Kết quả: 1

console.log(makeCounter()()); // Kết quả: 1

\_Biến myCounter được hiểu như là đối tượng lưu giá trị của biến counter.

\_Giá trị của **myCounter** bản chất là hàm ***make()*** bên trong, việc mỗi lần gọi hàm ***myCounter()*** sẽ là gọi hàm ***make()*** bên trong kia luôn chứ ko cần thông qua makeCounter() nữa.

\_Hàm make() ở đây là hàm closure(hàm ngoài return đến hàm trong thay đổi và lưu giá trị thì hàm đó sẽ là closure). Closures lưu trữ nội bộ các tham chiếu đến các biến bên ngoài của chúng và có thể truy cập và cập nhật các giá trị của chúng.

P/s: Làm thêm để xem kết quả tương tự:

function makeCounter() {

var counter = 0;

// Hàm bên trong hàm

function make() {

counter += 1;

function extra() {

counter += 1;

return counter;

}

}

return make;

}

/\* Thực thi hàm makeCounter() và lưu trữ giá trị

được trả về bằng biến myCounter \*/

var myCounter = makeCounter();

console.log(myCounter()()); // Kết quả: 1

console.log(myCounter()()); // Kết quả: 2

Nếu ta tạo 2 biến khác nhau cùng gọi hàm makeCounter() thì sao, kết quả bên dưới ha.

var myCounter = makeCounter();

console.log(myCounter()); // Kết quả: 1

console.log(myCounter()); // Kết quả: 2

console.log(myCounter()); // Kết quả: 3

var hisCounter = makeCounter();

console.log(hisCounter()); // Kết quả: 1

console.log(hisCounter()); // Kết quả: 2