Access a global variable in a browser environment:

[复制代码](javascript:void(0);)

myglobal = "hello"; // antipattern

console.log(myglobal); // "hello"

console.log(window.myglobal); // "hello"

console.log(window["myglobal"]); // "hello"

console.log(this.myglobal); // "hello"

[复制代码](javascript:void(0);)

1. The problem with Globals
   * Naming collisions
     1. Code not written by developers

• A third-party JavaScript library

• Scripts from an advertising partner

• Code from a third-party user tracking and analytics script

• Different kinds of widgets, badges, and buttons

* + 1. Implied globals

meaning that any variable you don't declare becomes a property of the global object.

**Solution** - Use var to declare variable inside the function.

[复制代码](javascript:void(0);)

function sum(x, y) {

var result = x + y;

return result;

}

// antipattern, do not use

function foo() {

var a = b = 0; // a is local but b becomes global

// ...

}

// The right way

function foo() {

var a, b;

// ...

a = b = 0; // both local

}

[复制代码](javascript:void(0);)

* + portability

Code to run in different environments (hosts), it's dangerous to use globals because you can accidentally overwrite a host object that doesn't exist in your original environment (so you thought the name was safe to use) but which does in some of the others.

1. Side Effects when Forgetting var

Difference between implied globals and explicitly defined ones—the difference is in the ability to undefine these variables using the delete operator

• Globals created with var(those created in the program outside of any function) cannot be deleted.

• Implied globals created without var(regardless if created inside functions) can be deleted.

[复制代码](javascript:void(0);)

// define three globals

var global\_var = 1;

global\_novar = 2; // antipattern

(function () {

global\_fromfunc = 3; // antipattern

}());

// attempt to delete

delete global\_var; // false

delete global\_novar; // true

delete global\_fromfunc; // true

// test the deletion

typeof global\_var; // "number"

typeof global\_novar; // "undefined"

typeof global\_fromfunc; // "undefined"

[复制代码](javascript:void(0);)

1. Access to the Global Object

Access the global object without hard-coding the identifier window, you can do the following from any level of nested function scope:

var global = (function () {

return this;

}());

1. Single var Pattern

• Provides a single place to look for all the local variables needed by the function

• Prevents logical errors when a variable is used before it's defined (see "Hoisting: A Problem with Scattered vars" )

• Helps you remember to declare variables and therefore minimize globals

• Is less code (to type and to transfer over the wire)

[复制代码](javascript:void(0);)

function func() {

var a = 1,

b = 2,

sum = a + b,

myobject = {},

i,

j;

// function body...

}

[复制代码](javascript:void(0);)

**Note:** all uninitialized and declared variables are initialized with the value undefined

[复制代码](javascript:void(0);)

function updateElement() {

var el = document.getElementById("result"),

style = el.style;

// do something with el and style...

}

[复制代码](javascript:void(0);)

1. Hoisting: A problem with Scattered vars

JavaScript enables you to have multiple var statements anywhere in a function, and **they all act as if the variables were declared at the top of the function**.

[复制代码](javascript:void(0);)

// antipattern

myname = "global"; // global variable

function func() {

// same as -> var myname = undefined;

alert(myname); // "undefined"

var myname = "local";

alert(myname); // "local"

}

func();

[复制代码](javascript:void(0);)