If you create a new function and assign it to the same variable that already holds another function, you’re overwriting the old function with the new one.

[复制代码](javascript:void(0);)

var scareMe = function () {

alert("Boo!");

scareMe = function () {

alert("Double boo!");

};

};

// using the self-defining function

scareMe(); // Boo!

scareMe(); // Double boo!

[复制代码](javascript:void(0);)

This pattern(lazy function definition) is useful when your function has some initial preparatory work to do and it needs to do it only once.

A drawback of the pattern is that any properties you’ve previously added to the original function will be lost when it redefines itself.

If the function is used with a different name, for example, assigned to a different variable or used as a method of an object, then the redefinition part will never happen and the original function body will be executed.

[复制代码](javascript:void(0);)

// 1. adding a new property

scareMe.property = "properly";

// 2. assigning to a different name

var prank = scareMe;

// 3. using as a method

var spooky = {

boo: scareMe

};

// calling with a new name

prank(); // "Boo!"

console.log(prank.property); // "properly"

// calling as a method

spooky.boo(); // "Boo!"

console.log(spooky.boo.property); // "properly"

// using the self-defined function

scareMe(); // Double boo!

console.log(scareMe.property); // undefined

[复制代码](javascript:void(0);)