2013-05-23

JavaScript quirk 7: inadvertent sharing of variables via closures

Labels: 12quirks, dev, javascript, jslang

[This post is part of a series on JavaScript quirks.]

Closures are a powerful JavaScript feature: If a function leaves the place where it was created, it still has access to all variables that existed at that place. This blog post explains how closures work and why one has to be careful w.r.t. inadvertent sharing of variables.

1. Closures

Let's start with an example of a closure:

```
function incrementorFactory(start, step) {
    return function () {  // (*)
        start += step;
        return start;
    }
}
```

This is how you use `incrementorFactory`:

```
> var inc = incrementorFactory(20, 2);
> inc()
22
> inc()
24
```

During all of its lifetime, the inner function (*) has access to the variables `start` and `step` of the outer function `incrementorFactory`. Thus, `incrementorFactory` returns not only the function, but somehow attaches the variables `start` and `step`. The data structure in which both variables are stored is called an `environment`. An environment is very similar to an object – it maps names to values. The function that is returned above contains a reference to the environment that was active at its birth, its *outer environment*. The combination function + environment is called a *closure*. The name stems from the fact that an environment "closes over" a function: It provides values for variables that were declared outside the function (so-called *free variables*).

When a function `f` is invoked, a new environment is created for its parameters and local variables. There is always a chain of environments:

- `f`'s environment
- `f`'s outer environment
- The outer environment of `f`'s outer environment
- ...
- The environment for global variables (the *global environment*)

When looking up the value of a variable, the complete chain is searched, starting with `f`'s environment.

2. The quirk: inadvertent sharing

Dr. Axel Rauschmayer

Free online books by Axel

Speaking JavaScript [up to ES5]
Exploring ES6

JavaScript training: Ecmanauten

Closures don't get snapshots of a certain point in time, they get "live" variables. The following is an example where that causes a problem.

```
var result = [];
for (var i=0; i < 5; i++) {
    result.push(function () { return i });  // (*)
}
console.log(result[3]()); // 5 (not 3)
```

When a function is created in line (*), the variable `i` has a certain value. You might expect that function to always return that value. Instead, the connection to the "live" `i` is never broken. That is, all functions in the array `result` share the same `i`, via their outer environment. And after the loop is finished, `i` has the value `5`.

One possible fix is to copy the current value of `i` via an IIFE [1]:

```
for (var i=0; i < 5; i++) {
    (function (i2) {  // snaphot of i
        result.push(function () { return i2 });
    }(i));
}
```

You can also use `bind()`, with a similar effect:

```
for (var i=0; i < 5; i++) {
    result.push(function (i2) { return i2 }.bind(null, i));
}
```

Another possibility is using `forEach` and the `range()` function of the Underscore.js library:

```
_.range(5).forEach(function (i) {
    result.push(function () { return i });
});
```

The above works, because `forEach` creates a new variable `i`, each time it calls its argument.

2.1. A practical example

Let's conclude with a more practical example. Two days ago, I implemented a user interface for the game Connect Four, as a demonstration of the DOM. It contained the following code snippet, which adds event listeners to links above the columns of the board.

```
for(var col=0; col < board4.DIM_X; col++) {
    document.getElementById('columnClick'+col)
            .addEventListener('click', function (col) {
                currentState.columnClick(col);
                event.preventDefault();
            }.bind(null, col));
}
```

An alternative is to use CSS classes instead of IDs and rewrite the above code:

```
Array.prototype.forEach.call(
    document.getElementsByClassName('columnClick'),
    function (elem, col) {  // (*)
        elem.addEventListener('click', function () {
            currentState.columnClick(col);
            event.preventDefault();
        });
    });
```

Again, each invocation of the function (*) creates a new variable `col` and no inadvertent sharing occurs.

The last post in this series will explain how ECMAScript 6 helps with the problem of inadvertent sharing.

3. Reference

[1] JavaScript quirk 6: the scope of variables

javascript (400)
computers (316)
life (194)
jslang (179)
esnext (156)
apple (107)
webdev (95)
mobile (83)
scitech (50)
hack (49)
mac (47)
google (39)
java (37)
ios (33)
business (32)
video (32)
clientjs (31)
hci (27)
entertainment (26)
nodejs (26)
society (26)
browser (25)
firefox (25)
html5 (24)
ipad (24)
movie (23)
psychology (22)
2ality (18)
tv (18)
android (17)
social (17)
chrome (16)
fun (16)
jsmodules (16)
tablet (16)
humor (15)
politics (15)
web (15)
cloud (14)
hardware (14)
microsoft (14)
software engineering (13)
blogging (12)
gaming (12)
eclipse (11)
gwt (11)
numbers (11)
programming languages (11)
app store (10)
media (10)
nature (10)
security (10)
semantic web (10)

5 Comments **The 2ality blog** 1 **Login**

Recommend **Share** **Sort by Best**

Join the discussion…

WebReflection • 4 years ago

typo also in the col example where `event` does not exist in the listener signature ... is also missing the classic/faster + **better** example of creating a function **outside** the loop.

```javascript

for (var
assign = function (i) {
return function () {
return i;
};
},
i = 0;
i < 5;
i++
) {
result.push(assign(i));
}

```

the reason this is the best way to solve the problem is that you don't have to bind a null context, you don't need bind at all, and you don't create useless garbage inside the loop.

regards

Reply • Share ›

Anders Ringqvist • 4 years ago

Remember Angus Croll´s "hack" from JSConfEU last year?

with ({i: i}) {
result.push(function () { return i });
}

I like how readable the code becomes and that you don't have to create a new copy of the function for every iteration. Too bad the FUD movement got with statement banned =P

Reply • Share ›

Axel Rauschmayer Mod → Anders Ringqvist • 4 years ago

It's not all FUD, there are some good reasons against `with` [1].
In ES6, your code becomes:

```
{
        let i2 = i;
        result.push( function () { return i2 } );
}
```

But the for-of loop will be even better. I'll explain both solutions in more detail in the last post of the "quirks" series.

[1] http://www.2ality.com/2011/06/...

Reply • Share ›

Guest • 4 years ago

Guest • 4 years ago

typo in bind example? Last i2 should be i, or I don't get it.

^ | v • Reply • Share ›

Axel Rauschmayer Mod ➦ Guest • 4 years ago

Yes, typo, thanks! Fixed.

^ | v • Reply • Share ›

Newer Post   Home   Older Post

Subscribe to: Post Comments (Atom)

- software (10)
- twitter (10)
- webos (10)
- 12quirks (9)
- education (9)
- jstools (9)
- photo (9)
- webcomponents (9)
- windows 8 (9)
- async (8)
- idea (8)
- iphone (8)
- itunes (8)
- scifi-fantasy (8)
- app (7)
- babel (7)
- bookmarklet (7)
- chromeos (7)
- english (7)
- es proposal (7)
- fringe (7)
- html (7)
- jsint (7)
- jsshell (7)
- thunderbolt (7)
- webapp (7)
- advancedjs (6)
- blogger (6)
- crowdsourcing (6)
- latex (6)
- lion (6)
- promises (6)
- ted (6)
- book (5)
- environment (5)
- gadget (5)
- googleio (5)
- intel (5)
- jsarrays (5)
- jshistory (5)
- layout (5)
- light peak (5)
- michael j. fox (5)
- music (5)
- pdf (5)
- polymer (5)
- shell (5)
- tc39 (5)
- template literals (5)
- underscorejs (5)
- vlc (5)
- __proto__ (4)
- coffeescript (4)
- concurrency (4)

dart (4)

facebook (4)

gimp (4)

googleplus (4)

health (4)

howto (4)

hp (4)

javafx (4)

kindle (4)

leopard (4)

macbook (4)

motorola (4)

münchen (4)

occupy (4)

pl fundamentals (4)

presenting (4)

publishing (4)

series (4)

textbook (4)

web design (4)

amazon (3)

asmjs (3)

back to the future (3)

bitwise_ops (3)

css (3)

es2016 (3)

flattr (3)

fluentconf (3)

food (3)

foreign languages (3)

house (3)

icloud (3)

info mgmt (3)

jsfuture (3)

jsstyle (3)

linux (3)

mozilla (3)

python (3)

regexp (3)

samsung (3)

tizen (3)

traffic (3)

typedjs (3)

unix (3)

adobe (2)

angry birds (2)

angularjs (2)

astronomy (2)

audio (2)

comic (2)

design (2)

dom (2)

ecommerce (2)

eval (2)

exploring es6 (2)

facebook flow (2)

facets (2)

flash (2)

free (2)

futurama (2)

guide (2)

history (2)

hyena (2)

internet explorer (2)

iteration (2)

journalism (2)

jquery (2)

jsengine (2)

jslib (2)

law (2)

lightning (2)

markdown (2)

math (2)

meego (2)

month (2)

nike (2)

nokia (2)

npm (2)

programming (2)

raffle (2)

repl (2)

servo (2)

sponsor (2)

steve jobs (2)

travel (2)

typescript (2)

usb (2)

winphone (2)

wwdc (2)

airbender (1)

amdefine (1)

aol (1)

app urls (1)

architecture (1)

atscript (1)

basic income (1)

biology (1)

blink (1)

bluetooth (1)

canada (1)

clip (1)

coding (1)

community (1)

cross-platform (1)

deutsch (1)

diaspora (1)

distributed-social-network (1)

dsl (1)

dvd (1)
dzone (1)
emacs (1)
emberjs (1)
energy (1)
esnext news (1)
esprop (1)
example (1)
facetator (1)
feedback (1)
firefly (1)
firefoxos (1)
fritzbox (1)
german (1)
git (1)
guest (1)
guice (1)
h.264 (1)
home entertainment (1)
hosting (1)
htc (1)
ical (1)
jsdom (1)
jsmyth (1)
library (1)
location (1)
marketing (1)
mars (1)
meta-data (1)
middle east (1)
mpaa (1)
msl (1)
mssurface (1)
netflix (1)
nsa (1)
obama (1)
openoffice (1)
opinion (1)
oracle (1)
organizing (1)
philosophy (1)
pixar (1)
pnacl (1)
prism (1)
privacy (1)
proxies (1)
puzzle (1)
raspberry pi (1)
read (1)
rodney (1)
rust (1)
safari (1)
sponsoring (1)
star trek (1)

- static generation (1)
- talk (1)
- technique (1)
- theora (1)
- thunderbird (1)
- typography (1)
- unicode (1)
- v8 (1)
- voice control (1)
- webassembly (1)
- webkit (1)
- webm (1)
- webpack (1)
- yahoo (1)