Domeniul de vizibilitate al variabilelor și închideri

Curs 3

Gabriel Danciu

D.E.C.

12 martie 2024

Prezentare Generală a Cursului

Variabile globale

- În JavaScript, următoarea variabilă ar avea scop global: aGlobalVariable = 'livin la vida global';.
- Orice variabilă declarată în JavaScript fără cuvântul cheie var este creată în scop global, adică în scopul accesibil fiecărei funcții din programul nostru

Exemplu de utilizare a unei variabile globală

```
_.map(_.range(2), function() { return aGlobalVariable });
//=> ["livin la vida global", "livin la vida global"]
```

Variabile lexicale

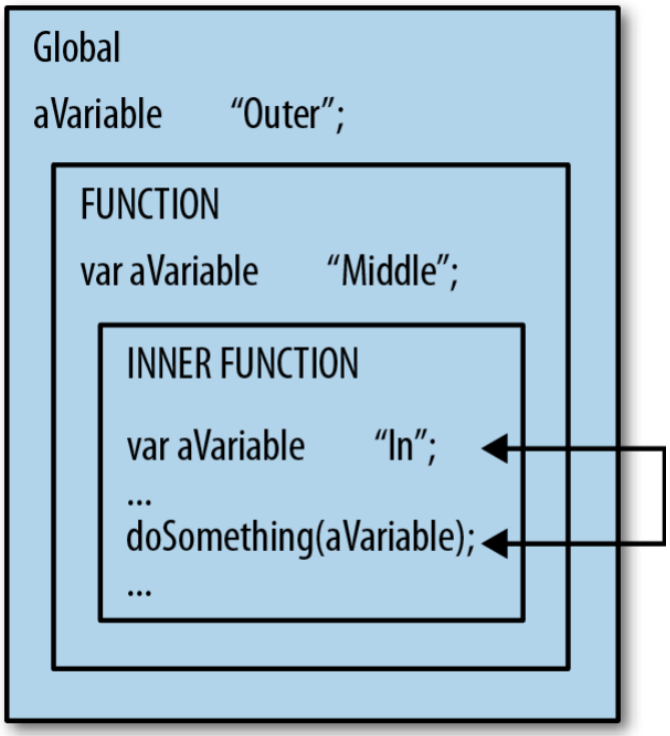
- Scopul lexical se referă la vizibilitatea unei variabile și a valorii sale într-un mod analog reprezentării sale textuale.

Exemplu de utilizare a unei variabile în domeniu lexical

```
aVariable = "Outer";
function afun() {
  var aVariable = "Middle";
  return _.map([1,2,3], function(e) {
    var aVariable = "In";
    return [aVariable, e].join('␣');
  });
}
afun();
//=> ["In 1", "In 2", "In 3"]
```

Variabile lexicale

- Valoarea celei mai interioare variabile, *In*, are prioritate când este utilizată în funcția transmisă către _.map. Scopul lexical lucrează astfel: deoarece atribuirea variabilei *aVariable* la *In* are loc textual aproape de utilizarea sa cea mai interioară/recentă, atunci aceasta va fi valoarea sa în momentul utilizării.



Variabile dinamice

- În scopul dinamic, domeniul de aplicare al unei variabile este determinat de secvența apelurilor de funcții care au condus la punctul curent de execuție. Aceasta diferă de scopul lexical (sau static), unde domeniul de aplicare este determinat la momentul compilării și se bazează pe structura codului.
- Conceptul se bazează pe un tabel global de valori numite, unde fiecare intrare în tabel poate fi potențial o stivă de valori. Această stivă reprezintă diferitele valori pe care o variabilă le-a luat în timpul execuției programului.

Variabile dinamice

Exemplu de utilizare a unei variabile în domeniu dinamic

```
var globals = {};
function makeBindFun(resolver) {
    return function(k, v) {
        var stack = globals[k] || [];
        globals[k] = resolver(stack, v);
        return globals;
    };
}
var stackBinder = makeBindFun(function(stack, v) {
    stack.push(v);
    return stack;
});
var stackUnbinder = makeBindFun(function(stack) {
    stack.pop();
    return stack;
});
```

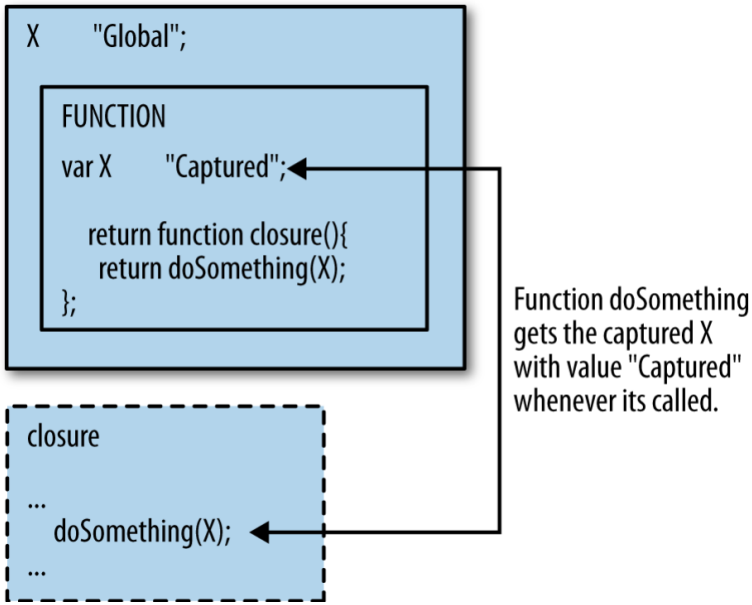
Variabile dinamice

Exemplu de utilizare a unei variabile în domeniu dinamic

```
var dynamicLookup = function(k) {
  var slot = globals[k] || [];
  return _.last(slot);
};
stackBinder('a', 1);
stackBinder('b', 100);
dynamicLookup('a');
globals;
//=> {'a': [1], 'b': [100]}
stackBinder('a', '*');
dynamicLookup('a');
//=> '*'
globals;
//=> {'a': [1, '*'], 'b': [100]}
```

Inchideri

- Pentru început, merită menționat că închiderile merg mână în mână cu funcțiile de primă clasă. Limbajele fără funcții "First Class" pot implementa închideri, dar sunt adesea foarte limitate. Din fericire, JavaScript implementează funcții de primă clasă, astfel încât închiderile sale sunt un mod puternic de a transmite stări ad-hoc încapsulate.
- O închidere este o funcție care „capturează" valorile aproape de "locul" declarației.



Inchideri

Exemplu de utilizare inchideri

```
function whatWasTheLocal() {
  var CAPTURED = "Oh␣kmon";
  return function() {
    return "The␣local␣was:␣" + CAPTURED;
  };
}
var reportLocal = whatWasTheLocal();
reportLocal();
//=> "The local was: Oh kmon"
```

Inchideri

Am vorbit deja despre cum variabilele locale ale funcției au "durata de viață" în corpul funcției în care au fost declarate, dar când o închidere capturează o variabilă, aceasta poate să trăiască -teoretic- pentru o perioadă nedeterminată.

Exemplu de utilizare inchideri

```
function makeAdder(CAPTURED) {
  return function(free) {
    return free + CAPTURED;
  };
}
var add10 = makeAdder(10);
add10(32);
//=> 42
```

Inchideri

Aici se ilustrează puterea închiderilor de a captura și a păstra accesul la variabile și argumente externe funcției închidere, permițând funcției să utilizeze aceste date chiar și după ce execuția contextului în care au fost capturate s-a încheiat

Exemplu de utilizare inchideri

```
function createScaleFunction(FACTOR) {
  return function(v) {
    return _.map(v, function(n) {
      return (n * FACTOR);
    });
  };
}

var scale10 = createScaleFunction(10);
scale10([1,2,3]);
//=> [10, 20, 30]
```

Exercitii

- Folosind conceptul de închidere, creează o funcție createCounter care permite incrementarea unui contor în mod ascuns. Funcția ar trebui să returneze un altă funcție care, atunci când este apelată, va incrementa contorul și va returna valoarea actuală. Nu ar trebui să fie posibil să accesăm sau să modificăm direct contorul din afara funcției createCounter.
- Scrie o funcție limitFunctionCallCount care acceptă ca argumente o funcție (func) și un număr (limit). Funcția returnată ar trebui să permită apelarea funcției *func* de maxim *limit* ori. După atingerea limitării, orice apel ulterior la funcția returnată nu ar trebui să execute *func*, ci doar să returneze undefined.

Exercitii

- Creează o funcție createGreetingFunction care acceptă un string greeting (ex: "Hello"). Funcția returnată ar trebui să accepte un nume (ex: "Alice") și să returneze o salutare completă (ex: "Hello, Alice!"). Utilizează închideri pentru a "captura" salutul specificat în funcția exterioară.
- Scrie o funcție createArrayIterator care acceptă un array ca argument și permite iterarea secvențială prin elementele acestuia folosind o închidere. Funcția returnată ar trebui să returneze următorul element din array la fiecare apel. Dacă toate elementele au fost parcurse, apelurile ulterioare ar trebui să returneze undefined.