Łańcuch prototypów, a konstruktor jako koncepcja pseudo klasycznego programowania

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

O czym będę mówił.

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

Zarys historyczny

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

Scheme

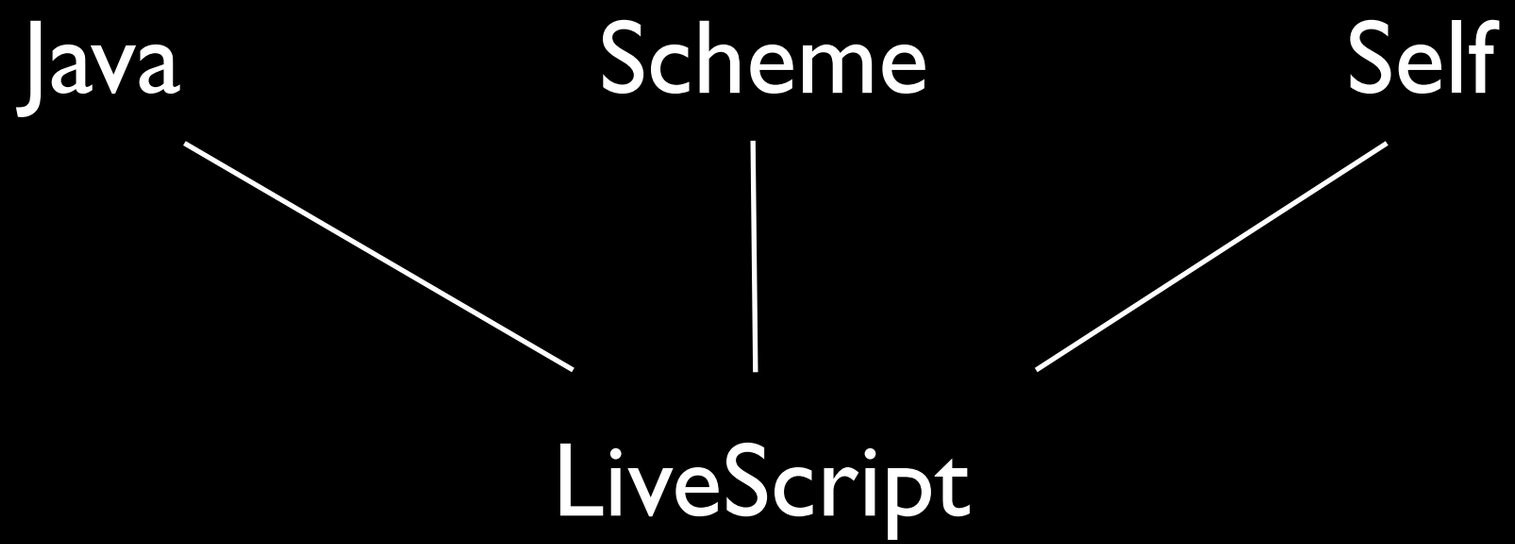
```scheme
(define (list-of-squares n)
  (let loop ((i n) (res '()))
    (if (< i 0)
        res
        (loop (- i 1) (cons (* i i) res)))))

(list-of-squares 9)

===> (0 1 4 9 16 25 36 49 64 81)
```

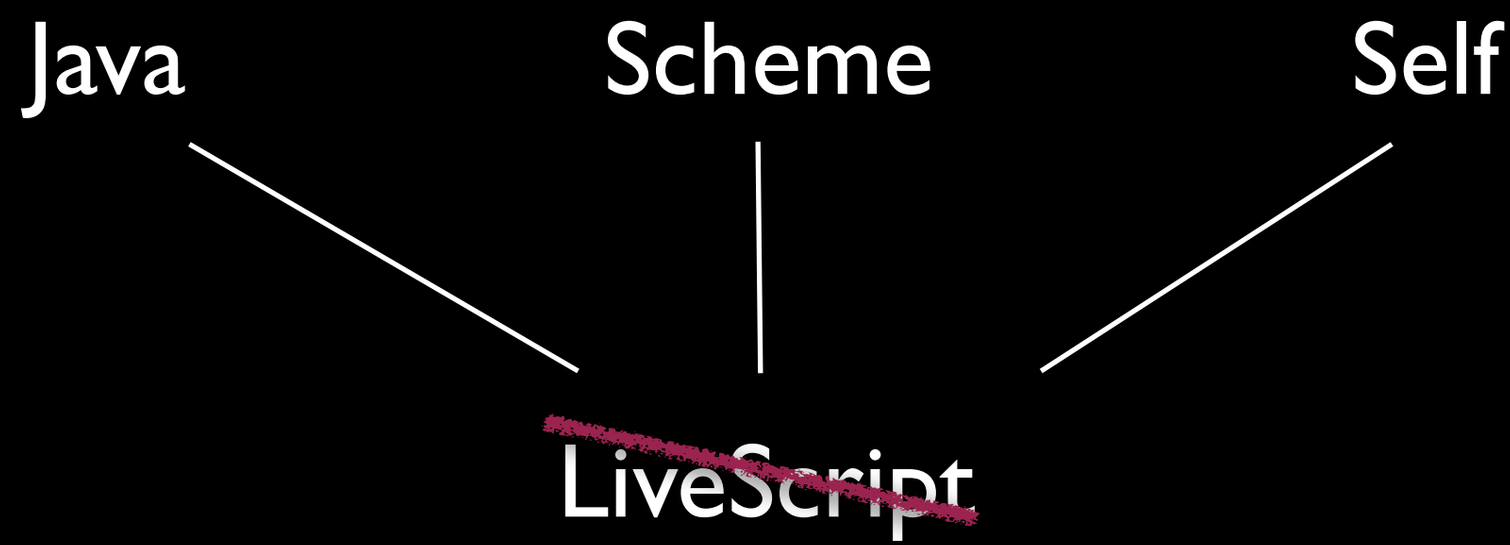
Scheme Self

Prototype-based programming,
styl programowania,
w którym nie występują klasy,
a dziedziczenie odbywa się za pomocą
procesu klonowania istniejących obiektów,
które służą jako prototypy.

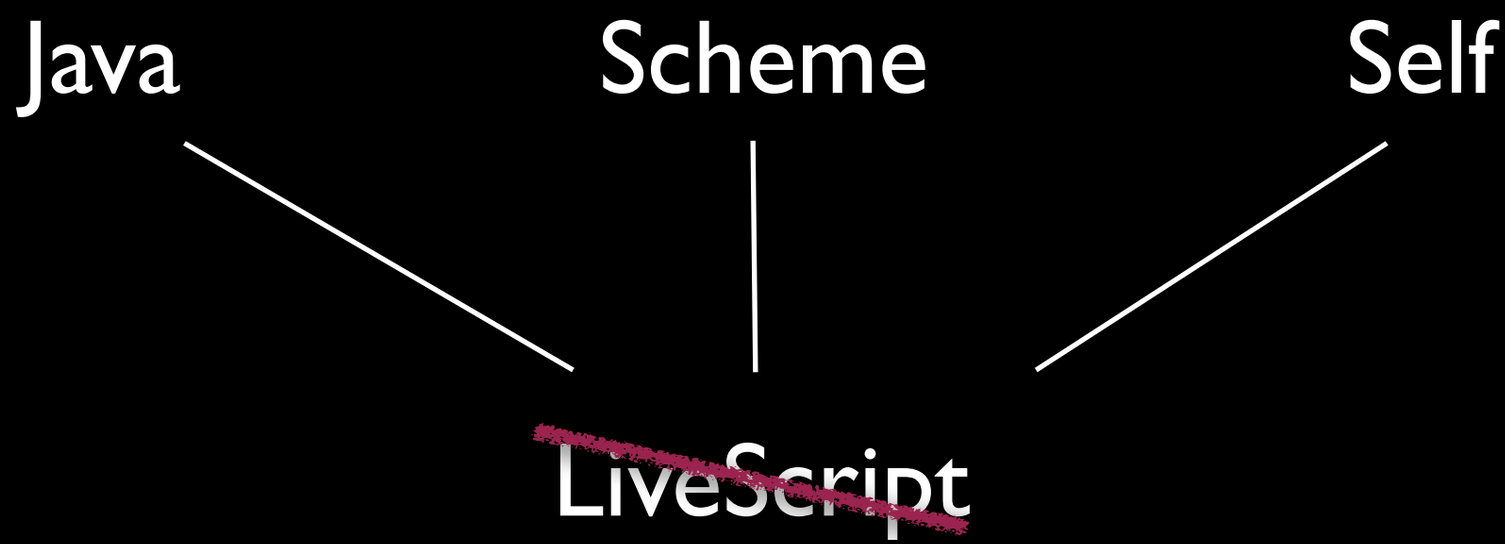
Java Scheme Self

C - like language


Java
Scheme
Self
LiveScript



JavaScript



JavaScript

ECMAScript

JavaScript to nie Java

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

Łańcuch prototypów

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

```
[]
    length: 0
        __proto__: Array
            constructor:  function Array() { [native code] }
            concat:  function concat() { [native code] }
            every:  function every() { [native code] }
            filter:  function filter() { [native code] }
            forEach:  function forEach() { [native code] }
            indexOf:  function indexOf() { [native code] }
            join:  function join() { [native code] }
            length: 0
            map:  function map() { [native code] }
            pop:  function pop() { [native code] }
            push:  function push() { [native code] }
            ...
                __proto__: Object
                    constructor:  function Object() { [native code] }
                    hasOwnProperty:  function hasOwnProperty() { [native code] }
                    isPrototypeOf:  function isPrototypeOf() { [native code] }
                    propertyIsEnumerable:  function propertyIsEnumerable() { [native code] }
                    toLocaleString:  function toLocaleString() { [native code] }
                    toString:  function toString() { [native code] }
                    valueOf:  function valueOf() { [native code] }
```

Pseudo klasyczny konstruktor

```
1  function Collection() {
2      if (!(this instanceof Collection)) {
3          return new Collection();
4      }
5  }
6
7  Collection.prototype = Array.prototype;
8
9  var myCollection = new Collection();
10
11 myCollection.push({
12     name: 'Marcin',
13     surename: 'Kaluzny',
14     email: 'poczta@rainy.pl'
15 });
16
17 // ===> 1
18
19 console.log(myCollection);
20
21 // ===> [Object]
```

Pseudo klasyczny konstruktor

```
myCollection
    length: 0
        __proto__: Array
            constructor:  function Array() { [native code] }
            concat:  function concat() { [native code] }
            every:  function every() { [native code] }
            filter:  function filter() { [native code] }
            forEach:  function forEach() { [native code] }
            indexOf:  function indexOf() { [native code] }
            join:  function join() { [native code] }
            length: 0
            map:  function map() { [native code] }
            pop:  function pop() { [native code] }
            push:  function push() { [native code] }
            ...
                __proto__: Object
                    constructor:  function Object() { [native code] }
                    hasOwnProperty:  function hasOwnProperty() { [native code] }
                    isPrototypeOf:  function isPrototypeOf() { [native code] }
                    propertyIsEnumerable:  function propertyIsEnumerable() { [native code] }
                    toLocaleString:  function toLocaleString() { [native code] }
                    toString:  function toString() { [native code] }
                    valueOf:  function valueOf() { [native code] }
```

Pseudo klasyczny konstruktor

```
(function () {
    var myCollection = new Collection(),
        data = [{
            name: 'Marcin',
            surename: 'Kaluzny',
            email: 'poczta@rainy.pl'
        }, {
            name: 'Waldek',
            surename: 'Kowalski',
            email: 'kowalski@poczta.pl'
        }];

    myCollection.push.apply(myCollection, data);

    Array.prototype.contains = function (item) {
        return this.indexOf(item) > -1;
    };

    console.log(myCollection.contains(data[0]));
}());
```

Pseudo klasyczny konstruktor

```
1  function Collection() {
2      if (!(this instanceof Collection)) {
3          return new Collection();
4      }
5
6      this.contains = function (item) {
7          return this.indexOf(item) > -1;
8      };
9  }
10
11 Collection.prototype = Array.prototype;
12
13 var myCollection = new Collection();
14
15 myCollection.push({
16     name: 'Marcin',
17     surename: 'Kaluzny',
18     email: 'poczta@rainy.pl'
19 });
20
```

```
myCollection
    contains: function (item) {}
    length: 0
        __proto__: Array
            constructor:  function Array() { [native code] }
            concat:  function concat() { [native code] }
            every:  function every() { [native code] }
            filter:  function filter() { [native code] }
            ...
    ...
        __proto__: Object
            constructor:  function Object() { [native code] }
            hasOwnProperty:  function hasOwnProperty() { [native code] }
            isPrototypeOf:  function isPrototypeOf() { [native code] }
            propertyIsEnumerable:  function propertyIsEnumerable() { [native code] }
            toLocaleString:  function toLocaleString() { [native code] }
            ...
```

Pseudo klasyczny konstruktor

```
1  function Handlers() {
2      if (!(this instanceof Handlers)) {
3          return new Handlers();
4      }
5
6      Collection.apply([].slice.call(arguments));
7  }
8
9  Handlers.prototype = new Collection();
10
11 (function () {
12     var handlers = new Handlers(),
13         click = function () {
14             // function body
15         };
16
17     handlers.push(click);
18
19     console.log(handlers.contains(click));
20 }());
```

Dziedziczenie prototypowe

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

Object.create()

meet.js - Gdańsk 27.06.2013

Marcin Kałużny

Dziedziczenie prototypowe

```
1  var collection = Object.create(Array.prototype);
2
3  Object.defineProperty(collection, 'contains', {
4      value: function (item) {
5          return this.indexOf(item) > -1;
6      }
7  });
```

- **value** – wartość własności
- **get** – funkcja-getter dla tej własności, lub undefined jeśli wartość ma być odczytana bezpośrednio z value
- **set** – funkcja-setter dla tej własności, lub undefined jeśli wartość ma być zapisana bezpośrednio do value
- **writable** – wartość logiczna określa, czy można modyfikować daną własność;
- **enumerable** – wartość logiczna określa, czy własność pojawi się przy wyliczeniach (np. pętla for...in, operator in itp.)
- **configurable** – wartość logiczna określa, czy można modyfikować deskryptor własności, oraz czy można usunąć własność operatorem delete

```
var handlers = Object.create(collection, {
    on: {
        value: function (type, callback) {

        }
    },
    off: {
        value: function (type, callback) {

        }
    },
    trigger: {
        value: function (type, data) {

        }
    }
});
```

Dziękuję za uwagę

Pytania? Walcie śmiało!

Jak nie dzisiaj to na
e-mail: **poczta@rainy.pl**