Ecma/TC39/2010/047

Iterable proxies

Dave Herman

July 28, 2010

Object iteration

x ranges over names of enumerable properties of obj

```
for (let x in obj) {
    ...
}
```

Pluggable iteration

```
for (let x in values(obj)) {
    ...
}
```

Pluggable iteration, ctd.

```
for (let [x, y] in properties(obj)) {
   ...
}
```

Pluggable iteration, ctd.

```
for (let { artist, title, year } in queryCDDB()) {
    ...
}
```

Iterable proxies

- New derived trap: iterate : **function**() → iterator
- An *iterator* is any object of shape:

  { next : **function**() → any }
- Standard **for-in** behavior: *enumeration*
- New **for-in** behavior uses iterate trap, if present

Iterable proxies, ctd.

```
function fibonacci() {
    var [prev, curr] = [0, 1];
    return {
        next: function() {
            [prev, curr] = [curr, prev + curr];
            return curr;
        }
    }
}
```

Iterable proxies, ctd.

```
var p = Proxy.create({ ..., iterate: fibonacci, ... });
for (let n in p) {
    print(n);
    if (n > 1000) break;
}
```

StopIteration

Iterators signal end-of-iteration by throwing an object whose

[[Class]] = “StopIteration”

```
function fibonacciPrefix() {
    return {
        next: function() {
            ... if (curr > 100) throw StopIteration; ...
        }
    }
}
```

Conveniences

```
for (let n in Iterator.create(fibonacci)) {
    ...
}
```

Conveniences, ctd.

- Iterator.create: *iterable null-proxy*

  function(**function**() → iterator) → iterator

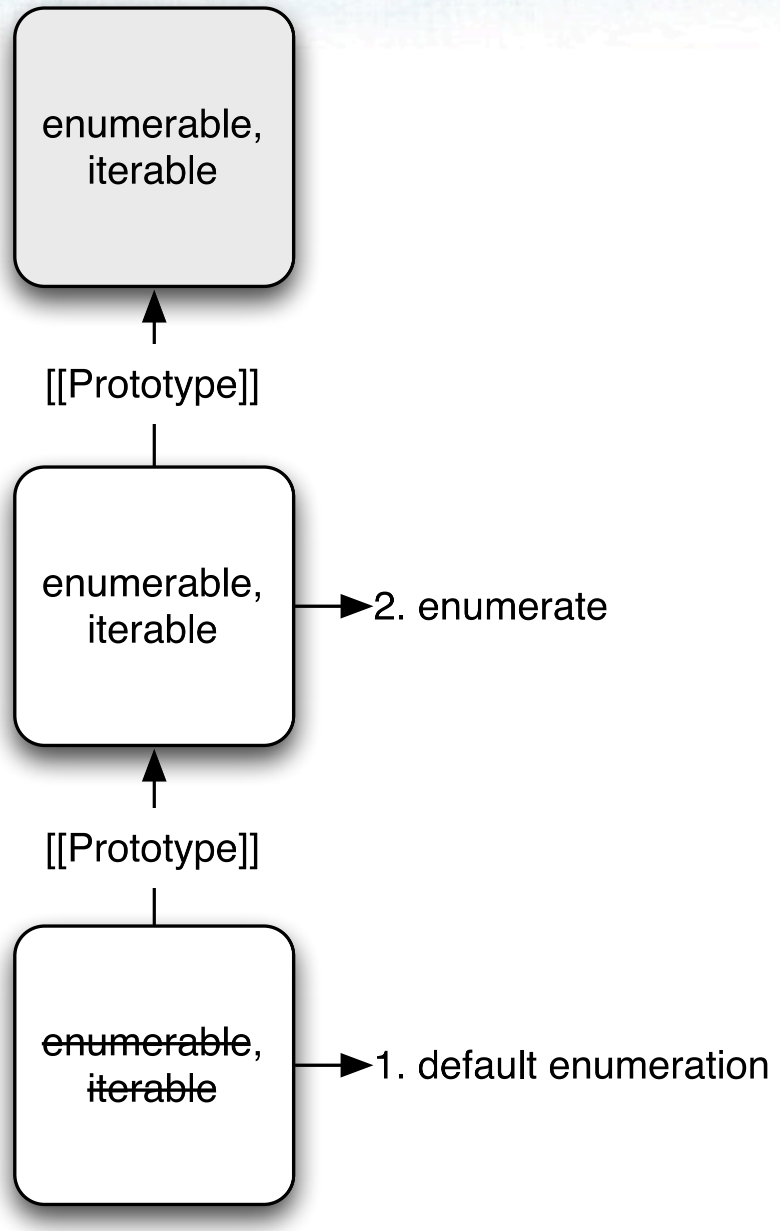
- Iterator.for: *iterable forwarding-proxy*

  function(Object, **function**() → iterator) → iterator

Enumeration

- Derived iterate trap defaults to enumerate trap.
- Only **for-in** looks for iterate trap.
- Within prototype chain enumeration, iterate trap is ignored.

Enumeration, ctd.



Iterable pseudo-classes

```
function MyCollection() {
    ...
    var self = this;
    return Iterator.for(self, function() {
        return self.makeIterator();
    }
}
MyCollection.prototype = {
    ...
    makeIterator: function() { ... }
}
```

replace this with an iterable forwarding proxy to this

Scoping for-in loops: let

separate binding per iteration

```
var a = [];
for (let x in [0,1,2,3,4]) {
    a.push(function() { return x });
}
print(a[0]()) // 0
```

Scoping for-in loops: const

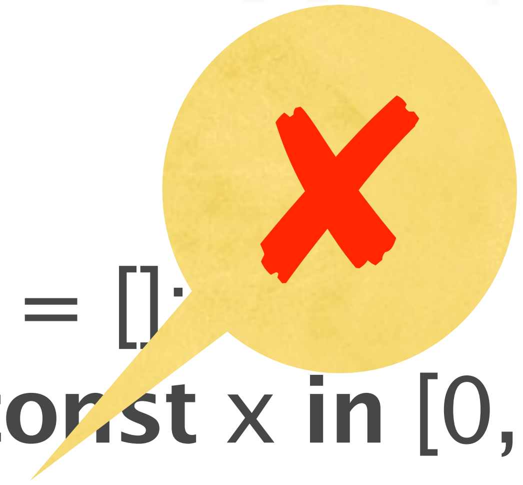
```
var a = [];
for (const x in [0,1,2,3,4]) {
    a.push(function() { return x });
}
print(a[0]())
```

Scoping for-in loops: const, ctd.

```
var a = [];
for (const x in [0,1,2,3,4]) {
    x++;
    a.push(function() { return x });
}
print(a[0]())
```

Scoping for-in loops: const, ctd.

```
var a = [];
for (const x in [0,1,2,3,4]) {
    x++;
    a.push(function() { return x });
}
print(a[0]())
```



Summary

- Iterable proxies via iterator objects and derived trap
- Default iteration behavior is enumeration
- Small convenience library for common iterator idioms
- Cleanup of **for-in** scoping