Harmony Proxies: loose ends

Tom Van Cutsem
Mark S. Miller
var p = Proxy.create({
    get: function(receiver, name) { ... },
    invoke: function(receiver, name, args) { ... },
    ...
});

p.x; // get(p,'x')
p.m(a); // invoke(p, 'm', [a])
invoke trap

- **Pro:**
  - invoke trap has access to arguments of invoked property
  - can distinguish o.m.call(o) from o.m()
  - faster than 'get' + 'call'

- **Con:**
  - Breaks argument evaluation order
    - Maintaining order is possible, at some cost
    - "var f = o.m; f()" not slower than "o.m()" for proxies
  - providing both 'get' and 'invoke' traps:
    - breaks invariant that o.m.call(o) <=> o.m()
    - more complex API: traps should evolve in sync
  - if c delegates to a proxy, and does not define "m", does c.m() trigger 'invoke' or 'get' trap of the proxy?
function forward(target) {
    return Proxy.create({
        get: function(rcvr, name) {
            return function(...args) {
                return target[name](...args);
            }
        }
    });
}
Standard default handlers

Since all handler traps are mandatory, and since the API is quite large, it makes sense to provide a couple of 'default' handlers that can be 'specialized'

Proxy.createHandler(target); // returns forwarding handler

Proxy.sinkHandler; // no-op handler
Array proxies

In TM implementation, Proxy.create accepts third [[Class]] arg:
Proxy.create(handler, proto, className)

Enables transparent Array proxies:

var a = Proxy.create({
    get: function(rcvr, name) {...},
    ...
}, Array.prototype, "Array");

a[15]; // should call handler.get(a, "15")
Proxy.isTrapping(obj) -> boolean
returns whether obj is a trapping proxy

This is the only method that breaks transparent virtualization of objects.

Should we remove it from the API to achieve fully transparent virtualization?
Note: ephemeron tables enable user-land implementation of Proxy.isTrapping for proxies that wish to be non-transparent
Recall:

```javascript
var proxy = Proxy.create({
    ...,
    enumerate: function() { return ['a','b','c']; } 
});

for (var name in proxy) {
    // enumerates 'a', 'b', 'c'
}
```
Proxies and Enumeration

- array returned by `enumerate()` is a snapshot
- for-in loop should enumerate properties in the order specified by the snapshot
- for-in loop should enumerate only props in the snapshot:
  - properties added to the proxy later are not enumerated
  - deleted properties should be skipped
  - if proxy is fixed during enumeration, continue enumeration based on snapshot
- Mutating the snapshot while enumerating: *as if* for-in loop enumerated snapshot as:
  
  ```javascript
  for (var i = 0; i < snapshot.length; i++) {...}
  for (var i = 0, len = snapshot.length; i < len; i++) {...}
  ```
Updated API

Fundamental traps

- `Object.getOwnProperty(proxy)`: function(name) -> `pd` | undefined
- `Object.getProperty(proxy)`: function(name) -> `pd` | undefined
- `Object.defineProperty(proxy, name, pd)`: function(name, pd) -> undefined
- `delete proxy.name`: function(name) -> boolean
- `Object.getOwnPropertyNames(proxy)`: function() -> `[ string ]`
- `for (name in proxy)`: enumerate: function() -> `[string]`
- `Object.{freeze|seal|preventExtensions}(proxy)`: function() -> `propertyMap` | undefined

Derived traps (less allocations)

- `name in proxy`: has: function(name) -> boolean
- `{}.hasOwnProperty.call(proxy, name)`: hasOwn: function(name) -> boolean
- `receiver.name`: get: function(receiver, name) -> any
- `receiver.name(...args)`: set: function(receiver, name, val) -> boolean
- `receiver.name = val`: enumerateOwn: function() -> `[string]`
- `Object.keys(proxy)`: fix: function() -> `propertyMap` | undefined