Notification Proxies: update TC39 May 2013

Tom Van Cutsem
with help from Mark S. Miller, based on idea by E. Dean Tribble
Why Notification Proxies?

• A simpler alternative to direct proxies

• Direct Proxies: require **runtime assertions** on trap return values to check non-configurable/non-extensible invariants

  • Adds **runtime overhead**, even when proxy handlers “behave”

  • Adds **integrity hazard**: if we forget an assertion, invariant can be violated

  • Adds **spec complexity**: invariants are different for each operation

    • Especially complex for ops that return collections, 
      e.g. `Object.getOwnPropertyNames`
Notification Proxies

- Key idea: traps are just notification callbacks, they don’t get to directly return the *result* of the intercepted operation

- Intercepted ops are always forwarded to the target after invoking the trap
  - No need to manually “forward” the operation

- Trap can optionally return a function to be invoked as a *post-trap*
  - Post-trap can observe the result, but cannot change it
Example

• With direct proxies:

```javascript
var target = {};

var handler = {
  get: function(target, name, receiver) {
    console.log("getting: " + name);
    var result = Reflect.get(target, name, receiver);
    console.log("got: " + result);
    return result;
  }
};

var proxy = new Proxy(target, handler);
```
Example

- With notification proxies:

```javascript
var target = {};

var handler = {
  get: function(target, name, receiver) {
    console.log("getting: " + name);
    return function(target, name, receiver, result) {
      console.log("got: " + result);
    }
  }
};

var proxy = new Proxy(target, handler);
```
Direct Proxies

1. Operation intercepted on proxy
   proxy.x
Direct Proxies

1. Operation intercepted on proxy

2. Does handler define trap?

\[ \text{trap} = \text{handler}[\text{“get”}] \]
Direct Proxies

2. Does handler define trap?
   2.a If not, forward Operation to target and return result

return target.x;
Direct Proxies

2. Does handler define trap?
3. If yes, call trap and get result

result = trap.call(handler, target, "x", proxy);
Direct Proxies

4. If target has an invariant for operation, **check whether result satisfies invariant**

3. If yes, call trap and get result

```javascript
var desc = Object.getOwnPropertyDescriptor(target, "x");
if (desc && !desc.configurable && !desc.writable) {
  assert [[SameValue]](result, desc.value);
}
```
Direct Proxies

4. If target has an invariant for operation, check whether result satisfies invariant

5.a If yes, return result
5.b If not, throw TypeError
Notification Proxies

1. Operation intercepted on proxy
   proxy.x
Notification Proxies

1. Operation intercepted on proxy

2. Does handler define a pre-trap?

\[
\text{trap} = \text{handler["onGet"]}
\]
Notification Proxies

2. Does handler define a pre-trap?

2.a If not, forward Operation to target and return result

return target.x;
Notification Proxies

2. Does handler define trap?
3. If yes, call “pre-trap”, may return a “post-trap”

```
postTrap = trap.call(handler, target, "x", proxy);
```
Notification Proxies

result = target.x;
Notification Proxies

4. Forward operation to target, storing result
5. If post-trap is a function, call it and ignore its result

postTrap.call(handler, target, "x", proxy, result)
Notification Proxies

6. return result;

5. If post-trap is a function, call it and ignore its result
Notification Proxies

- Handler API same as for direct proxies

- “on” prefix to suggest callback-nature of traps

```javascript
onGetOwnPropertyDescriptor: function(target, name)
onGetOwnPropertyNames: function(target)
onGetPrototypeOf: function(target)
onDefineProperty: function(target, name, desc)
onDeleteProperty: function(target, name)
onFreeze: function(target)
onSeal: function(target)
onPreventExtensions: function(target)
onIsFrozen: function(target)
onIsSealed: function(target)
onIsExtensible: function(target)
onHas: function(target, name)
onHasOwn: function(target, name)
onGet: function(target, name, receiver)
onSet: function(target, name, val, receiver)
onEnumerate: function(target)
onKeys: function(target)
onApply: function(target, thisArg, args)
onConstruct: function(target, args)
```
Prototype: reflect.js library

github.com/tvcutsem/harmony-reflect

- Implements Direct Proxies on top of original Harmony Proxies
- Monkey-patches primordials to recognize emulated direct proxies

```
<script src="reflect.js">
```
Prototype: reflect.js library

- Now also supports Notification Proxies

- Handler logic:
  - Direct Proxies: 850 LoC
  - Notification Proxies: 312 LoC

```
<script src="notify-reflect.js">
```

notification proxies

platform w/ original proxies

original proxy

notification proxy handler (user-defined)

original proxy handler (reflect.js)

target
Membranes

- Goal: isolate two object graphs
- Litmus test for expressiveness of proxies
- Must be transparent: maintain invariants on both sides of the membrane

```javascript
var wetB = {};
var wetA = { x: wetB };
var dryA = wet2dry(wetA);
var dryB = dryA.x;
```
Membranes with direct proxies: try 1

- Works fine as long as `wetTarget` doesn’t have any invariants

```javascript
var wetB = {};  
var wetA = { x: wetB };  

var dryA = wet2dry(wetA);  
var dryB = dryA.x;

function wet2dry(wetTarget) {
  ...
  var dryProxy = new Proxy(wetTarget, {
    ...
      get: function(wetTarget, name, dryThis) {
        return wet2dry(Reflect.get(wetTarget, name, dry2wet(dryThis)));
      }
  });
  ...
}
```
Membranes with direct proxies: try 1

• Now assume \texttt{wetTarget} is frozen

• Because \texttt{wetA.x} is non-configurable non-writable, and \texttt{wetA.x === wetB}, the proxy asserts that \texttt{dryA.x === wetB}

\begin{verbatim}
var wetB = {};
var wetA = Object.freeze({ x: wetB });
var dryA = wet2dry(wetA);
dryA.x // TypeError: cannot report inconsistent value for non-writable, non-configurable property ‘x’
\end{verbatim}
Membranes with direct proxies: try 2

• Use a “shadow” target: a dummy target object to store wrapped properties

```javascript
var wetB = {}; 
var wetA = Object.freeze({ x: wetB }); 
var dryA = wet2dry(wetA); 
var dryB = dryA.x; // ok: shadowTarget.x === dryB
```
Membranes with direct proxies: try 2

- In the case of membranes: shadow and real target are on opposite sides of the membrane

```javascript
var wetB = {};
var wetA = Object.freeze({ x: wetB });
var dryA = wet2dry(wetA);
var dryB = dryA.x; // ok: shadowTarget.x === dryB

function wet2dry(wetTarget) {
    ...
    var dryShadow = {};
    var dryProxy = new Proxy(dryShadow, {
        ...
        get: function(dryShadow, name, dryThis) {
            // copy wet2dry(wetTarget[name]) to dryShadow
            return Reflect.get(dryShadow, name, dryThis);
        }
    });
    ...
}
```
Membranes with direct proxies

- Optimization: if no invariant is at stake, just ignore shadow target

```javascript
function wet2dry(wetTarget) {
  ...
  var dryShadow = {};
  var dryProxy = new Proxy(dryShadow, {
    ...
    get: function(dryShadow, name, dryThis) {
      // no-invariant case: fast-path, no copying
      if (isWritableOrConfigurable(wetTarget, name)) {
        return wet2dry(Reflect.get(wetTarget, name, dry2wet(dryThis)));
      }

      // invariant case: need to copy to shadow
      // copy wet2dry(wetTarget[name]) to dryShadow
      return Reflect.get(dryShadow, name, dryThis);
    }
  });
  ...
}
```
Membranes with notification proxies

- Must also use shadow target technique

- Naive implementation: always copy the accessed property from real target to shadow target in the pre-trap.

  - When the notification proxy then forwards the operation, it will find the right value on the shadow.

```javascript
function wet2dry(wetTarget) {
  ...
  var dryProxy = new Proxy(dryShadow, {
    ...
    onGet: function(dryShadow, name, dryThis) {
      // copy wet2dry(wetTarget[name]) to dryShadow
      return undefined;
    }
  });
  ...
}
```
Membranes: conclusion

- Both Direct Proxies and Notification Proxies can express membranes, with roughly the same implementation strategy:
  - Membranes with Direct proxies: 470 LoC
  - Membranes with Notification proxies: 402 LoC
- Direct Proxies can optimize for objects without invariants: no copying to shadow target needed
- Notification Proxies: optimizations are possible, but must copy each accessed property to the shadow target at least once
Micro-benchmarks

• Simple micro-benchmarks to get some indication of relative performance difference

• Setup: traverse large data structure wrapped in a membrane from outside the membrane

• Tested both frozen and non-frozen data structure (invariants vs. no invariants)

• Apples-to-apples: both Direct and Notification proxies self-hosted in JS

  • But: only look at relative perf. The absolute numbers are not interesting, built-in impls will be orders-of-magnitude faster.
Benchmark #1: Array Loop

- Creates a wrapper per entry, one property access per wrapper

```javascript
var dryArray = wet2dry(createArray(1000));
for (var i=0; i < 1000; i++){
    sum += dryArray[i].nth;
}
```
Benchmark #2: Tree Walk

- Creates a wrapper per node, 5x property access + 1x method invocation per node

```javascript
// binary tree of depth 10 = 1023 nodes
wetTree = binaryTree(10)

dryTree

function traverse(dryTree) {
  var l = dryTree.left ? traverse(dryTree.left) : 0;
  var r = dryTree.right ? traverse(dryTree.right) : 0;
  return dryTree.depth() + l + r;
}
```
Array loop (Firefox 20)

-51.04%  -2.23%

(box plot of 16 runs in same browser session, each individual run = average of 10 traversals)
Array loop (Chrome 26)

non-frozen

-5.85%

frozen

+22.24%
Tree walk (Firefox 20)

non-frozen

-6.94%

frozen

+8.84%
Tree walk (Chrome 26)

non-frozen

frozen

-2.41%

+33.38%
Micro-benchmarks

• As always with micro-benchmarks, take these numbers with a grain of salt
  • Ample room for optimization in the membrane code
  • Built-in Proxy/WeakMap implementations still young
• Inconclusive. My gut feeling: either API can be made efficient.

<table>
<thead>
<tr>
<th></th>
<th>Firefox 20</th>
<th>Chrome 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Loop</td>
<td>-51.04%</td>
<td>-5.85%</td>
</tr>
<tr>
<td>Frozen Array Loop</td>
<td>-2.23%</td>
<td>22.24%</td>
</tr>
<tr>
<td>Tree Walk</td>
<td>-6.94%</td>
<td>2.41%</td>
</tr>
<tr>
<td>Frozen Tree Walk</td>
<td>8.84%</td>
<td>33.38%</td>
</tr>
</tbody>
</table>

Table: relative perf gain/loss of notification proxies compared to direct proxies
Conclusion

• Notification Proxies:

  • Pro: simpler design, easier to spec

  • Con: “virtual objects” must always copy each accessed property at least once (direct proxies must only copy for objects with invariants)

• Perf: let’s not draw any conclusions just yet
References

• Self-hosted implementation of direct proxies:
  https://github.com/tvcutsem/harmony-reflect/blob/master/reflect.js

• Self-hosted implementation of notification proxies:
  https://github.com/tvcutsem/harmony-reflect/blob/master/notification/notify-reflect.js

• Membranes with direct proxies:
  https://github.com/tvcutsem/harmony-reflect/blob/master/examples/membrane.js

• Membranes with notification proxies:

• Benchmarks: https://github.com/tvcutsem/harmony-reflect/tree/master/test/membranes

• Paper with details on direct proxies and the shadow target technique: