Ecma/TC39/2011/018

Harmony Proxies: strawmen

Tom Van Cutsem

Custom prototypes for function proxies (dherman)

- Callable subtypes of Function

- Proposal: allow function proxies to inherit from objects that inherit from Function.prototype:

```
Proxy.createFunction(handler, call, construct, proto)
// throws if proto !== Function.prototype &&
//             !(proto instanceof Function)
```

- Optional argument, defaults to Function.prototype

http://wiki.ecmascript.org/doku.php?id=strawman:function_proxy_prototype

Handler access to proxies

- Proxy handler currently has no access to the proxy instance it is serving
    - Shared handler may want to store per-proxy state in a WeakMap
    - Get at the proxy's prototype (needed to emulate inheritance chain)
        - Leads to less fundamental traps (see later)
- Do we want handler to access the proxy?

http://wiki.ecmascript.org/doku.php?id=strawman:handler_access_to_proxy

Handler access to proxies: API choices

- Add proxy as a *last* argument to all traps:
  - optional, inconsistent position w.r.t. trapped code:

```
Object.getOwnPropertyDescriptor(proxy, name)
getOwnPropertyDescriptor: function(name, proxy) {...}
```

- Add proxy as a *first* argument to all traps:
  - consistent w.r.t trapped code except for “has”, not always optional
  - consistent with existing “get” and “set” traps

Handler access to proxies: API proposal

(changes marked in red)

```
Object.getOwnPropertyDescriptor(proxy,name)
   Object.getPropertyDescriptor(proxy,name)
       Object.defineProperty(proxy,name,pd)
          Object.getOwnPropertyNames(proxy)
             Object.getPropertyNames(proxy)
                          delete proxy.name
                 for (name in proxy) { ... }
Object.{freeze|seal|preventExtensions}(proxy)
```

Fundamental traps

```
handler.getOwnPropertyDescriptor(proxy,name)
handler.getPropertyDescriptor(proxy,name)
handler.defineProperty(proxy,name,pd)
handler.getOwnPropertyNames(proxy)
handler.getPropertyNames(proxy)
handler.delete(proxy,name)
handler.enumerate(proxy)
handler.fix(proxy)
```

Derived traps

```
                  name in proxy
({}).hasOwnProperty.call(proxy, name)
                  receiver.name
                proxy.name = val
              Object.keys(proxy)
```

```
handler.has(proxy,name)
handler.hasOwn(proxy,name)
handler.get(receiver,name)
handler.set(proxy,name,val)
handler.keys(proxy)
```

More derived traps

- getPropertyDescriptor and getPropertyNames currently fundamental
- Could become derived if given access to the proxy object (or its prototype)
- Would make all “prototype-climbing” traps derived

```
getPropertyDescriptor: function(proxy, name) {
  var pd = Object.getOwnPropertyDescriptor(proxy, name); // calls getOwnPropertyDescriptor trap
  var proto = Object.getPrototypeOf(proxy);
  while (pd === undefined && proto !== null) {
    pd = Object.getOwnPropertyDescriptor(proto, name);
    proto = Object.getPrototypeOf(proto);
  }
  return pd;
}

getPropertyNames: function(proxy, name) {
  var props = Object.getOwnPropertyNames(proxy); // calls getOwnPropertyNames trap
  var proto = Object.getPrototypeOf(proxy);
  while (proto !== null) {
    props = props.concat(Object.getOwnPropertyNames(proto));
    proto = Object.getPrototypeOf(proto);
  }
  // remove duplicate property names from props (not shown)
  return props;
}
```


http://wiki.ecmascript.org/doku.php?id=strawman:proxy_derived_traps

Forwarding Handler, Derived Traps

- Default forwarding handler forwards all operations to a target object
- Two possible semantics for its derived traps:
  - “Forwarding semantics”: e.g.

```
has: function(name) { return name in target; }
```

  - “Fallback semantics”: e.g.

```
has: function(name) { return !!this.getPropertyDescriptor(name); }
```


http://wiki.ecmascript.org/doku.php?id=strawman:derived_traps_forwarding_handler

Forwarding Handler, Derived Traps

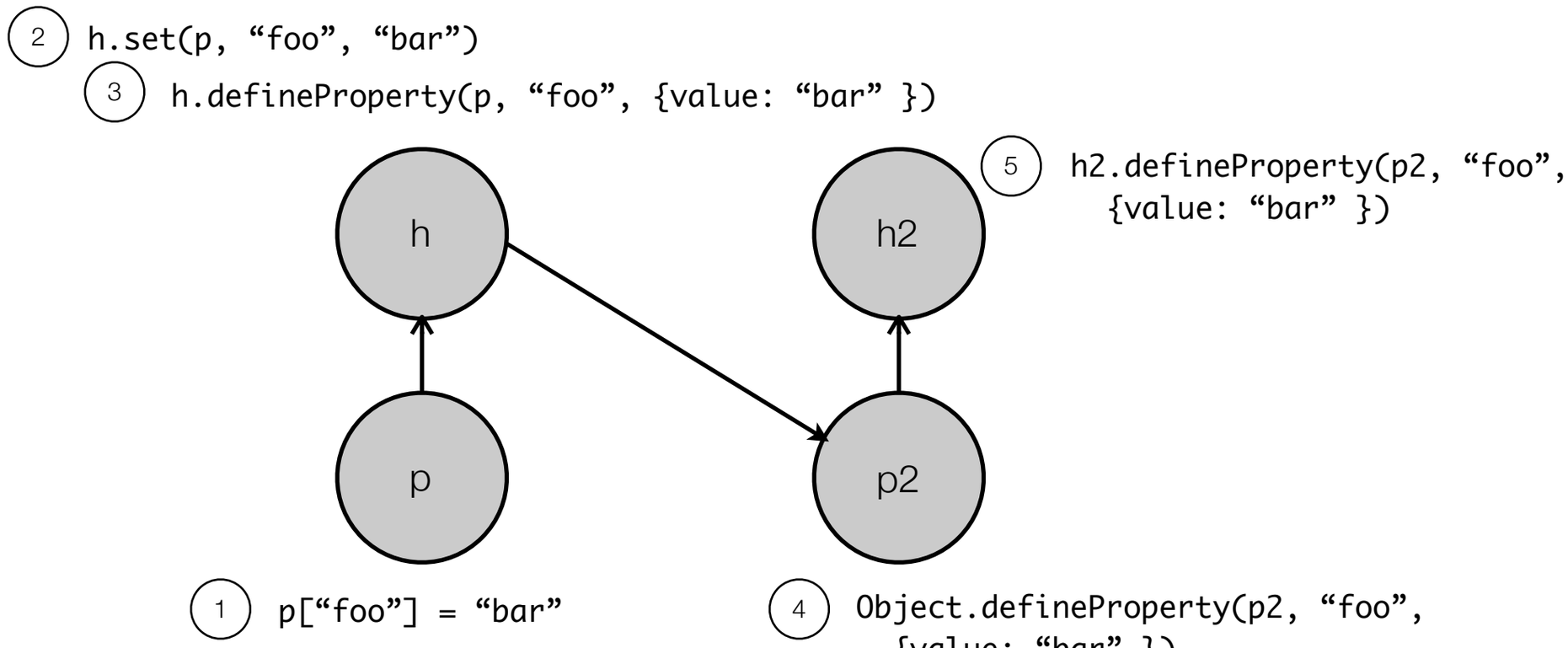
- Forwarding semantics requires fundamental & derived traps to be overridden “in sync”:

```
var target = {...};
var h = new Proxy.Handler(target);
h.defineProperty = function(name, pd) {...}; // override a fundamental trap
var p = Proxy.create(h);

p["foo"] = "bar"; // triggers default forwarding 'set' trap. Sets "foo" on target.
// programmer may have expected this to trigger overridden fundamental trap instead

delete Object.getPrototypeOf(h).set;
p["foo"] = "bar"; // triggers overridden defineProperty trap
```

Forwarding Handler, Derived Traps

- Derived semantics is surprising w.r.t. chains of proxies:

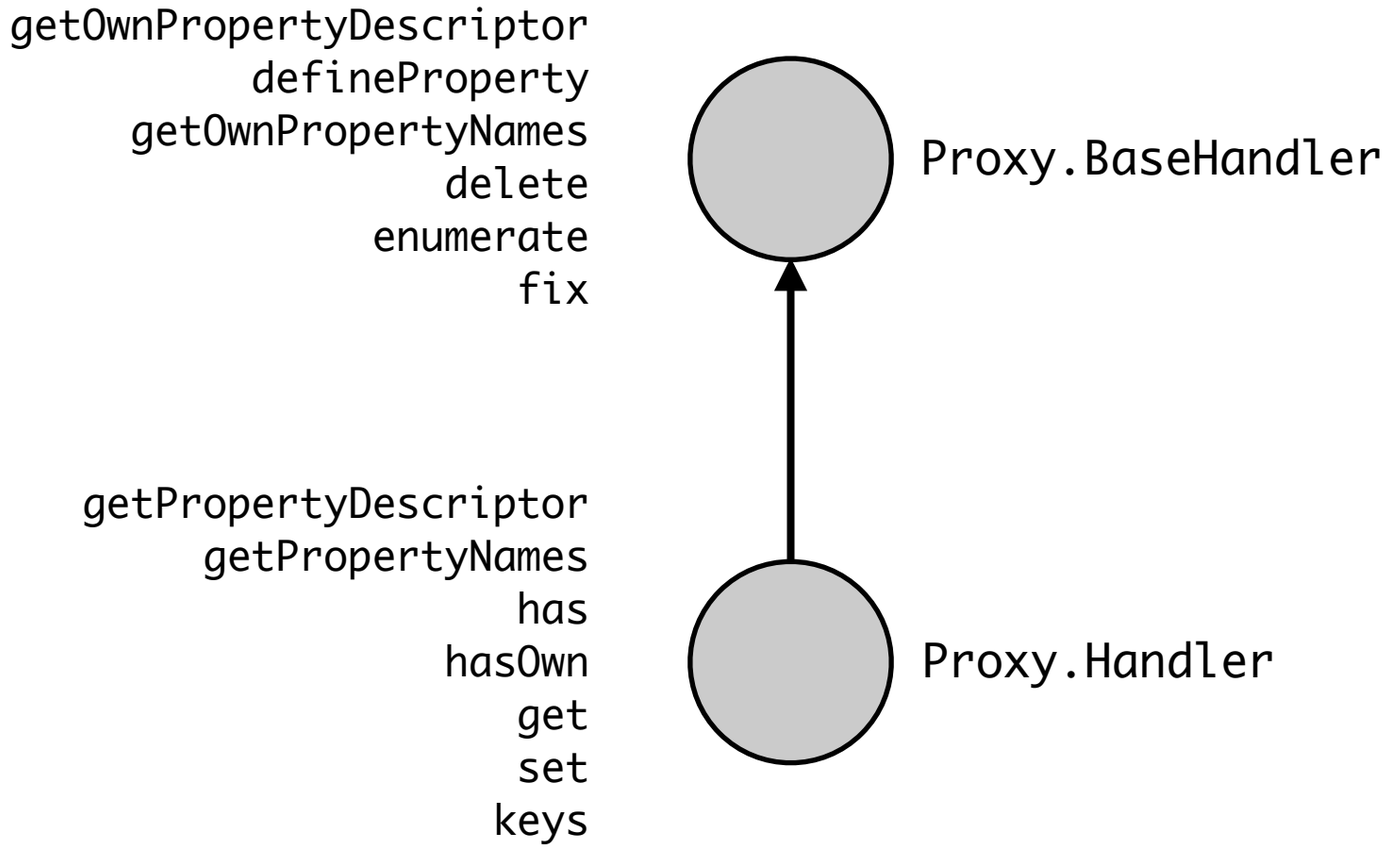
```
var p2 = Proxy.create(h2);
var h = new Proxy.Handler(p2); // p forwards to p2
var p = Proxy.create(h);

p["foo"] = "bar";
```



Forwarding Handler, Derived Traps: proposal

- Proxy.BaseHandler defines only fundamental traps (derived traps have fallback semantics)

- Proxy.Handler inherits from BaseHandler, adds forwarding derived traps (derived traps have forwarding semantics).



Derived set trap

- ES5 [[Put]] triggers both getOwnPropertyDescriptor and getPropertyDescriptor traps on own accessor properties
- Without proxies: unobservable, with proxies: observable and unnecessary

```
set: function(receiver, name, val) {
   var desc = this.getOwnPropertyDescriptor(name);
   if (desc) {
     if ('writable' in desc) {
       if (desc.writable) {
         desc.value = val;
         this.defineProperty(name, desc);
         return true;
       } else {
         return false;
       }
     } else { // accessor
       this.getPropertyDescriptor(name);
       if (desc.set) {
         desc.set.call(receiver, val);
         return true;
       } else {
         return false;
       }
     }
   }
   ...
```


http://wiki.ecmascript.org/doku.php?id=strawman:proxy_set_trap