[[strawman:
inherited_explicit_soft_fields]]

ES
Wiki

Show pagesource

Old revisions

Recent changes

Search

Trace: » es3.1 » start » iterators_and_generators » const_functions » inherited_explicit_soft_fields

Explicit Inherited Soft Fields

The following derived abstraction combines the explicitness of explicit soft own fields with the visibility across inheritance chains of inherited soft fields. Below is an executable specification as a wrapper around weak_maps. This strawman page suggests standardizing this derived abstraction because a primitive implementation is likely to be more efficient that the code below.

As with our previous "`EphemeronTable`", the name "`ExplicitSoftField`" is only a placeholder until someone suggests an acceptable name.

```
  const ExplicitSoftField() {
    const et = WeakMap();
    const mascot = {}; // fresh and encapsulated, thus differs from any possible provided value.
    return Object.freeze({
      get: const(base) {
        while (base !== null) {
          const result = et.get(base);
          if (result !== undefined) {
            return result === mascot ? undefined : result;
          }
          base = Object.getPrototypeOf(base);
        }
        return undefined;
      },
      set: const(key, val) {
        et.set(key, val === undefined ? mascot : val);
      },
      has: const(key) {
        return et.get(key) !== undefined;
      },
      delete: const(key) {
        et.set(key, undefined);
      }
    });
  }
```

A Less Aggressive GC Contract?

At (es-discuss:011705) I (MarkM) wrote regarding the contrast with the names strawman:

```
 If the only semantic difference is (not normally observable) less aggressive GC
 obligations, great. I'm confident we can converge those.
```

Given the gc semantics of weak maps, the gc obligations implied by the above `ExplicitSoftField` implementation *vs.* names seems to be:

Labels	ExplicitSoftFields	Names
-	Given field F where `F.get(K) === V`	Given name F where `K[F] === V`
-	Similarities	
a	F and K may retain V	
b	F must not retain K	
c	F without K must not retain V	
-	Differences	
x	K must not retain F	K may retain F
y	K without F must not retain V	K without F may retain V

Why it might not matter

The classes as sugar strawman uses ExplicitSoftFields to implement class-private instance variables. Similar classes strawmen have suggested using names for similar purposes. How do the above differences affect the gc semantics of these class proposals?

For a given class C, let’s call IC the set of all instances of C. The ExplicitSoftField or name F created by the desugaring is reachable only from C and from IC, and cannot escape from this set. The keys of F are exactly all the members of IC. Under these circumstances, the implied GC obligations seem to be exactly the same. To see why, say that some non-empty subset of IC, SIC, is reachable from outside C and IC. Since F is reachable from all members of SIC (independent of whether F is an ExplicitSoftField or name), SIC’s non-emptiness implies that F may be retained. By rule #b, SIC and F and C together must not retain the remaining members of IC outside of SIC. Since these are not retained, by rule #c the private facets of these instances are also not retained. For all members of SIC, since they retain F, by rule #a, SIC together with F retain all the private facets of SIC.

The reason that the different obligations of ExplicitSoftFields and names don’t matter for classes is that there are no circumstances where a K may be reachable but the corresponding F might not be, since the Ks in question − the instances of C − reference F anyway.

What if it does matter?

Nevertheless, we may want to reduce the gc obligations of ExplicitSoftFields towards that of names. I’m not sure, but I think only difference #y matters. Rule #x *by itself* would only affect how many empty ExplicitSoftFields are retained, since the number of Key-to-Value associations retained is determined by the other rules. We can change our executable spec above to represent these reduced obligations as follows:

```
const ExplicitSoftField = (const(){
  const globalET = WeakMap(); // necessarily reachable
  return const() {
    const et = WeakMap();
    const mascot = {};
    return Object.freeze({
      //...other methods same as before...
      set: const(key, val) {
        et.set(key, val === undefined ? mascot : val); // as before
        globalET.set(key, et);
      }
    });
  };
})();
```

See

The thread beginning at WeakMap API questions?

strawman/inherited_explicit_soft_fields.txt · Last modified: 2010/09/06 04:20 by markm

Show pagesource
Old revisions
Login
Index
Back to top