

Value Objects

Brendan Eich

<brendan@mozilla.org>

Use Cases

- `symbol`, arguably
- `int64`, `uint64`
- `Int32x4`, `Int32x8` (SIMD)
- `float32`
- `Float32x4`, `Float32x8` (SIMD)
- `bignum`
- `decimal`
- `rational`
- `complex`

Overloadable Operators

- | ^ &
- ==
- < <=
- << >> >>>
- + -
- * / %
- ~ *boolean-test unary- unary+*

Preserving Boolean Algebra

- `!=` and `!` are not overloadable to preserve identities including

- $X \text{ ? } A : B \iff !X \text{ ? } B : A$

- $!(X \ \&\& \ Y) \iff !X \ || \ !Y$

- $!(X \ || \ Y) \iff !X \ \&\& \ !Y$

- $X \ != \ Y \iff !(X \ == \ Y)$

Preserving Relational Relations

- $>$ and \geq are derived from $<$ and \leq as follows:
 - $A > B \iff B < A$
 - $A \geq B \iff B \leq A$
- We provide \leq in addition to $<$ rather than derive $A \leq B$ from $\neg (B < A)$ in order to allow the \leq overloading to match the same value object's $==$ semantics, or otherwise to be customized

Strict Equality Operators

- The strict equality operators, `===` and `!==`, cannot be overloaded
- They work on frozen-by-definition value objects via a structural recursive strict equality test
- Same-object-reference remains a fast-path optimization

Why Not Double Dispatch?

- Left-first asymmetry (v value, n number):
 - $v + n \implies v.add(n)$
 - $n + v \implies v.radd(n)$
- Anti-modular: exhaustive other-operand type enumeration required in operator method bodies
- Consequent loss of compositionality: `complex` and `rational` cannot be composed to make `ratplex` without modifying source or wrapping in proxies

Cacheable Multimethods

- Proposed in 2009 by Christian Plesner Hansen (Google) in es-discuss
- Avoids double-dispatch drawbacks from last slide: binary operators implemented by 2-ary functions for each pair of types
- Supports PIC optimizations (Christian was on the V8 team)

Binary Operator Example

- For the expression $v + u$
 - Let $p = v.[[Get]](@@ADD)$
 - If p is not an Array, throw a TypeError
 - Let $q = u.[[Get]](@@ADD_R)$
 - If q is not an Array, throw a TypeError
 - Let $r = p$ *intersect* q
 - If $r.length \neq 1$ throw a TypeError
 - Let $f = r[0]$; if f is not a function, throw
 - Evaluate $f(v, u)$ and return the result

API Idea from CPH 2009

```
function addPointAndNumber(a, b) {  
    return Point(a.x + b, a.y + b);  
}
```

```
Function.defineProperty('+', addPointAndNumber, Point, Number);
```

```
function addNumberAndPoint(a, b) {  
    return Point(a + b.x, a + b.y);  
}
```

```
Function.defineProperty('+', addNumberAndPoint, Number, Point);
```

```
function addPoints(a, b) {  
    return Point(a.x + b.x, a.y + b.y);  
}
```

```
Function.defineProperty('+', addPoints, Point, Point);
```

Literal Syntax

- `int64(0)` ==> `0L` // as in C#
- `uint64(0)` ==> `0UL` // as in C#
- `float32(0)` ==> `0f` // as in C#
- `bignum(0)` ==> `0I` // as in F#
- `decimal(0)` ==> `0m` // or `M`, C/F#
- We want a syntax extension mechanism, but declarative not runtime API
- This suggests declarative syntax for operator definition -- and scoped usage too

To new or not to new?

- new connotes reference type semantics, heap allocation, mutability by default (that's JS!)
- Proposal: `new int64(42)` throws (for any scalar or “non-aggregate” value object)
- Option: `new Float32x4(a, b, c, d)` makes a mutable 4-vector, but calling `Float32x4(...)` without `new` means observably immutable, so even stack allocatable (important to enable)
- Alternative: always immutable, but then why allow `new` instead of call to “create a value”

typeof travails and travesties

- Invariant -- these two imply each other in JS:
 - `typeof x == typeof y && x == y`
 - `x === y`
- `0m == 0 && 0L == 0 => 0m == 0L` -- and per the invariant `typeof 0m != typeof 0L`
- Usability favors `typeof 0L == "int64"` and `typeof 0m == "decimal"` anyway
- Making `typeof` extensible requires a per-realm registry with throw-on-conflict

25 July 2013 TC39 Resolutions

- NaN requires separately overloadable `<=` and `<` [Slide 5]
- Intersection means function identity matters, so multimethods can break cross-realm [Slide 9]
- Mark objects that I as `bignum` suffix conflicts with `complex` [Slide 11]
- Always throw on `new --` value objects are never mutable and should not appear to be so, even if aggregate [Slide 12]
- Need to work through any side channel hazard of the `typeof` registry [Slide 13]