Arbitrary precision integers in JavaScript and the web

Daniel Ehrenberg
Igalia

Why?

- Number can represent integers only up to 2^{53} accurately
- 64-bit int use cases
  - GUIDs, protobufs with 64-bit ints
  - For Node, fields of stat, other system calls
  - WebAssembly FFI, Uint64Array
  - Accurate timestamps
- Larger than 64-bit use cases
  - Direct crypto implementation
  - Solving Project Euler problems!

How?

- New primitive type `BigInt`
- Literal syntax
- Operator overloading

```
1n + 2n

0xffn < 0x100n

let x = 1n; x++;
```

Code sample

```
// Takes a BigInt as an argument and returns a BigInt
function nthPrime(nth) {
  function isPrime(p) {
    for (let i = 2n; i < p; i++) {
      if (p % i === 0n) return false;
    }
    return true;
  }
  for (let i = 2n; ; i++) {
    if (isPrime(i)) {
      if (--nth === 0n) return i;
    }
  }
}
```

Code sample: asm.js

```
function add64(stdlib, buffer, aIndex, bIndex) {
  "use asm";
  var cast = stdlib.BigInt.asUintN;
  var values = new stdlib.Uint64Array(buffer);
  aIndex = aIndex|0;
  bIndex = bIndex|0;
  var aValue = cast(values[aIndex>>3], 64);
  var bValue = cast(values[bIndex>>3], 64);
  return cast(a + b, 64);
}
```

Library features

- Uint64Array, Int64Array
  - elements are BigInts
- BigInt static methods
  - BigInt.parseInt
  - BigInt.asUintN, BigInt.asIntN

No implicit coercion

- The point of BigNums: maintain integer precision
- No good answer for BigInt + Number
- 0.5 + 2n**53n has an answer outside of the range
- Solution: Require explicit casts
- Call Number(), BigInt() to convert
  - E.g., Number(1n) + 2.0 => 3.0

No implicit coercion

- What to do when coercion would happen?
- Solution: throw a TypeError, on:
  - BigInt + Number
  - BigInt < Number (?)
  - + BigInt
  - Passing a BigInt to any Web API expecting a Number
  - Any ToNumber() call in the JS
  - etc
- BigInt === Number ⇒ false

Optimization potential

- Use Smi infrastructure to optimize as Int64 (sometimes)
- Could work with asm.js
- Multiple browsers expressed excitement and optimism about implementing with good performance

Integration into the Web Platform

- Disabled by default--requires explicit cast
- Any Web Platform APIs want to use this?

??????

References

- https://github.com/littledan/proposal-bigint

Backup slides

Comparison semantics: current proposal

`1 < 1n`	TypeError
`1 == 1n`	TypeError
`1 === 1n`	false

Comparison semantics: Allow semantic comparison

`1 < 1n`	false
`1 == 1n`	true
`1 === 1n`	false

Comparison semantics: Strictly prohibited

`1 < 1n`	TypeError
`1 == 1n`	TypeError
`1 === 1n`	TypeError

Comparison semantics: Avoid throwing from ==

`1 < 1n`	TypeError
`1 == 1n`	false
`1 === 1n`	false