10th February 2020

BigInt

A recent addition

This is a recent addition to the language. You can find the current state of support at https://caniuse.com/#feat=bigint.

`BigInt` is a special numeric type that provides support for integers of arbitrary length.

A bigint is created by appending `n` to the end of an integer literal or by calling the function `BigInt` that creates bigints from strings, numbers etc.

```
1  const bigint = 1234567890123456789012345678901234567890n;
2
3  const sameBigint = BigInt("1234567890123456789012345678901234567890");
4
5  const bigintFromNumber = BigInt(10); // same as 10n
```

Math operators

`BigInt` can mostly be used like a regular number, for example:

```
1  alert(1n + 2n); // 3
2
3  alert(5n / 2n); // 2
```

Please note: the division `5/2` returns the result rounded towards zero, without the decimal part. All operations on bigints return bigints.

We can't mix bigints and regular numbers:

```
1  alert(1n + 2); // Error: Cannot mix BigInt and other types
```

We should explicitly convert them if needed: using either `BigInt()` or `Number()`, like this:

```
1  let bigint = 1n;
2  let number = 2;
3
```

```
// number to bigint
alert(bigint + BigInt(number)); // 3

// bigint to number
alert(Number(bigint) + number); // 3
```

The conversion operations are always silent, never give errors, but if the bigint is too huge and won't fit the number type, then extra bits will be cut off, so we should be careful doing such conversion.

The unary plus is not supported on bigints

The unary plus operator `+value` is a well-known way to convert `value` to a number.

On bigints it's not supported, to avoid confusion:

```
let bigint = 1n;

alert( +bigint ); // error
```

So we should use `Number()` to convert a bigint to a number.

Comparisons

Comparisons, such as `<`, `>` work with bigints and numbers just fine:

```
alert( 2n > 1n ); // true

alert( 2n > 1 ); // true
```

Please note though, as numbers and bigints belong to different types, they can be equal `==`, but not strictly equal `===`:

```
alert( 1 == 1n ); // true

alert( 1 === 1n ); // false
```

Boolean operations

When inside `if` or other boolean operations, bigints behave like numbers.

For instance, in `if`, bigint `0n` is falsy, other values are truthy:

```
if (0n) {
  // never executes
}
```

Boolean operators, such as `||`, `&&` and others also work with bigints similar to numbers:

```
alert( 1n || 2 ); // 1 (1n is considered truthy)

alert( 0n || 2 ); // 2 (0n is considered falsy)
```

Polyfills

Polyfilling bigints is tricky. The reason is that many JavaScript operators, such as `+`, `-` and so on behave differently with bigints compared to regular numbers.

For example, division of bigints always returns a bigint (rounded if necessary).

To emulate such behavior, a polyfill would need to analyze the code and replace all such operators with its functions. But doing so is cumbersome and would cost a lot of performance.

So, there's no well-known good polyfill.

Although, the other way around is proposed by the developers of JSBI library.

This library implements big numbers using its own methods. We can use them instead of native bigints:

Operation	native `BigInt`	JSBI
Creation from Number	`a = BigInt(789)`	`a = JSBI.BigInt(789)`
Addition	`c = a + b`	`c = JSBI.add(a, b)`
Subtraction	`c = a - b`	`c = JSBI.subtract(a, b)`
...	...	...

…And then use the polyfill (Babel plugin) to convert JSBI calls to native bigints for those browsers that support them.

In other words, this approach suggests that we write code in JSBI instead of native bigints. But JSBI works with numbers as with bigints internally, emulates them closely following the specification, so the code will be "bigint-ready".

We can use such JSBI code "as is" for engines that don't support bigints and for those that do support – the polyfill will convert the calls to native bigints.

References

- MDN docs on BigInt.
- Specification.

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)