Iterator Helpers Update

July 2022 · 91st meeting of TC39

Michael Ficarra & Kevin Gibbons

Proposal Summary

https://github.com/tc39/proposal-iterator-helpers

Proposal Summary

- Iterator()
- Iterator.from(O)
- Iterator.prototype
  - .constructor
  - .map(mapper)
  - .filter(predicate)
  - .take(limit)
  - .drop(limit)
  - .indexed()
  - .flatMap(mapper)
  - .reduce(reducer [, initialValue])
  - .toArray()
  - .forEach(effect)
  - .some(predicate)
  - .every(predicate)
  - .find(predicate)
  - [@@toStringTag]

- AsyncIterator()
- AsyncIterator.from(O)
- AsyncIterator.prototype
  - .constructor
  - .map(mapper)
  - .filter(predicate)
  - .take(limit)
  - .drop(limit)
  - .indexed()
  - .flatMap(mapper)
  - .reduce(reducer [, initialValue])
  - .toArray()
  - .forEach(effect)
  - .some(predicate)
  - .every(predicate)
  - .find(predicate)
  - [@@toStringTag]

Proposal Summary

- we consider this to be a minimal first step
- mostly overlaps with familiar Array methods
- very little flexibility in their design space
- many promising proposals for future additions (see later)

Resolutions

throw RangeError in take/drop when input is NaN-ish

https://github.com/tc39/proposal-iterator-helpers/pull/181

- `iterator.drop("not a number") // RangeError, not .drop(0)`
- `iterator.drop() // RangeError, not .drop(0)`
- `iterator.drop("2") // .drop(2)`

don't preserve the generator protocol

https://github.com/tc39/proposal-iterator-helpers/pull/194

> Iterator helpers can't coherently preserve the generator protocol, so they shouldn't try.

AsyncIterator.from uses the PromiseResolve AO

https://github.com/tc39/proposal-iterator-helpers/pull/197

- Now does `Promise.resolve(underlying.next())`
  - (with a try-catch to turn sync errors into rejected promises)
- Previously did new `Promise(res => res(underlying.next())`
- Difference is fewer ticks, and the identity of the Promise

AsyncIterator#toArray does not await promises

https://github.com/tc39/proposal-iterator-helpers/issues/168

matches for-await-of and Array.fromAsync proposal

Open Questions

or issues

Iterator.prototype.flat?

- We have map and flatMap, but not flat
- is `flatMap(x => x)` good enough for now?

Iterator.prototype.toAsync?

AsyncIterator.from(iter) works, but is kind of annoying in a chain

```
arr.values()
  .map(foo)
  .toAsync()
  .filter(async x => await bar(x))
```

```
AsyncIterator.from(
  arr.values()
    .map(foo)
)
  .filter(async x => await bar(x))
```

https://github.com/tc39/proposal-iterator-helpers/pull/202

Should .drop be eager?

- Currently specified as lazy
- Will wait until resulting iterator is advanced for the first time to advance past dropped entries

web compat: writability of toStringTag on Iterator.prototype

https://github.com/tc39/proposal-iterator-helpers/issues/115

- regenerator-runtime depends on writing to toStringTag of Generator.prototype
  - in older versions, which are shipped in a lot of places
- Generator.prototype inherits from Iterator.prototype
- toStringTag added as non-writable on Iterator.prototype (and everything else)
- strict mode code, so it throws
- possible solutions:
  - writable toStringTag
  - don't add toStringTag
  - ???

reachability of new intrinisics from the global

https://github.com/tc39/proposal-iterator-helpers/issues/173

- %WrapForValidIteratorPrototype% and %WrapForValidAsyncIteratorPrototype% are not reachable via repeated member access from the global
- don't think this should be a requirement imposed on all new intrinsics

Future Work

THESE ARE NOT PART OF THIS PROPOSAL

More helpers

- .takeWhile(predicate) / .takeUntil(predicate) / .dropWhile(predicate) / .dropUntil(predicate)
- .zip(otherIter) /
  .zipWith(combineFn, otherIter) /
  .zipLongest(fillerElem, otherIter)
- .tap(effect)
- .chunks(length) /
  .windows(size)
- Iterator.concat(...iterators)
- Iterator.repeat(value)
- probably other stuff!

Cleanup

https://github.com/tc39/proposal-iterator-helpers/issues/162

https://github.com/tc39/proposal-iterator-helpers/issues/164

```
function idsToUser(idIter) {
  let dbHandle = getDbHandle();
  return idIter.map(i => dbHandle.getId(i));
  // how to close dbHandle when this iterator is closed?
}
```

Call for Stage 2 Reviewers!