Closing iterators

Dave Herman
Iterators need an early-disposal protocol
• Closing a synchronous sequence is a bit of an abstract question (though not irrelevant).

• But we will want asynchronous sequences, and closing those is definitely important.

• We should future-proof for symmetry.
for (let x of heap.deflateGzipData()) {
  ...
  break;
  ...
}
... 

deflateGzipData() {
    let i = this.malloc(...);
    return {
        next() { ... }, // iterate typed array
        return() { this.free(i) }
    }
}
... deflateGzipData: function*() {
    let i = this.malloc(...);
    try {
       ... // iterate typed array
    } finally {
       this.free(i)
    }
} ...
...
for (await x of db.select(...)) {
  ...
  ...
  break;
  ...
  ...
}
... select(query) {
  let records = ...;
  try {
  } finally {
    records.close();
  }
}
...
When does the early-return get called?
for (let x of y) {
  ...
  ...
  break;
  ...
  ...
}
outer:
for (let i = 0; i < N; i++) {
  for (let x of y) {
    ...
    break outer;
    ...
    ...
  }
}
}
outer:

for (let i = 0; i < N; i++) {
    for (let x of y) {
        ...
        ...
        continue outer;
        ...
    }
}

}
for (let x of y) {
  ...
  throw new Error();
  ...
}
for (let x of y) {
    ...
    f(); // throws
    ...
    ...
}
for (let x of y) {
    ...
    return;
    ...
    ...
}
for (let x of y) {
  ...
  yield; // returns via .return()
  ...
}

for (let x of y) {
  ...
  yield* g(); // returns via .return()
  ...
  ...
}

• In short: any abrupt completion of the loop.

• Normal completion should not call the method; in that case the iterator itself decided to close.
What if the iterator refuses to stop?
function* f() {
    try {
        yield;
        yield;
    } finally {
        yield;
    }
}
• Disallow `yield` in a `finally`?

• *No!* Bad idea – and doesn't solve the problem.
function* f() {
    try {
        try {
            try {
                yield;
            } finally { throw "override"; }
        } catch (ignore) {} 
        yield;
    }
}
function* f() {
    try {
        yield* g();
    }
    catch (ignore) {
    }
    yield;
}
function* g() {
    try {
        yield;
    } finally {
        throw "override";
    }
}
function* g() {
    try {
        yield;
        yield;
    } finally {
        cleanup();
    }
}
• Disallow `yield` dynamically, once we start the disposal process?

• *No!* Another bad idea, and doesn't solve the problem for hand-written iterators.
• Better framing: `for...of` gives iterators the **opportunity** to do resource disposal.

• Impossible to force an iterator to stop iterating.

• Still, failure to stop iterating is probably a bug in the *contract* between the iterator and the loop.
interface IterationResult {
  value: any,
  done: boolean
}

interface Generator extends Iterator {
  next(value: any?) : IterationResult,
  throw(value: any?) : IterationResult
}
interface IterationResult {
  value: any,
  done: boolean
}

interface Generator extends Iterator {
  next(value: any?): IterationResult,
  throw(value: any?): IterationResult,
  return(value: any?): IterationResult
}
interface Iterator {
    next(value: any?) : IterationResult,
    return?() : IterationResult
}
• On abrupt exit, `for...of` looks for `return` method.

• If present, it calls the method with no arguments.

• If the result has falsy `done` property, throw an error.
Bikeshed city
interface Iterator {
    next(value: any?) : IterationResult,
    close?() : IterationResult
}
5 Jun 14 Resolutions
• Agreed to design, schedule permitting.

• Early termination method is called \texttt{return}.

• If we run out of time, stopgap semantics:
  
  • \texttt{reject yield} in \texttt{try} blocks with \texttt{finally} clause
  
  • early exit from \texttt{for...of} puts generator in \texttt{GeneratorComplete} state