Cancellation in JavaScript

Token-based cancellation in JavaScript using the unifying .NET cancellation pattern

By Andrew Nosenko - @noseratio

Intro

- Cancellation is a pattern used to stop an asynchronous operation or a long-running synchronous operation.

- .NET uses a unified model for cooperative cancellation, based on a lightweight object called a **Cancellation Token**, and it's parent object called **Cancellation Token Source**.

- Despite many takes, an established cancellation framework is somehow still missing in JavaScript world. The current TC39 proposal is based on the .NET model (it's in the early stage 1).

Why is cancellation important?

- It’s natural. Many real-life process can be interrupted by an external request.
- It’s **useful** for developing **front-end workflows**.
- It may **improve the scalability of the back-end**, by stopping pending operations which are no longer relevant.
- Proper cleanup and release of resources with ***try/catch/finally***.
- If implemented properly, **cancellation is pervasive through all code layers** and from the front-end to the back-end, similar to *async/await* itself.

Autocomplete as a well-known example

1. The user types a character.
2. We start an async delay, e.g.: ***await new Promise(r => setTimeout(r));***
3. When the delay is completed, we start and await a REST ***fetch*** call.
4. Meanwhile, the user may have typed another character, while we still haven't received the server's response for the fetch.
5. **Ideally, we now should cancel the fetch (or the delay, whichever is pending)**, before we start a new sequence of these async calls.
6. On the server, we may also want to cancel a pending call to 3rd party microservice API that we use for auto-complete, to avoid extra charges
7. Now we can start another delay and the fetch for the newly entered text, and so on.

An overview of the .NET cancellation pattern

In a nutshell, the cancellation pattern in .NET deals with the following three parts:

- ***CancellationTokenSource*** class. This is the producer part of the API, the code which triggers cancellation. It's normally external to the asynchronous operation itself. The cancellation is requested via *CancellationTokenSource.cancel()*.
- ***CancellationToken*** struct. This is the consumer part of the API, returned by *CancellationTokenSource.Token*, used to observe external cancelation requests.
- ***OperationCanceledException*** exception to tell the operation was cancelled.

Observing the cancellation request

- A *cancellation token* lets the code conducting a long-running operation to observe the cancellation request and act upon it to end such operation.
- We observe the request via a callback registered with ***CancellationToken.register()***
  - **Think of it as of *addEventListener* for a hypothetical *cancel* event.**
  - Back to our Autocomplete example, that's where we may want to call *clearTimer()* and reject the delay promise, or call *AbortController.abort()* to cause the rejection of the *fetch* promise.
- On tight loops, we may want to call ***CancellationToken.ThrowIfCancellationRequested()*** to poll for a cancellation requests.

Cancelling a hierarchy of async operations

- Cancelling a complex tree or a graph of workflows is possible with linked token sources, created with ***CancellationTokenSource.CreateLinkedTokenSource(linkedTokens)***

- For example, we may want to create a new linked token source and pass its token to a child workflow we're about to start (as later in the demo).

- The child workflow then can be cancelled internally by the parent workflow (that created it), or as a part of the whole cancellation tree, if requested externally.

What’s up in the land of JavaScript?

Ben Lesh in his “Promise Cancellation Is Dead — Long Live Promise Cancellation!" article offers the following options:

- Use Bluebird library
- Use Another Promise (as a token)
- Use Rx Subscriptions
- Ditch promises and just use Observables

I might add that *AbortController/AbortSignal/AbortError* is being increasingly adopted by Node.js. I personally was looking for a **library that already implements the .NET model for JavaScript**, and I’ve found one: **the Prex library.**

Promise Extensions for JavaScript (prex)

- A library created by Ron Buckton, a Senior SDE for TypeScript, a member of TC39 committee and the author of the current ECMAScript Cancellation proposal.
- **Prex provides lots of async counterpart APIs** for what we use and love in .NET, for example:
  - *Deferred* (.NET *TaskCompletionSource*);
  - *Semaphore* (.NET *SemaphoreSlim*);
  - *AsyncQueue* (.NET Channel);
  - *delay* (.NET *Delay*);
- **And of course, for cancellation:**
  - *CancellationTokenSource* , *CancellationToken*, *CancelError* *(.NET OperationCanceledException)*

Use token-based cancellation for native APIs

We can use *CancellationTokenSource* and *CancellationToken* to wrap many native cancellation and clean-up APIs in JavaScript, both in front-end and Node.js.
For example:

- *AbortController* and *AbortSignal* for HTTP *fetch*;
- *clearTimeout* for *setTimeout*,
  cancelAnimationFrame for *requestAnimationFrame*
- *stream.end or stream.destroy* in Node.js.
- Cancelling gRPC calls

A typical code pattern

For example, calling ***fetch*** for a URL while observing cancellation:

```
async function fetchUrl(url, token) {
  const abortController = new AbortController();
  const abortSignal = abortController.signal;
  const rego = token.register(() => abortController.abort());
  try {
    const response = await fetch(url, { abortSignal });
    // may throw CancelError
    token.throwIfCancellationRequested();
    return await response.json();
  }
  finally {
    rego.unregister();
  }
}
```

APIs that can’t be cancelled natively

There is not much we can do, besides we can become *disinterested* in the results and bail out earlier by using a **dedicated cancellation promise** and ***Promise.any***:

```
async function callApi(url, token) {
  const cancelDeferred = new prex.Deferred();
  const rego = token.register(() => cancelDeferred.resolve());
  try {
    // it's tempring to use prex.Delay(token),
    // it may result in an unhandled rejection events
    const apiPromise = asyncApi();
    await Promise.any([
      apiPromise.catch(console.warn),
      cancelDeferred.promise]);
    token.throwIfCancellationRequested();
    return await apiPromise;
  }
  finally {
    rego.unregister();
  }
}
```

Demo time

A pure client-side JavaScript app that draws a mouse trail using canvas.

https://github.com/noseratio/cancellation-talk

Or simply search for “@noseratio”

Final point

If you already use and love the cancellation pattern in .NET/C#, bring it on to JavaScript/TypeScript, both for front-end and back-end!

References

- ECMAScript Cancellation (the current TC39 proposal)
- Cancellation in Managed Threads (in .NET)
- Promise Cancellation Is Dead — Long Live Promise Cancellation, by Ben Lesh (@BenLesh)
- Promise Extensions for JavaScript (prex), by Ron Buckton (@rbuckton)