TooTallNate / **once** Public

Code Issues Pull requests Actions Projects Security and quality

Commit b9f43cc

TooTallNate committed on Feb 9 · 0 / 12 · Verified

Fix promise hang when `AbortSignal` is aborted

When an `AbortSignal` was aborted, the promise returned by `once()` would remain permanently pending, causing any await usage to hang indefinitely. Now the promise rejects with an `AbortError` when the signal is aborted, including the case where the signal is already aborted at call time.

Fixes #8.

master · v3.0.1

1 parent a8c9dc3 commit b9f43cc

3 files changed +80 -8

Top

Filter files…

- src
  - index.ts
  - types.ts
- test
  - once.test.ts

Search within code

src/index.ts

```
@@ -15,7 +15,7 @@ export default function once<
15 15 ): Promise<EventListenerParameters<Emitter, Event>> {
16 16     return new Promise((resolve, reject) => {
17 17         function cleanup() {
```

```
18      -             signal?.removeEventListener('abort', cleanup);
        18  +         signal?.removeEventListener('abort', onAbort);
19      19            emitter.removeListener(name, onEvent);
20      20            emitter.removeListener('error', onError);
21      21        }
          @@ -27,7 +27,17 @@ export default function once<
27      27            cleanup();
28      28            reject(err);
29      29        }
30      -         signal?.addEventListener('abort', cleanup);
        30  +     function onAbort() {
        31  +         cleanup();
        32  +         const err = new Error('The operation was aborted');
        33  +         err.name = 'AbortError';
        34  +         reject(err);
        35  +     }
        36  +     if (signal?.aborted) {
        37  +         onAbort();
        38  +         return;
        39  +     }
        40  +     signal?.addEventListener('abort', onAbort);
31      41        emitter.on(name, onEvent);
32      42        emitter.on('error', onError);
33      43    });
```

src/types.ts

```
          @@ -30,6 +30,7 @@ export type EventListenerParameters<
30      30    export type WithDefault<T, D> = [T] extends [never] ? D : T;
31      31
32      32    export interface AbortSignal {
        33  +     aborted: boolean;
33      34        addEventListener: (name: string, listener: (...args: any[]) => any) => void;
34      35        removeEventListener: (
35      36            name: string,
```

test/once.test.ts

```
          @@ -99,19 +99,80 @@ describe('once()', () => {
```

```
 99   99            await new Promise((r) => process.nextTick(r));
100  100
101  101            expect(wasResolved).toEqual(true);
     102  +     });
     103  +
     104  +     it('should reject with AbortError when signal is aborted', async () => {
     105  +         const emitter = new EventEmitter();
     106  +         const controller = new AbortController();
     107  +         const { signal } = controller;
102  108
103       -         // Reset
104       -         wasResolved = false;
     109  +         const promise = once(emitter, 'foo', { signal });
105  110
106       -         once(emitter, 'foo', { signal }).then(onResolve, onResolve);
     111  +         controller.abort();
     112  +
     113  +         try {
     114  +             await promise;
     115  +             throw new Error('Should not happen');
     116  +         } catch (err: any) {
     117  +             expect(err.name).toEqual('AbortError');
     118  +             expect(err.message).toEqual('The operation was aborted');
     119  +         }
     120  +     });
107  121
108       -         // This time abort
     122  +     it('should reject immediately if signal is already aborted', async () => {
     123  +         const emitter = new EventEmitter();
     124  +         const controller = new AbortController();
     125  +         controller.abort();
     126  +
     127  +         const { signal } = controller;
     128  +
     129  +         try {
     130  +             await once(emitter, 'foo', { signal });
     131  +             throw new Error('Should not happen');
     132  +         } catch (err: any) {
     133  +             expect(err.name).toEqual('AbortError');
     134  +             expect(err.message).toEqual('The operation was aborted');
```

```
        }
    });

    it('should not resolve with event after being aborted', async () => {
        const emitter = new EventEmitter();
        const controller = new AbortController();
        const { signal } = controller;

        let wasResolved = false;
        let wasRejected = false;

        const promise = once(emitter, 'foo', { signal }).then(
            () => { wasResolved = true; },
            () => { wasRejected = true; }
        );

        // Abort first, then emit
        controller.abort();
        emitter.emit('foo');

-       // Promise is fulfilled on next tick, so wait a bit
-       await new Promise((r) => process.nextTick(r));
        await promise;

        expect(wasResolved).toEqual(false);
        expect(wasRejected).toEqual(true);
    });

    it('should clean up listeners on the emitter when aborted', async () => {
        const emitter = new EventEmitter();
        const controller = new AbortController();
        const { signal } = controller;

        once(emitter, 'foo', { signal }).catch(() => {});

        expect(emitter.listenerCount('foo')).toEqual(1);
        expect(emitter.listenerCount('error')).toEqual(1);

        controller.abort();
```

```
+          await new Promise((r) => process.nextTick(r));
+
+          expect(emitter.listenerCount('foo')).toEqual(0);
+          expect(emitter.listenerCount('error')).toEqual(0);
        });
    });
```

Comments 0

Please sign in to comment.