FetchEventLike

August 17, 2026

FetchEventLike

Kind: Class

Source: `src/types.ts`

`FetchEventLike` represents the event object handled during a fetch request. It coordinates response handling, exception pass-through behavior, and asynchronous work tied to the request lifecycle.

Methods

Method	Signature	Returns
`respondWith`	`respondWith(promise: Response | Promise)`	
`passThroughOnException`	`passThroughOnException()`	`void`
`waitUntil`	`waitUntil(promise: Promise<void>)`	`void`

Properties

Property	Type
`request`	`Request`

Diagram

```
graph LR
  Handler[Fetch handler] --> Event[FetchEventLike]
  Event --> Response[respondWith()]
  Event --> Exceptions[passThroughOnException()]
  Event --> Background[waitUntil()]
```

Usage

```ts
function handleFetch(event: FetchEventLike): void {
  event.passThroughOnException();

  event.waitUntil();

  event.respondWith();
}
```

AI Coding Instructions

- Pass `FetchEventLike` into fetch handler code instead of depending on a platform-specific event type.
- Call `respondWith()` when the handler should control the request response.
- Call `passThroughOnException()` before work that may fail when fallback request behavior is desired.
- Register request-related asynchronous work with `waitUntil()` so it remains associated with the event.

How it works

`FetchEventLike` is an exported abstract TypeScript class that describes the FetchEvent-shaped execution context used by Hono. It declares, but does not implement, four members: a read-only `Request`, `respondWith`, `passThroughOnException`, and `waitUntil`. [src/types.ts:2773-2778]

- `request` must be a read-only `Request`. [src/types.ts:2774]
- `respondWith` accepts either a `Response` or `Promise<Response>` and returns `void`. [src/types.ts:2775]
- `passThroughOnException` takes no arguments and returns `void`. [src/types.ts:2776]
- `waitUntil` accepts `Promise<void>` and returns `void`. [src/types.ts:2777]

Hono stores this type as one possible request execution context, alongside `ExecutionContext`, when constructing a `Context`. [src/context.ts:237-252] During request dispatch, `HonoBase` passes its received execution context into that `Context`. [src/hono-base.ts:407-428]

`Context.event` exposes the stored value as `FetchEventLike` only when the value exists and has a `respondWith` property. Otherwise, it throws `Error('This context has no FetchEvent')`. [src/context.ts:377-382] This is the visible runtime validation for access

through `c.event`; the check does not verify the declared `request`, `passThroughOnException`, or `waitUntil` members. [src/context.ts:377-382]

The service-worker adapter passes its incoming event to `app.fetch` as the execution context and calls that event's `respondWith` with the app response promise. [src/adapter/service-worker/handler.ts:25-35] Its local service-worker `FetchEvent` declaration has a read-only request, `respondWith`, and `waitUntil`; its `respondWith` also accepts `PromiseLike<Response>`. [src/adapter/service-worker/types.ts:1-14]

`FetchEventLike` contains no method bodies, validation logic, error handling, or direct side effects of its own; those depend on the concrete event object. [src/types.ts:2773-2778]