Context

August 18, 2026

Context

Kind: Class

Source: `src/context.ts`

`Context` owns an internal response factory through its private `#newResponse()` method. The factory returns a `Response`, keeping response creation inside the context implementation.

Methods

Method	Signature	Returns
`#newResponse`	`#newResponse(data: Data	null, arg: StatusCode

Properties

Property	Type
`#rawRequest`	`Request`
`#req`	`HonoRequest<P, I['out']>`
`env`	`E['Bindings']`
`#var`	`Map<unknown, unknown>`
`finalized`	`boolean`
`error`	`Error`
`#status`	`StatusCode`
`#executionCtx`	`FetchEventLike`
`#res`	`Response`
`#layout`	`Layout<PropsForRenderer & { Layout: Layout }>`

Property	Type
`#renderer`	`Renderer
`#notFoundHandler`	`NotFoundHandler
`#preparedHeaders`	`Headers
`#matchResult`	`Result<[H, RouterRoute]>
`#path`	`string
`render`	`Renderer`
`setLayout`	`any`
`getLayout`	`any`
`setRenderer`	`any`
`header`	`SetHeaders`
`status`	`any`
`set`	`Set< IsAny<E> extends true ? { Variables: ContextVariableMap & Record<string, any> } : E >`
`get`	`Get< IsAny<E> extends true ? { Variables: ContextVariableMap & Record<string, any> } : E >`
`newResponse`	`NewResponse`
`body`	`BodyRespond`
`text`	`TextRespond`
`json`	`JSONRespond`
`html`	`HTMLRespond`
`redirect`	`any`
`notFound`	`any`

Where it refuses work

- `Context` stops the work with an early return when `!this.#var`.

Diagram

```
graph LR
  Context --> Factory["#newResponse()"]
```

```
  Factory --> Response
```

Usage

```
// Pattern for code added inside src/context.ts
class Context {
  #newResponse(): Response {
    return new Response()
  }

  createResponse(): Response {
    return this.#newResponse()
  }
}
```

AI Coding Instructions

- Keep response construction behind `#newResponse()`.
- Call the private factory with `this.#newResponse()` only from within `Context`.
- Return the factory result as a `Response` without exposing the private method.
- Treat `Response` as the integration point for code that consumes context output.

How it works

`Context<E, P, I>` is the per-request state object used by handlers and middleware. It stores the incoming `Request`, environment bindings, optional execution context, route-match data, response state, a handler error, renderer/layout state, and context variables. Its generic parameters type the environment, route path, and validated request input. src/context.ts:293-344

- The constructor requires a `Request`. When options are passed, it stores `executionCtx`, `env`, `notFoundHandler`, `path`, and `matchResult`; without options, `env` remains its initialized empty object. src/context.ts:315-315 src/context.ts:352-361
- `req` lazily constructs and caches a `HonoRequest` around the original request, path, and match result. `HonoRequest` stores the raw request and path and receives the route-match result for parameter lookup. src/context.ts:366-369 src/request.ts:69-77
- `env` is a public bindings field, `error` is a public `Error | undefined` field, and `finalized` is a public boolean initialized to `false`. Middleware composition assigns

`error` when a handler throws an `Error` and an error handler exists. src/context.ts:315-317 src/context.ts:333-333 src/compose.ts:50-59

Runtime context access

- `event` returns the stored execution context only when it exists and has a `respondWith` property; otherwise it throws `Error('This context has no FetchEvent')`. src/context.ts:377-383
- `executionCtx` returns the stored execution context cast as `ExecutionContext`; it throws `Error('This context has no ExecutionContext')` when none was supplied. src/context.ts:391-397
- The `ExecutionContext` type declares `waitUntil`, `passThroughOnException`, `props`, and optional `exports`. src/context.ts:31-52

Response state and headers

- Reading `res` returns the current response, or creates and caches an empty `Response` with the accumulated prepared headers when no response has been set. src/context.ts:403-407
- Assigning `res` marks the context finalized. If a response already exists, the setter clones the incoming response, then copies prior response headers over it except `content-type`; for `set-cookie`, it replaces the incoming cookie headers with all cookies from the prior response. src/context.ts:414-434
- During middleware composition, a truthy handler result is assigned to `context.res` when the context is not finalized; an error-handler result is assigned even if it was already finalized. src/compose.ts:67-70
- `header(name, value, { append })` writes to the current response headers or to prepared headers if no response exists. An `undefined` value deletes the header; `append: true` appends it; otherwise it replaces it. If the context is finalized, it first clones the response before changing headers. src/context.ts:515-527
- `status(status)` stores a status code for later response construction. src/context.ts:529-531
- `newResponse(data, statusOrInit, headers)` creates a `Response`. It starts with current response headers, if any, otherwise prepared headers; merges headers from an init object and explicit headers; preserves multiple `set-cookie` values while merging init headers; and selects a numeric argument's status, an init object's status, or the stored status in that order. src/context.ts:604-654

Response helpers

- `body(data, statusOrInit?, headers?)` delegates to `newResponse`. Its overloads statically restrict non-null bodies to contentful status codes, while `null` may use any `StatusCode`. src/context.ts:122-141 src/context.ts:677-681
- `text(text, statusOrInit?, headers?)` returns text with a default `Content-Type` of `text/plain; charset=UTF-8`. If there are no prepared headers, stored status, arguments beyond text, explicit headers, or finalized response, it directly constructs `new Response(text)` instead. src/context.ts:279-285 src/context.ts:695-707
- `json(object, statusOrInit?, headers?)` serializes the value with `JSON.stringify` and constructs a response whose default `Content-Type` is `application/json`. src/context.ts:721-734
- `html(html, statusOrInit?, headers?)` constructs a response whose default `Content-Type` is `text/html; charset=UTF-8`. For an object input, including a `Promise<string>`, it resolves it through `resolveCallback` and returns a promise of the response; for a string, it returns the response directly. src/context.ts:220-230 src/context.ts:736-746
- `redirect(location, status?)` sets `Location`, converts the location with `String`, URI-encodes it if it contains characters outside the `\x00` – `\xFF` range, and returns an empty response with the supplied redirect status or `302`. src/context.ts:763-775
- `notFound()` invokes the configured not-found handler with this context. If none exists, it caches a handler that returns an empty `Response`. src/context.ts:789-792

Variables and rendering

- `set(key, value)` lazily creates an internal `Map` and stores the value. `get(key)` returns the stored value or `undefined` if no variable map exists or the key is absent. src/context.ts:546-556 src/context.ts:571-580
- `var` returns `{}` when no variables have been set; otherwise it returns `Object.fromEntries` of the internal map. Its type is read-only, but each access creates a plain object from the current map contents. src/context.ts:593-602
- `render` calls the configured renderer. If no renderer has been set, it caches a default renderer that delegates to `html`. `setRenderer` replaces that renderer. src/context.ts:448-451 src/context.ts:495-497
- `setLayout(layout)` stores and returns the layout function; `getLayout()` returns the stored layout or `undefined`. src/context.ts:459-465 src/context.ts:472-472

Used by

2 references from 2 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (2)

- `EventContext` — `src/adapter/cloudflare-pages/handler.ts`:12
- `HonoOptions` — `src/hono-base.ts`:46