Router

August 17, 2026

Router

Overview

Router maps an HTTP method and an already-extracted request path to an ordered set of handlers plus route parameters. Its common contract is `add(method, path, handler)` followed by `match(method, path)`, and a match result either carries handler-specific parameter maps directly or pairs parameter-index maps with a captured-value stash. [src/router.ts:29-52; src/router.ts:54-98]

`HonoBase` registers each route after uppercasing its method and merging its path with the application base path; it passes the handler together with route metadata into the configured router. During dispatch, it obtains the request path, calls `router.match`, and places the resulting match data in the request `Context` before invoking the matching handler chain. [src/hono-base.ts:386-398; src/hono-base.ts:407-451]

A default `Hono` instance creates a `SmartRouter` whose candidates are `RegExpRouter` followed by `TrieRouter`, unless the caller supplies a router in the constructor options. [src/hono.ts:26-33] `SmartRouter` records registrations until its first match, then attempts to add the complete recorded route set to each candidate in order; it selects and caches the first candidate that does not throw `UnsupportedPathError`. [src/router/smart-router/router.ts:13-19; src/router/smart-router/router.ts:21-60]

The matching contract returns all applicable handlers rather than a single endpoint. This lets registrations such as a wildcard middleware route and an exact route both participate in the same request, while retaining registration order for the returned handlers. [src/router/common.case.test.ts:129-139; src/router/common.case.test.ts:193-210] The dispatcher either invokes the single matched handler directly or composes the returned handler list when more than one handler matched. [src/hono-base.ts:430-466]

How it works

Registration and route syntax

All router implementations receive a method string, a path string, and an opaque handler value through the shared interface. [src/router.ts:29-52] `METHOD_NAME_ALL` is the string `ALL`; implementations compare a route's method against the requested method and also include `ALL` registrations. [src/router.ts:6-17; src/router/linear-router/router.ts:25-40; src/router/trie-router/node.ts:94-111]

The path utility expands a terminal optional named segment into paths with and without that segment. A route ending in an optional label therefore becomes separate route entries before `LinearRouter`, `TrieRouter`, or `RegExpRouter` store it. [src/utils/url.ts:171-206; src/router/linear-router/router.ts:15-23; src/router/trie-router/router.ts:13-23; src/router/reg-exp-router/router.ts:122-139]

`TrieRouter` is the general fallback implementation. Registration splits a route into slash-separated routing segments while preserving brace-delimited expressions, asks `getPattern` whether each segment is a named parameter, constrained parameter, or wildcard, and inserts the resulting sequence into a node tree. [src/router/trie-router/router.ts:13-27; src/router/trie-router/node.ts:44-84; src/utils/url.ts:16-21; src/utils/url.ts:50-78] Each terminal node stores the handler, the parameter keys that may apply to it, and an insertion score used later to restore registration order. [src/router/trie-router/node.ts:6-14; src/router/trie-router/node.ts:76-84]

At match time, the trie splits the request path and tracks candidate nodes. It first follows a literal child where present, then evaluates wildcard and parameter-pattern children; named values are accumulated into a parameter object for the relevant branch. [src/router/trie-router/node.ts:114-163; src/router/trie-router/node.ts:165-229] A terminal node selects the request method or its `ALL` entry, copies applicable values into that handler's parameter map, and the final list is sorted by registration score when necessary. [src/router/trie-router/node.ts:87-112; src/router/trie-router/node.ts:237-244]

A trailing wildcard has special treatment in the trie: when a literal branch reaches the last request segment, the search also checks a child wildcard, so a route ending in `/*` can match both its prefix and descendants. [src/router/trie-router/node.ts:136-146] A wildcard in the middle of a route consumes a segment and remains a candidate for later segments. [src/router/trie-router/node.ts:149-163] Constrained labels use regular expressions constructed by `getPattern`; when a following literal segment exists, the helper builds a lookahead for that next segment. [src/utils/url.ts:60-74; src/router/trie-router/node.ts:171-208]

The compiled regular-expression path

`RegExpRouter` initially keeps separate method-indexed maps for middleware-like wildcard routes, normal routes, and a `Trie` used to construct matchers. Its constructor creates each of those structures under `ALL`. [src/router/reg-exp-router/router.ts:47-57] When a method is first seen, it creates method-specific maps, clones existing `ALL` handler entries, and inserts the corresponding paths into that method's trie. [src/router/reg-exp-router/router.ts:75-84]

For a path ending in a wildcard, `RegExpRouter` builds a wildcard regular expression, inserts the route into matching method tries, and appends the handler to already-known middleware and route paths that satisfy that wildcard. [src/router/reg-exp-router/router.ts:86-119] For an ordinary route, it expands an optional terminal parameter, inserts each expanded path, carries forward matching wildcard middleware, and records how many named parameters belong to each handler. [src/router/reg-exp-router/router.ts:122-139]

The internal regular-expression trie tokenizes dynamic paths, retains a path-to-handler-index and parameter-association entry for dynamic paths, and treats every character of static paths as a literal node. [src/router/reg-exp-router/trie.ts:10-17; src/router/reg-exp-router/trie.ts:20-56] Its nodes order alternatives so literals precede constrained patterns, ordinary labels, and wildcards; that ordering is encoded by `compareKey` before the tree emits a regular-expression string. [src/router/reg-exp-router/node.ts:13-42; src/router/reg-exp-router/node.ts:137-167]

When `RegExpRouter` first matches, the shared `match` function calls `buildAllMatchers`, replaces the instance's `match` method with a closure over the built matcher map, and immediately delegates the current request to that closure. [src/router/reg-exp-router/matcher.ts:10-33] Building a matcher separates exact static paths into a direct lookup map and converts dynamic path data into a regular expression, a handler-data array, and parameter-index mappings. [src/router/reg-exp-router/router.ts:160-210] Subsequent matches first check the static map; otherwise they run the generated expression, locate the handler list using the empty marker's capture position, and return that list with the expression's captured values. [src/router/reg-exp-router/matcher.ts:14-29]

Matcher construction releases `RegExpRouter`'s mutable registration maps and clears its wildcard-expression cache after compilation. [src/router/reg-exp-router/router.ts:144-157] Consequently, adding a route after matcher construction throws the shared "matcher is already built" error. [src/router/reg-exp-router/router.ts:67-73; src/router.ts:19-22]

The regular-expression trie rejects route sets it cannot represent without ambiguity. It throws its private path marker for duplicate terminals, conflicting literal and dynamic branches, incompatible dynamic branches, named patterns containing unsupported capturing groups, a named `.*` pattern, and ambiguous single-character meta-character patterns. [src/router/reg-exp-router/node.ts:51-135] `RegExpRouter` converts that marker to `UnsupportedPathError`, which is the signal `SmartRouter` catches when choosing the next candidate. [src/router/reg-exp-router/router.ts:59-65; src/router/smart-router/router.ts:32-44]

Other implementations

`LinearRouter` keeps registered triples in an array and scans them on every match. It recognizes exact paths, whole-path wildcards, wildcard-containing paths, and named labels; it accepts a trailing slash for exact and label routes. [src/router/linear-router/router.ts:11-23; src/router/linear-router/router.ts:25-70; src/router/linear-router/router.ts:70-143] A label with a brace expression is executed as a regular expression against the remaining path, while ordinary labels consume through the next slash. [src/router/linear-router/router.ts:80-116] A route containing both labels and wildcards causes `UnsupportedPathError` in this implementation. [src/router/linear-router/router.ts:135-142]

`PatternRouter` converts each registered path into one anchored regular expression and stores it with the method and handler. It converts labels into named capture groups, escapes literal expression metacharacters, makes a terminal wildcard unanchored, and recursively registers the shortened form of a terminal optional segment. [src/router/pattern-router/router.ts:8-42] A malformed generated expression becomes `UnsupportedPathError`; matching executes each same-method or `ALL` expression and returns its named capture groups as parameters. [src/router/pattern-router/router.ts:33-59]

`PreparedRegExpRouter` is for a precomputed matcher shape. Its constructor accepts matcher data and a relocation map, and its `add` operation attaches handlers only to paths known by that relocation map, except whole-path wildcard registrations. [src/router/reg-exp-router/prepared-router.ts:9-17; src/router/reg-exp-router/prepared-router.ts:47-86] An unknown non-wildcard path throws an error stating that the path is not registered. [src/router/reg-exp-router/prepared-router.ts:73-76] `buildInitParams` constructs that data by temporarily registering the input paths in a `RegExpRouter`, extracting matchers, deriving relocation entries, and clearing handler arrays; `serializeInitParams` renders the matcher and relocation data as JavaScript-source text

while preserving regular-expression literals. [src/router/reg-exp-router/prepared-router.ts:95-154; src/router/reg-exp-router/prepared-router.ts:156-165]

Configuration

This part does not read environment variables, feature flags, or process-level settings in the router implementations inspected here; router selection instead comes from the optional `router` field in `Hono` options, while `SmartRouter` receives its candidate routers through constructor input. [src/hono.ts:26-33; src/router/smart-router/router.ts:4-11]

The caller can substitute any implementation satisfying the `Router<T>` interface, including the exported `RegExpRouter`, `PreparedRegExpRouter`, `TrieRouter`, `SmartRouter`, `LinearRouter`, and `PatternRouter` entry points. [src/router.ts:29-52; src/router/reg-exp-router/index.ts:6-7; src/router/trie-router/index.ts:6; src/router/smart-router/index.ts:6; src/router/linear-router/index.ts:6; src/router/pattern-router/index.ts:6]

Wiring

The inbound boundary is route registration from `HonoBase` and request matching during `HonoBase` dispatch. `HonoBase` owns method normalization and base-path merging before calling `add`, owns request-path acquisition before calling `match`, and owns creation of the execution `Context`. [src/hono-base.ts:386-398; src/hono-base.ts:407-428]

The outbound boundary is the `Result<T>` consumed by dispatch and composition. Router implementations return handler values with either direct parameter maps or parameter-index maps plus a captured-value stash; `Context` receives that result, and dispatch passes the handler list to either direct invocation or `compose`. [src/router.ts:67-98; src/hono-base.ts:419-451]

Internally, `TrieRouter` depends on the URL helpers for route segmentation and parameter patterns, while `RegExpRouter` depends on its trie and matcher modules to turn registrations into static lookups and compiled regular expressions. [src/router/trie-router/node.ts:1-4; src/router/reg-exp-router/router.ts:7-11; src/router/reg-exp-router/matcher.ts:1-9] `SmartRouter` depends only on the shared router interface and `UnsupportedPathError`, allowing it to switch from the compiled regular-expression strategy to the trie strategy without changing the dispatch boundary. [src/router/smart-router/router.ts:1-2; src/router/smart-router/router.ts:26-60]

21 entities in `src/router`. Nothing else in this repository depends on it.

What it is made of

Its 21 entities sit in 10 files under `src/router` : 9 classes, 6 type aliases, 3 functions, 2 constants and 1 more.
`matcher.ts` holds 6 of them — more than any other file here.
`PreparedRegExpRouter` declares 4 methods, the widest surface here.

Where work enters

- `LinearRouter` — `src/router/linear-router/router.ts` :11
- `PatternRouter` — `src/router/pattern-router/router.ts` :8
- `HandlerData` — `src/router/reg-exp-router/matcher.ts` :4
- `StaticMap` — `src/router/reg-exp-router/matcher.ts` :5

How it refuses and fails

7 of its components record a refusal or a failure handler.
6 of them refuse work outright, under a condition written into the component itself.
Their `catch` blocks handle a failure that already happened in 3 places.

Boundaries

It depends on `Utils` , and on nothing else in this repository.