Jsx

August 17, 2026

Jsx

Overview

Jsx is the rendering layer that turns JSX elements and function components into either escaped HTML for server output or mutable browser DOM. Its public JSX entrypoint exports element construction, fragments, context, error boundaries, suspense, children helpers, and hook APIs; it also exposes a React-compatibility version string. [src/jsx/index.ts:6-73; src/jsx/base.ts:459]

The common representation is `JSXNode`, which holds a string tag or function tag, props, optional key, and children. Construction rejects invalid non-function tag names before rendering starts. [src/jsx/base.ts:171-184] A child may be text, a promise of text, a number, a JSX node, nullish or boolean content, or nested child arrays. [src/jsx/base.ts:162-170]

`jsx` attaches explicit children to props, removes `key` from the props passed to rendering, delegates node selection to `jsxFn`, and stores the key on the resulting node. [src/jsx/base.ts:327-339] `jsxFn` distinguishes function components, special intrinsic-element component functions, namespace roots, and ordinary HTML tags. [src/jsx/base.ts:342-373] The same JSX tree can therefore enter a string-rendering path through `JSXNode.toString()` or a browser-rendering path through `createRoot`, `render`, or `hydrateRoot`. [src/jsx/base.ts:196-207; src/jsx/dom/client.ts:23-65; src/jsx/dom/render.ts:783-799]

How it works

Server HTML rendering

A `JSXNode.toString()` call creates a callback-aware string buffer and runs serialization inside `runWithRenderContext`. A synchronous buffer returns directly; a buffer containing deferred work goes through `stringBufferToString`. [src/jsx/base.ts:196-207] Element

serialization writes the opening tag, normalizes intrinsic keys, ignores invalid attribute names, escapes text attribute values, converts style objects into CSS declarations, and writes allowed true boolean attributes as empty attributes. [src/jsx/base.ts:209-249]

The serializer rejects simultaneous children and `dangerouslySetInnerHTML`, replaces children with raw `__html` when that property is used alone, and rejects function-valued props unless their name begins with `on` or is `ref`. Event handlers and refs are ignored in server component output rather than emitted as attributes. [src/jsx/base.ts:250-267] Void tags without children close immediately; other tags serialize their children and emit a closing tag. [src/jsx/base.ts:271-280]

Child processing escapes strings, omits booleans and nullish values, appends numbers, recursively descends into arrays, preserves callback lists on already escaped HTML, and inserts promises into the buffer for later resolution. [src/jsx/base.ts:135-159] A function component receives copied props with children injected, then its return value is handled as omitted content, deferred output, a nested JSX node, a child array, escaped HTML, a number, or escaped text. [src/jsx/base.ts:284-317]

Async function-component results that resolve to nodes or child arrays are rendered under a captured render context when contexts exist, so deferred subtree serialization resumes with the values observed before suspension. [src/jsx/base.ts:116-133; src/jsx/base.ts:298-304] `Fragment` is a node whose buffer serialization writes only its children. [src/jsx/base.ts:321-324; src/jsx/base.ts:421-434]

The `hono/jsx/server` compatibility surface offers string and readable-stream renderers. `renderToString` invokes the element's string conversion and throws when that conversion is asynchronous. [src/jsx/dom/server.ts:21-29] `renderToReadableStream` converts non-object content to text and delegates the result plus its error callback to the JSX streaming implementation. [src/jsx/dom/server.ts:50-63]

The streaming renderer first resolves callbacks scheduled before streaming, enqueues the initial encoded HTML, then executes stream-phase callbacks and enqueues their resolved output. Promise failures in deferred callbacks are logged, passed to `onError`, and replaced with empty output; failures in stream startup are also passed to `onError` before the stream closes unless cancelled. [src/jsx/streaming.ts:146-219]

`Suspense` examines child results while retaining render context for a thrown promise. When work is pending, it writes a template marker and fallback initially, then emits a

template-and-script replacement payload once the child results settle. [src/jsx/streaming.ts:42-85; src/jsx/streaming.ts:87-136] `StreamingContext` carries an optional script nonce, and suspense uses it on generated replacement scripts. [src/jsx/streaming.ts:18-32; src/jsx/streaming.ts:50; src/jsx/streaming.ts:105-117]

The experimental server `ErrorBoundary` renders fallback content for synchronous child failures or rejected deferred work. For streamed deferred content, it installs marker replacement callbacks and emits a replacement script when an error arrives after the initial output. [src/jsx/components.ts:50-125; src/jsx/components.ts:128-180]

Context during rendering

`createContext` creates a callable provider with a default-value stack, aliases `Provider` to that callable context, records the context globally, and attaches a DOM-specific provider renderer through a symbol. [src/jsx/context.ts:213-250] On the server, a provider pushes its value, stringifies its child or fragment, then pops the value on completion, error, or promise settlement. [src/jsx/context.ts:215-240] `useContext` reads the top value for the current render store, or the default if no value was installed in that store. [src/jsx/context.ts:98-107; src/jsx/context.ts:253-261]

Render stores use `AsyncLocalStorage` when the runtime can load `node:async_hooks` through supported process APIs. [src/jsx/context.ts:34-67] With that storage, `runWithRenderContext` installs a fresh weak-map store and propagation continues across awaits. [src/jsx/context.ts:161-172] Without it, the fallback store exists only during synchronous work; an async context read after an await falls back to the default value and emits a warning once while fallback async renders are in flight. [src/jsx/context.ts:73-80; src/jsx/context.ts:174-188]

Browser rendering and updates

`createRoot` returns `render` and `unmount` methods for an element or document fragment. Its initial render builds a wrapper component whose state contains the JSX tree; subsequent `render` calls update that state, while calls after `unmount` throw. [src/jsx/dom/client.ts:23-65] `hydrateRoot` creates that root and immediately calls `render`, so hydration follows the same behavior as rendering in this implementation. [src/jsx/dom/client.ts:68-84]

The DOM renderer transforms primitive children into text nodes and attaches hook storage to function nodes. It also wraps `svg` and `math` descendants in a namespace context with the corresponding namespace URI. [src/jsx/dom/render.ts:667-703]

Building a function node selects its DOM-specific renderer symbol when present, merges default props when defined, and calls the selected function while the node sits on `buildDataStack`. [src/jsx/dom/render.ts:274-290]

During reconciliation, old children are matched by text status, or by key and tag when keyed, or by tag otherwise. Matched function components can skip rebuilding when their memo comparator reports equal props and their captured context values are unchanged. [src/jsx/dom/render.ts:541-583] `memo` attaches that comparator and the source component through renderer symbols, with shallow prop comparison as the default. [src/jsx/base.ts:376-418]

Applying nodes creates text nodes or namespaced/plain elements, applies props, recurses into children, positions elements in the container, and then runs insertion effects, layout effects, and scheduled effects in that order. [src/jsx/dom/render.ts:387-480] Property application attaches and removes event listeners, writes `dangerouslySetInnerHTML`, invokes callback or object refs with cleanup handling, updates style strings or objects, and handles form value, checked, selected, and ordinary attributes separately. [src/jsx/dom/render.ts:164-272]

`useState` stores each hook's state by call position in the current function node's stash. A changed state value schedules a renderer update; outside a build stack, it returns the initial value and a no-op setter. [src/jsx/hooks/index.ts:182-243] `use` caches fulfilled or rejected promise outcomes and throws an unresolved promise, allowing DOM `Suspense` to register a retry after settlement and return its fallback. [src/jsx/hooks/index.ts:336-350; src/jsx/dom/components.ts:25-38] DOM error boundaries rethrow promises, but invoke `onError` and return fallback content for non-promise errors. [src/jsx/dom/components.ts:7-23]

Configuration

This part does not read an application environment variable or configuration file in the examined paths. Its runtime-sensitive setting is availability of `AsyncLocalStorage`, discovered through process APIs; the fallback behavior is a synchronous-only context store and a warning for post-await context reads. [src/jsx/context.ts:34-80; src/jsx/context.ts:161-188]

The server compatibility options declare familiar fields, but string rendering warns whenever any options are passed. [src/jsx/dom/server.ts:11-29] Readable-stream rendering accepts an error callback; it warns when passed any option other than `onError`. [src/jsx/dom/server.ts:32-62] Client root options are accepted as a generic

record but currently only cause a warning when nonempty. [src/jsx/dom/client.ts:15-35]

CSS context creation takes a required style-element identifier and optional class-name slug and invalid-slug callback. [src/jsx/dom/css.ts:171-188] In the browser, the resulting CSS object locates `style#<id>`, throws asynchronously if that stylesheet cannot be found, and inserts a rule only when its class name has not already been added. [src/jsx/dom/css.ts:77-113] On the server-side CSS helper path, the same identifier is used to locate or append generated CSS to the style element, and an optional nonce is retained for generated append scripts. [src/helper/css/index.ts:72-86; src/helper/css/index.ts:88-121; src/helper/css/index.ts:192-206]

Wiring

The server boundary is the base node serializer plus HTML helpers for raw escaped values, escaping buffers, callback resolution, and buffer-to-string conversion. [src/jsx/base.ts:1-14; src/jsx/base.ts:196-207] Streaming builds on those escaped HTML callbacks and calls `childrenToString`, context capture, DOM suspense rendering metadata, and DOM build-stack support for promise suspension. [src/jsx/streaming.ts:6-16]

The browser boundary begins at `src/jsx/dom`: its client root calls the DOM renderer, while the renderer consumes shared base node types, child normalization, context, hooks, and symbol contracts. [src/jsx/dom/client.ts:6-9; src/jsx/dom/render.ts:1-16] Symbols isolate cross-layer renderer metadata: DOM renderer overrides, error handlers, hook stashes, internal tags, memo comparators, and form-action permalinks. [src/jsx/constants.ts:1-6]

Intrinsic metadata elements are intercepted by `jsxFn` through the intrinsic tag map. [src/jsx/base.ts:342-359] Their server path records emitted title, script, style, link, and meta markup against the rendering context, removes duplicates according to per-tag keys, and inserts retained markup before `</head>` when that closing tag is present. [src/jsx/intrinsic-element/components.ts:17-91; src/jsx/intrinsic-element/common.ts:3-23] The ordinary element path remains `JSXNode` serialization, which is why metadata behavior is conditional rather than global. [src/jsx/intrinsic-element/components.ts:93-119; src/jsx/base.ts:209-280]

The CSS helper outside this directory depends directly on JSX DOM symbols and CSS DOM objects: it attaches a DOM renderer to its `Style` component so browser rendering can create the style element while server rendering uses HTML callbacks to collect rules. [src/helper/css/index.ts:6-10; src/helper/css/index.ts:81-84;

src/helper/css/index.ts:192-206] This is the practical boundary: JSX owns tree construction, string rendering, context, streaming, and DOM reconciliation; CSS generation and HTML escaping remain in their respective helper modules. [src/jsx/base.ts:1-24; src/jsx/dom/render.ts:1-16; src/helper/css/index.ts:6-28]

147 entities in `src/jsx`. **2 other subsystems depend on it**, which makes it the 3rd most depended-upon part of this codebase.

What it is made of

Its 147 entities sit in 24 files under `src/jsx`: 87 functions, 31 type aliases, 22 constants, 5 interfaces and 2 more.
`index.ts` holds 26 of them — more than any other file here.

Where work enters

- `RenderToStringOptions` — `src/jsx/dom/server.ts`:11
- `RenderToReadableStreamOptions` — `src/jsx/dom/server.ts`:32
- `Props` — `src/jsx/base.ts`:27
- `DOMAttributes` — `src/jsx/base.ts`:38

How it refuses and fails

1 of its components records a refusal or a failure handler.
It refuses work outright, under a condition written into the component itself.

Boundaries

2 other subsystems depend on this one — `Middleware`, `Helper`. Changing what it exposes changes them.

Those 2 hold 3 edges between them, unevenly: `Helper` reaches in across 2 edges, while another holds one. 27 edges leave it against 3 arriving — it reads more of this repository than this repository reads of it. What they reach is narrower than the folder: 3 of its 147 members carry every inbound edge — `renderToReadableStream` (1), `DOM_RENDERER` (1) and `createCssJsxDomObjects` (1). Of the 27 it sends out, 18 go to `Helper` — more than to any other.

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

How this code is named

These conventions cover most of the codebase. Learning them is faster than reading an index —
each one lets you find any member of its family without looking it up.

Pattern	Where	Count	Examples
`use*`	exported symbols	22	`useId`, `useRef`, `useMemo`, `useState`
`create*`	exported symbols	8	`createRef`, `createRoot`, `createPortal`, `createContext`
`jsx*`	exported symbols	5	`jsxFn`, `jsxDEV`, `jsxAttr`, `jsxEscape`