The Pragmatic Programmers

Pragmatic exPress

Server-Driven Web Apps with htmx

Any Language,
Less Code,
Simpler Code

R. Mark Volkmann

edited by Don N. Hagist

CHAPTER 4

Recipes for Common Scenarios

You now know all about the options for getting data into and out of endpoints for htmx applications. But there are many patterns we've yet to explore.

Newcomers to htmx sometimes wonder if features they know how to implement using other web frameworks can be easily implemented using htmx. The good news is that I personally have not yet encountered any feature that I couldn't implemented using htmx.

In this chapter we explore a number of web app features and share htmx solutions in cookbook style.

Inherited htmx Attributes

Many htmx attributes are inherited by descendant elements, meaning they take on the same value for the attribute. Check the official htmx reference at https://htmx.org/reference/ for details.

For example, the documentation for the `hx-boost` attribute (described next) says "hx-boost is inherited and can be placed on a parent element".

In the htmx documentation, whenever you see the term "parent", it really means "ancestor". Likewise, "child" means "descendant".

Boosting

For multi-page web apps, you can improve the performance of loading new pages by adding `hx-boost="true"` to the elements that load them. This can be applied to `a` (anchor) and `form` elements (or their submit buttons). It only works for pages at the same domain as the web app.

Boosting uses an AJAX request to obtain the content of the target page. The contents of the target page `body` element replace the content of the current body element. The only element inside the target page head element that is processed is the title element. The link elements (typically used to load CSS files) and script elements (typically used to load JavaScript code) are not processed, so boosting is only useful when all the CSS and JavaScript needed by the target page has already been loaded by the current page.

When applied to an anchor tag, history is pushed and the URL in the browser address bar is updated. This enables using the browser back button to return to the previous page.

Let's look at a simple example that demonstrates the effect of boosting an anchor element. Here is the main page of the web app, containing two anchor tags. The first does not use hx-boost and the second does.

```
<html>
  <head>
    <title>hx-boost Demo</title>
    <link rel="stylesheet" href="styles.css" />
    <script src="https://unpkg.com/htmx.org@1.9.11"></script>
  </head>
  <body>
    <a href="another.html">Without boost</a>
    <a href="another.html" hx-boost="true">With boost</a>
  </body>
</html>
```

When this page is loaded, the link and script tags are processed. The background becomes light blue (see below) and the htmx library is loaded.

The file styles.css that is loaded by the main page contains the following CSS rule:

```
body {
  background-color: lightblue;
  font-family: sans-serif;
}
```

Here is the file another.html that is referenced by both anchor tags. Note that the head element contains link and script elements.

```
<html>
  <head>
    <title>Another Page</title>
    <link rel="stylesheet" href="another.css" />
    <script src="another.js"></script>
  </head>
  <body>
    <h1>Another Page</h1>
  </body>
```

```
</html>
```

The file another.css that is referenced by another.html contains the following CSS rule:

```
body {
  background-color: red;
}
```

Here is the file another.js that is referenced by another.html.

```
window.onload = () => {
  alert('another.js was loaded.');
};
```

When the “Without boost” link on the main page is clicked, the another.html page is loaded in the normal way. The link and style tags are processed, so the alert in another.js is displayed and the background changes to red.

When the With boost link on the main page is clicked, the another.html page is loaded, but the link and style tags are not processed. The alert is not displayed and the background remains light blue.

Lazy Loading

When displaying content that is expensive to acquire, it is useful to delay requesting it until the rest of the page has loaded or until the part of the page that will display it scrolls into view.

To wait to send a request until the page has loaded, use hx-trigger="load". To wait until an element is scrolled into view, use hx-trigger="revealed".

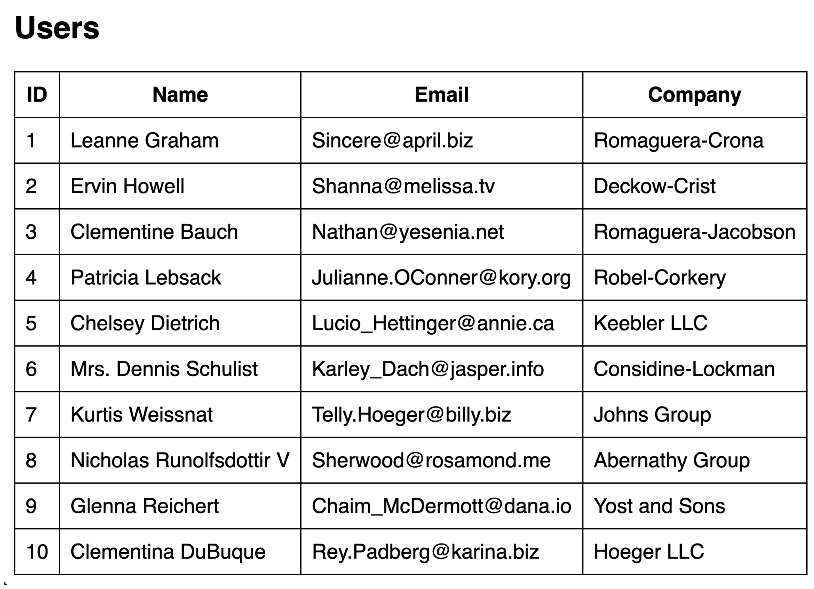
For example:

```
<table hx-get="/weather/forecast" hx-trigger="revealed"></table>
```

The following HTML contains a div element that appears near the bottom of the page so it is out of view when the page is first loaded. It uses hx-trigger="revealed" so a GET request to /users is not sent until the data is needed.

It also uses the hx-indicator attribute to specify an element to display while the request is being processed. The CSS opacity property of the element starts at 0, changes to 1 when the request is sent, and changes back to zero after the response is received. A good choice for the element is a spinner GIF image.

Here’s a screenshot that is produced by the HTML below.



Users

ID	Name	Email	Company
1	Leanne Graham	Sincere@april.biz	Romaguera-Crona
2	Ervin Howell	Shanna@melissa.tv	Deckow-Crist
3	Clementine Bauch	Nathan@yesenia.net	Romaguera-Jacobson
4	Patricia Lebsack	Julianne.OConner@kory.org	Robel-Corkery
5	Chelsey Dietrich	Lucio_Hettinger@annie.ca	Keebler LLC
6	Mrs. Dennis Schulist	Karley_Dach@jasper.info	Considine-Lockman
7	Kurtis Weissnat	Telly.Hoeger@billy.biz	Johns Group
8	Nicholas Runolfsdottir V	Sherwood@rosamond.me	Abernathy Group
9	Glenna Reichert	Chaim_McDermott@dana.io	Yost and Sons
10	Clementina DuBuque	Rey.Padberg@karina.biz	Hoeger LLC

Recipes/lazy-loading.html

```html
<html>
  <head>
    <title>htmx Lazy Loading</title>
    <link rel="stylesheet" href="styles.css" />
    <script src="https://unpkg.com/htmx.org@1.9.11"></script>
  </head>
  <body>
    <!-- Lots of content omitted. -->
    <h2>Users</h2>
    <div
      hx-get="/users"
      hx-indicator=".htmx-indicator"
      hx-trigger="revealed"
    />
    <img alt="loading" class="htmx-indicator" src="/spinner.gif" />
  </body>
</html>
```

The server is defined by the following code. First, we import things we need from the Hono library, define a User type, and specify the URL for getting fake users from the JSONPlaceholder[1] API.

Recipes/lazy-loading.tsx

```tsx
import {type Context, Hono} from 'hono';
import {serveStatic} from 'hono/bun';
```

1. https://jsonplaceholder.typicode.com

```
type User = {
  id: number;
  name: string;
  email: string;
  company: {
    name: string;
  };
};

const URL = 'https://jsonplaceholder.typicode.com/users';
```

Next, we create a Hono server instance and configure it to serve static files from the "public" directory, which includes index.html and styles.css.

Recipes/lazy-loading.tsx

```
const app = new Hono();

app.use('/*', serveStatic({root: './public'}));
```

Finally, we define the GET /users endpoint. This fetches user data and returns it in an HTML table. This table is added as the innerHTML of the div element that triggered the request.

Recipes/lazy-loading.tsx

```
app.get('/users', async (c: Context) => {
  Bun.sleepSync(1000); // simulates long-running query
  const res = await fetch(URL);
  const users = await res.json();
  return c.html(
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Name</th>
          <th>Email</th>
          <th>Company</th>
        </tr>
      </thead>
      <tbody>
        {users.map((user: User) => (
          <tr>
            <td>{user.id}</td>
            <td>{user.name}</td>
            <td>{user.email}</td>
            <td>{user.company.name}</td>
          </tr>
        ))}
      </tbody>
    </table>
  );
});
```

```
export default app;
```

See the working example project at lazy-load[2].

2. https://github.com/mvolkmann/htmx-examples/tree/main/lazy-load