gatsby-plugin-preload-fonts

This plugin preloads all necessary fonts per route to decrease time to first meaningful paint. It works really well with services like Google Fonts that link to stylesheets that in turn link to font files.

Install

```
npm install gatsby-plugin-preload-fonts
```

Once you've installed the plugin, you'll want to add the included script to your project's scripts in `package.json`.

```
{
  "scripts": {
    "preload-fonts": "gatsby-preload-fonts"
  }
}
```

How to use

In your `gatsby-config.js`:

```
module.exports = {
  plugins: [`gatsby-plugin-preload-fonts`],
}
```

Before building your application, you will need to generate a font asset map using the included `gatsby-preload-fonts` script.

```
npm run preload-fonts
```

You'll need to run this script again after adding routes to your application (including new dynamic routes generated by source plugins), as well as any time you add new font assets or stylesheets that link to font assets.

This script outputs a file named `font-preload-cache.json` to the root of your project. You will want to check this file into your project's source control so all contributors have the latest version of the cache.

You may pass any additional args to Puppeteer when running this script, like so:

```
npm run preload-fonts -- --no-sandbox
```

See Puppeteer for more information.

Using with CI

If you're planning to use this plugin in CI you can set the environment variable `CI=true` (most CI providers like Travis and CircleCI will set this for you). Please note that in this case `font-preload-cache.json` will only get rebuilt when routes change.

Explanation of method

A common pattern in modern web development is to link to a remote stylesheet that in turn links to other remote font assets. If it takes too long for the font to resolve/download, typography will either not show up or flash in a different system font before swapping out for the correct font.

This can be avoided by preloading font assets on the initial page load instead of waiting for them to resolve indirectly. In order to do this, `<link rel="preload">` tags will need to be added for each font asset required on each route at build time.

To accomplish this, this plugin runs in two steps:

1. Scrape each application route to determine what font assets are ultimately required.
2. Inject `<link rel="preload">` tags for each asset at build time.

The included `gatsby-preload-fonts` script spins up a puppeteer instance and visits each of your application's routes to determine the required font assets. Once it has done so, it caches this mapping and creates a unique scrape hash to help avoid rescraping the same set of routes in subsequent runs (note that you will still need to re-run the scraper if your application routes don't change but you add new font assets).

Configuration options

`crossOrigin` [string|function][default: `anonymous`]

Whether or not to include the `cross-origin` attribute on injected `<link>` tags. By default, this will be set to `anonymous` (this is usually correct).

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-plugin-preload-fonts`,
      options: {
        crossOrigin: `use-credentials`,
        // OR
        crossOrigin: pathname =>
          pathname.match(/^\/elevated/) ? `use-credentials` : `anonymous`,
      },
    },
  ],
}
```

Troubleshooting

If an asset isn't showing up, this is likely due to its route not being scraped properly. To insure that the route in question is being scraped, you have a couple options. First, routes that don't show up under the `allSitePage` query on your site's GraphQL server will not be scraped.

If you're still having trouble, you can run `gatsby-preload-fonts` with a lower log level to view what paths it's visiting in real time.

```
# mac/linux
LOG_LEVEL=info npm run preload-fonts

# windows
set LOG_LEVEL=info & npm run preload-fonts
```

Available log levels include `info`, `debug`, `warn`, `error`, and `silent`, respectively.

Puppeteer

If you see a huge stack trace that looks like this

```
(node:30511) UnhandledPromiseRejectionWarning: Error: Failed to launch chrome!
[0705/172123.766471:FATAL:zygote_host_impl_linux.cc(116)] No usable sandbox! Update
your kernel or see
https://chromium.googlesource.com/chromium/src/+/master/docs/linux_suid_sandbox_develop
 for more information on developing with the SUID sandbox. If you want to live
dangerously and need an immediate workaround, you can try using --no-sandbox.
...
```

you can pass the `--no-sandbox` flag to Puppeteer when you run the script:

```
npm run preload-fonts -- --no-sandbox
```

This comes with inherent security risks, but you should be alright since you're only running it locally.

Use different Chrome/Chromium executable

In some cases, you might have to point Puppeteer to an external installation of Chrome/Chromium (e.g., on Alpine Linux, the build-in version of Chromium does not work). You can set the `PUPPETEER_EXECUTABLE_PATH` environment variable to the path of your custom chromium installation. A list with all environment variables to configure Puppeteer can be found at the official docs.