Using pagination in the REST API

In this article

Learn how to navigate through paginated responses from the REST API.

About pagination

When a response from the REST API would include many results, GitHub will paginate the results and return a subset of the results. For example, `GET /repos/octocat/Spoon-Knife/issues` will only return 30 issues from the `octocat/Spoon-Knife` repository even though the repository includes over 1600 open issues. This makes the response easier to handle for servers and for people.

You can use the `link` header from the response to request additional pages of data. If an endpoint supports the `per_page` query parameter, you can control how many results are returned on a page.

This article demonstrates how to request additional pages of results for paginated responses, how to change the number of results returned on each page, and how to write a script to fetch multiple pages of results.

Using `link` headers

When a response is paginated, the response headers will include a `link` header. If the endpoint does not support pagination, or if all results fit on a single page, the `link` header will be omitted.

The `link` header contains URLs that you can use to fetch additional pages of results. For example, the previous, next, first, and last page of results.

To see the response headers for a particular endpoint, you can use curl, GitHub CLI, or a library you're using to make requests. To see the response headers if you are using a library to make requests, follow the documentation for that library. To see the response headers if you are using curl or GitHub CLI, pass the `--include` flag with your request. For example:

```
curl --include --request GET \
--url "https://api.github.com/repos/octocat/Spoon-Knife/issues" \
--header "Accept: application/vnd.github+json"
```

If the response is paginated, the `link` header will look something like this:

```
link: <https://api.github.com/repositories/1300192/issues?page=2>; rel="prev",
```

```
<https://api.github.com/repositories/1300192/issues?page=4>; rel="next",
<https://api.github.com/repositories/1300192/issues?page=515>; rel="last",
<https://api.github.com/repositories/1300192/issues?page=1>; rel="first"
```

The `link` header provides the URL for the previous, next, first, and last page of results:

- The URL for the previous page is followed by `rel="prev"`.
- The URL for the next page is followed by `rel="next"`.
- The URL for the last page is followed by `rel="last"`.
- The URL for the first page is followed by `rel="first"`.

In some cases, only a subset of these links are available. For example, the link to the previous page won't be included if you are on the first page of results, and the link to the last page won't be included if it can't be calculated.

You can use the URLs from the `link` header to request another page of results. For example, to request the last page of results based on the previous example:

```
curl --include --request GET \
--url "https://api.github.com/repositories/1300192/issues?page=515" \
--header "Accept: application/vnd.github+json"
```

The URLs in the `link` header use query parameters to indicate which page of results to return. The query parameters in the `link` URLs may differ between endpoints, however each paginated endpoint will use the `page`, `before` / `after`, or `since` query parameters. (Some endpoints use the `since` parameter for something other than pagination.) In all cases, you can use the URLs in the `link` header to fetch additional pages of results. For more information about query parameters see "Getting started with the REST API."

Changing the number of items per page

If an endpoint supports the `per_page` query parameter, then you can control how many results are returned on a page. For more information about query parameters see "Getting started with the REST API."

For example, this request uses the `per_page` query parameter to return two items per page:

```
curl --include --request GET \
--url "https://api.github.com/repos/octocat/Spoon-Knife/issues?per_page=2" \
--header "Accept: application/vnd.github+json"
```

The `per_page` parameter will automatically be included in the `link` header. For example:

```
link: <https://api.github.com/repositories/1300192/issues?per_page=2&page=2>;
rel="next", <https://api.github.com/repositories/1300192/issues?
per_page=2&page=7715>; rel="last"
```

Scripting with pagination

Instead of manually copying URLs from the `link` header, you can write a script to fetch multiple pages of results.

The following examples use JavaScript and GitHub's Octokit.js library. For more information about Octokit.js, see "Getting started with the REST API" and the Octokit.js README.

Example using the Octokit.js pagination method

To fetch paginated results with Octokit.js, you can use `octokit.paginate()`. `octokit.paginate()` will fetch the next page of results until it reaches the last page and then return all of the results as a single array. A few endpoints return paginated results as array in an object, as opposed to returning the paginated results as an array. `octokit.paginate()` always returns an array of items even if the raw result was an object.

For example, this script gets all of the issues from the `octocat/Spoon-Knife` repository. Although it requests 100 issues at a time, the function won't return until the last page of data is reached.

JavaScript

```
import { Octokit } from "octokit";

const octokit = new Octokit({ });

const data = await octokit.paginate("GET /repos/{owner}/{repo}/issues", {
  owner: "octocat",
  repo: "Spoon-Knife",
  per_page: 100,
  headers: {
    "X-GitHub-Api-Version": "2022-11-28",
  },
});

console.log(data)
```

You can pass an optional map function to `octokit.paginate()` to end pagination before the last page is reached or to reduce memory usage by keeping only a subset of the response. You can also use `octokit.paginate.iterator()` to iterate through a single page at a time instead of requesting every page. For more information, see the Octokit.js documentation.

Example creating a pagination method

If you are using another language or library that doesn't have a pagination method, you can build your own pagination method. This example still uses the Octokit.js library to make requests, but does not rely on `octokit.paginate()`.

The `getPaginatedData` function makes a request to an endpoint with `octokit.request()`. The data from the response is processed by `parseData`, which handles cases where no data is returned or cases where the data that is returned is an object instead of an array. The processed data is then appended to a list that contains all of the paginated data collected so far. If the response includes a `link` header and if the `link` header includes a link for the next page, then the function uses a RegEx pattern (`nextPattern`) to get the URL for the next page. The function then repeats the previous steps, now using this new URL. Once the `link` header no longer includes a link to the next page, all of the results are returned.

JavaScript

```
import { Octokit } from "octokit";

const octokit = new Octokit({ });

async function getPaginatedData(url) {
  const nextPattern = /(?<=<)([\S]*)(?=>; rel="Next")/i;
  let pagesRemaining = true;
```

```
  let data = [];

  while (pagesRemaining) {
    const response = await octokit.request(`GET ${url}`, {
      per_page: 100,
      headers: {
        "X-GitHub-Api-Version":
          "2022-11-28",
      },
    });

    const parsedData = parseData(response.data)
    data = [...data, ...parsedData];

    const linkHeader = response.headers.link;

    pagesRemaining = linkHeader && linkHeader.includes(`rel=\"next\"`);

    if (pagesRemaining) {
      url = linkHeader.match(nextPattern)[0];
    }
  }

  return data;
}

function parseData(data) {
  // If the data is an array, return that
    if (Array.isArray(data)) {
      return data
    }

  // Some endpoints respond with 204 No Content instead of empty array
  //   when there is no data. In that case, return an empty array.
  if (!data) {
    return []
  }

  // Otherwise, the array of items that we want is in an object
  // Delete keys that don't include the array of items
  delete data.incomplete_results;
  delete data.repository_selection;
  delete data.total_count;
  // Pull out the array of items
  const namespaceKey = Object.keys(data)[0];
  data = data[namespaceKey];

  return data;
}

const data = await getPaginatedData("/repos/octocat/Spoon-Knife/issues");

console.log(data);
```

Legal

© 2023 GitHub, Inc. Terms Privacy Status Pricing Expert services Blog