Easy URL Manipulation with URLSearchParams

By Eric Bidelman

Engineer @ Google working on web tooling: Headless Chrome, Puppeteer, Lighthouse

The **`URLSearchParams`** API provides a consistent interface to the bits and pieces of the URL and allows trivial manipulation of the query string (that stuff after "**`?`**").

Traditionally, developers use regexs and string splitting to pull out query parameters from the URL. If we're all honest with ourselves, that's no fun. It can be tedious and error prone to get right. One of my dark secrets is that I've reused the same **`get`**|**`set`**|**`removeURLParameter`** helper methods in several large Google.com app, including Google Santa Tracker and the Google I/O 2015 web.

It's time for a proper API that does this stuff for us!

URLSearchParams API

Try the demo

Chrome 49 implements **`URLSearchParams`** from the URL spec, an API which is useful for fiddling around with URL query parameters. I think of **`URLSearchParams`** as an equivalent convenience to URLs as **`FormData`** was to forms.

So what can you do with it? Given a URL string, you can easily extract parameter values:

```
// Can also constructor from another URLSearchParams
const params = new URLSearchParams('q=search+string&version=1&person=Eric');

params.get('q') === "search string"
params.get('version') === "1"
Array.from(params).length === 3
```

Note: If there are several values for a param, **`get`** returns the first value. **iterate** over parameters:

```
for (let p of params) {
  console.log(p);
}
```

set a parameter value:

```
params.set('version', 2);
```

Note: If there are several values, `set` removes all other parameters with the same name.

append another value for an existing parameter:

```
params.append('person', 'Tim');
params.getAll('person') === ['Eric', 'Tim']
```

delete a parameter(s):

```
params.delete('person');
```

Note: this example removes all `person` query parameters from the URL, not just the first occurrence.

Working with URLs

Most of the time, you'll probably be working with full URLs or modifying your app's URL. The `URL` constructor can be particularly handy for these cases:

```
const url = new URL('https://example.com?foo=1&bar=2');
const params = new URLSearchParams(url.search);
params.set('baz', 3);

params.has('baz') === true
params.toString() === 'foo=1&bar=2&baz=3'
```

To make actual changes to the URL, you can grab parameters, update their values, then use `history.replaceState` to update the URL.

```
// URL: https://example.com?version=1.0
const params = new URLSearchParams(location.search);
params.set('version', 2.0);
```

```
window.history.replaceState({}, '', `${location.pathname}?${params}`);
// URL: https://example.com?version=2.0
```

Here, I've used ES6 template strings to reconstruct an updated URL from the app's existing URL path and the modified params.

Integration with other places URLs are used

By default, sending `FormData` in a `fetch()` API request creates a multipart body. If you need it, `URLSearchParams` provides an alternative mechanism to POST data that's urlencoded rather than mime multipart.

```
const params = new URLSearchParams();
params.append('api_key', '1234567890');

fetch('https://example.com/api', {
  method: 'POST',
  body: params
}).then(...)
```

Although it's not yet implemented in Chrome, `URLSearchParams` also integrates with the `URL` constructor and `a` tags. Both support our new buddy by providing a read-only property, `.searchParams` for accessing query params:

```
const url = new URL(location);
const foo = url.searchParams.get('foo') || 'somedefault';
```

Links also get a `.searchParams` property:

```
const a = document.createElement('a');
a.href = 'https://example.com?filter=api';

// a.searchParams.get('filter') === 'api';
```

Feature detection and browser support

Currently, Chrome 49, Firefox 44, and Opera 36 support `URLSearchParams`.

```
if ('URLSearchParams' in window) {
  // Browser supports URLSearchParams
```

```
}
```

For polyfills, I recommend the one at github.com/WebReflection/url-search-params.

Demo

Try out the sample!

To see `URLSearchParams` in a real-world app, check out Polymer's material design Iconset Generator. I used it to setup the app's initial state from a deep link. Pretty handy :)

Last updated July 2, 2018.