Feature Policies

Some of these policies may be under Origin Trials or may not be available in browsers at the time the article was written. Check Chromestatus for a list of policies that are either active or under consideration.

Feature policies are a way to restrict what web APIs we make available to what web context (page or `iframe`) thus reducing the risk of malicious third-party code, and `foot`guns (shooting ourselves in the foot) by misusing APIs or using too many of them.

With Feature Policy, you opt-in to a set of “policies” for the browser to enforce on specific features used throughout your site. These policies restrict what APIs the site can access or modify the browser’s default behavior for certain features.

Policies are a contract between developer and browser. They tell the browser about what out intent as developers is and keeps us honest when our app tries to go off the rails and do something we’re not allowed to. If the site or embedded third-party content violates any policies, the browser overrides the behavior with better UX or blocks the API altogether.

The full set of APIs we can restrict with feature policies is listed below. For more information about the policies, browser support, and discussion of how they work (for the feature policies that have been implemented) see featurepolicy.info

- accelerometer
- ambient-light-sensor
- autoplay
- camera
- document-domain
- document-write
- encrypted-media
- font-display-late-swap
- fullscreen
- geolocation
- gyroscope
- layout-animations
- lazyload

- legacy-image-formats
- magnetometer
- microphone
- midi
- oversized-images
- payment
- picture-in-picture
- speaker
- sync-script
- sync-xhr
- unoptimized-images
- unsized-media
- usb
- vertical-scroll
- vr
- wake-lock

Set headers on the server

The best and best way to set up feature policies globally is to set the headers on your server configuration.

These examples use image-related feature policies to tighten the type of images that get served to your users.

Apache

Apache 2.4.7 and later allow you to set headers only if they are empty. These are normal headers that can be set everywhere Apache would normally allow you to do so.

```
<Location />
  Header setifempty Feature-policy
    unsized-media 'none';
    oversized-images 'self'(2.0) *(inf);
    unoptimized-lossy-images 'self'(1) *(inf);
    unoptimized-lossless-images 'self'(1) *(inf);
    unoptimized-lossless-images-strict 'self'(1) *(inf);
```

```
</Location>
```

Nginx

In Nginx I've added multiple policies in the same header. The result is the same

```
location / {
  add_header Feature-Policy "unsized-media 'none';
    unoptimized-lossy-images 'self'(0.5) *(inf);
    unoptimized-lossless-images 'self'(1) *(inf)
    unoptimized-lossless-images-strict 'self'(1) *(inf);"
}
```

Using Feature Policy in iframes

Another way we can use feature poolicy for our content is inside the `allow` attribute of an iframe element. The example below, taken from Youtube, shows how it works.

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/ht_HDdtyy9s"
frameborder="0"
allow="accelerometer; fullscreen; autoplay;
encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
```

The `allow` attribute contains a list of all the feature policies that are allowed for that specific iframe.

To make sure we can handle older browsers, we have to add the old-style attribute, in the example we use `allow` to set up the full screen feature policy for the iframe. We also use the older `allowfullscreen` standalone attribute for older browsers that don't support the Feature Policies or where it hasn't been implemented.

If both the feature policy and the equivalent attribute are present and the values conflict, the more restrictive of the two will win.

Javascript API

To help in our code we can use `document.featurePolicy` to query what Policy features are available, whether a page allows a given feature, whether an origin is allowed to use a feature available through policy and what origins has the page allowed to use a feature through a feaature policy.

```
// Lists feature policies allowed by the page.
document.featurePolicy.allowedFeatures();

// True if the page allows the feature.
document.featurePolicy.allowsFeature('geolocation');

'geolocation'// True if the origin allows the feature.turePolicy.allowsFe

// List of fe'geolocation'// List of feature policies allowed
// by the browser regardless if they're active
document.featurePolicy.features();

// Lists origins on the page allowed
// to use the featuretAllowlistForFeature('geolocation');
'geolocation'
```

The idea is that we can use these methods to tailor the code based on what features are allowed or not.

The example below checks if the client supports geolocation and if the feature is allowed for the page it’s hosted on.

```
if (("geolocation" in navigator) &&  document.featurePolicy.allowsFeature(
  console.log('Geolocation supported and allowed');
} else {
  console.log('Geolocation not supported or not allowed');
}
```

We could get more detailed information by nesting the tests to know if it's not supported or not allowed but, for most cases, one test is enough.

Use Cases

So, we know how Feature Policy works but why would we use them?

I can think of two use cases where having featur policies will help improve application performance.

Image Performance

The first casse is to guard against image bloat and image-caused text jump. Using feature policies, we can ensure that all images on the page have height and width attributes set and that we don't send images to clients that are too large to display and will take too long to load.

The example below will only work with Apache 2.4.7 where the `setifempty` header directive was introduced.

The feature policy directives that we use in this case are:

- **oversized-images**: the number of pixels of a container determines the resolution of an image served inside. It is unnecessary to use an image that is much larger than what the viewing device can actually render. The example will trigger and block images that are twice as large as their dimensions

- **unsized-media**: enforces explicit dimensions for images and videos. If dimensions aren't specified on the element, the browser sets a default size of 300x150 when this policy is active

- **unoptimized-lossy-images**: requires the data size (in bytes) of images using lossy compression to be no more than X times bigger than its rendering area (in pixels). If the images is larger than the desired size, the browser will render a placceholder instead

  A lossy `<img>` element should not exceed a byte-per-pixel ratio of X, with a fixed 1KB overhead allowance. For a W x H image, the file size threshold is calculated as W x H x X + 1024 (where X is specified in the policy). Any image whose file size exeeds the limit will be blocked.

- **unoptimized-lossless-images**: requires the data size (in bytes) of images using lossless compression to be no more than X times bigger than its rendering area (in pixels). If the images is larger than the desired size, the browser will render a placeholder instead

  A lossless <img> element should not exceed a byte-per-pixel ratio of X, with a fixed 1KB overhead allowance. For a W x H image, the file size threshold is calculated as W x H x X + 1024 (where X is specified in the policy). Any image whose file size exeeds the limit will be blocked.

```
<Location />
  Header setifempty Feature-Policy
    unsized-media 'none';
    oversized-images 'self'(2) * (inf);
    unoptimized-lossy-images 'self'(0.5) *(inf);
    unoptimized-lossless-images 'self'(1) *(inf)
</Location>
```

These policies will also help keep me honest in case I forget to resize or compress images. They will also keep individual developers and the design teams honest by not rendering images that don't match the criteria that has, hopefully, been agreed upon.

Third Party Privacy

Another aspect of feature policy that I find intriguing is using them to control what browser and computer features sites have access to.

This set of feature questies disable access to the features listed. Some are user-facing privacy considerations like not granting access to camera, microphone or geolocation.

Others have to do with with older web features that have security implications, like being able to programmatically write content to the page and wipe existing content.

```
<Location />
  Header setifempty Feature-Policy
    geolocation 'none';
    camera 'none';
    microphone 'none';
    usb 'none';
    document-domain 'none';
    document-write 'none'
</Location>
```

This should disable the features for all sites, including our own.

If we want to disable third party access but retain the ability to use the features on our own sites we can change none to `self`.

To grant access to third party sites you can either replace none with the URL of the sites or sites that you want to give permissioon to or add the URL after `self` if you’ve given permission to your site.

Conclusion

Feature policy offers an interesting way to keep ourselves honest and limit the damage a rogue site can do to our users.

Support is spotty and uneven, please check https://featurepolicy.info/ and caniuse.com Feature Policy entry for up-to-date information about the policy directives and browser support.

Links and Resources

- Introducition to Feature Policy
- Feature Policy & the Well-Lit Path for Web Development (Chrome Dev Summit 2018)
- Feature Policy (MDN)
- Using Feature Policy (MDN)
- Feature Policy Demos
- Feature Policy Tester