Web Bundles

Felipe Erias, Dan Ehrenberg

Igalia, in partnership with eyeo

WPACK - IETF 111

July 2021

Index

Proposed spec changes

Overview

- Goal: base Web Bundle spec as simple as possible
- Focus on consensus functionality
  - use cases like resource preloading
- Allow for extensions
  - use cases that require a more complex bundle format can be addressed in other documents

Remove the primary URL

wpack-wg/bundled-responses/pull/5

- Browsing inside a bundle requires a primary URL specified in the bundle
- Primary URL could be provided in a separate section
- However, it is outside the scope of the core bundle specification to define this section

Remove manifest section

wpack-wg/bundled-responses/pull/6

- As previous, outside the scope of the core spec
- Perhaps redundant?
  - primary URL response could already include it

```
Content-Security-Policy: manifest-src <source>;
```

Remove content negotiation

wpack-wg/bundled-responses/pull/7

- More efficient to do it on the server to select which bundle to send, rather than including several variants in the bundle and doing it on the client
- Related discussions:
    - **bundled-responses/issues/2**
    - **WICG/webpackage/pull/618**

Add bundle-format.md

wpack-wg/bundled-responses/pull/8

- A summary of the specification, providing a formal description of the format of the bundle
  - no new information
- This PR assumes that all the others are accepted, so we can see their collective result

Updated bundle format

Name	Description
`magic`	`F0 9F 8C 90 F0 9F 93 A6` (🌐📦)
`version`	spec version, no change except as last resort
`section-lengths`	array of `section-length`
`sections`	array of sections
`length`	length of the bundle in bytes

Updated bundle format

`section-length`	*(updated)*
`section-name`	name of the section
`length`	length of the section in bytes

Sections ✏️	
`index`	map from a URL to a `location-in-responses`
`critical`	*(optional)* sections that the client needs to understand
`responses`	array: HTTP responses for the bundle's representations

Updated bundle format

`location-in-responses`	*(updated)*
`offset`	Offset within the `responses` section
`length`	Size of the response in bytes

`response`	
`headers`	map of field names to field values
`payload`	content of the HTTP response as a byte string

Second document

wpack-wg/bundled-responses/issues/9

- Proposal: WPACK group creates a second document:
  - an extension focused on *"browsing into a bundle"*
- Additional sections may be specified in `"critical"`
- Goals:
  - simple, consensus base spec
  - extension spec: less general use cases that require a more complex bundle format

Discussion

- Pull requests
  - Remove the primary URL section
  - Remove the manifest section
  - Remove content negotiation
  - Add bundle-format.md
- Issue/proposal
  - Extension document for browsing into a bundle

Bundling and resource preloading

Proposal: WICG/resource-bundles

References

- Igalia: **"Bundle Preloading"**
  - **WICG/resource-bundles**
- Chromium has an experimental implementation of very similar ideas, this document enumerates the small differences between the two:
  - *"Web Bundles and Bundle Preloading"* **link**

Background

- Web sites are composed of multiple resources
  - hundreds or thousands in real-world web apps
  - fetching them one by one has poor performance
- *Bundlers*
  - combine and transform resources for deployment
  - **webpack**, **browserify**, **rollup**, **parcel**, **esbuild**…

Bundlers

- Dependency graph
- *Bundling*
  - combine resources for efficient deployment
- *Tree shaking*
  - identify and remove unused code
- *Virtualization*
  - include other resources inline in JS code
  - fewer network exchanges, but slower to load
- *Code splitting*
  - divide code/resources into various chunks
  - can be loaded on demand or in parallel

Motivations

- Efficiency
  - Retrieving bundled resources is more costly
  - Does not fit well with browsers' caching strategies
- Interoperability
  - Bundling strategies are not standardized
  - Bundling strategies are not interoperable

Goals

- Efficiently distribute web content
  - Web platform: standard, interoperable
- Keep benefits of today's bundler ecosystem
  - improve network performance, load content faster
  - facilitate revving, code splitting, tree shaking, etc.
  - reduce bundler build time and logic
- Keep benefits of accessing individual resources
  - flexibility when loading and processing responses
  - each response can be cached individually

Outline

- **Resource preloading with bundled responses**
  1. *document* provides a list of required resources
  2. *client* sends HTTP request for those not in its cache
  3. *server* replies with a bundled response
- **API components**
  - static API: declare resources to retrieve in a bundle
  - imperative API in JS: preload bundled resources
  - request headers, corresponding response behavior

Preserving

- Resource identity
  - bundles contain same resources as individual URLs
- Origin model
  - resources within the same origin as the bundle
- Path restriction
  - resources have the same `path-1` as the bundle
- URL consistency
  - each URL corresponds to a resource
  - same response for individual and bundled requests
    - can verify that servers are well-behaved
  - graceful degradation, content blocking

Usability goals

- For developers
  - no need to change tools (except bundlers)
  - code splitting, transformations, etc. remain possible
- For final and intermediary servers
  - no large time/space penalties if not supported
  - stateless serving logic
  - deploy static content just by copying files
- For browsers
  - fit current fetching and caching architectures
  - support graceful degradation

Privacy goals

- Personalization
  - do not enable disguising personalized content
  - do not weaken the significance of URLs.
- Content blocking
  - must be compatible with content blocking
  - "trusted" intermediary can not "repackage" sites
  - do not enable cheap rotation of URLs in the bundle
  - do not download blocked content

HTML `<script>` tag

```
<!-- https://www.example.com/index.html -->
<script type="bundlepreload">
    {
        "source": "./assets/resources.wbn",
        "resources": [
            "render.js",
            "profile.png"
        ]
    }
</script>
```

JavaScript API

```
// https://www.example.com/index.html
window.bundlePreload({
    source: "./assets/resources.wbn",
    resources: ["render.js","profile.png"]
});

let image = document.createElement("img");
image.src = "assets/profile.png";
...
```

HTTP request

```
GET /assets/resources.wbn HTTP/1.1
...
Host: www.example.com
Bundle-Preload: "render.js", "profile.png"
...
```

HTTP response

- The response must be a *bundled response* containing HTTP responses for each of the requested URLs:

```
https://www.example.com/assets/render.js
https://www.example.com/assets/profile.png
```

- These resources may be cached and references to them later on may be loaded from the cache

Code splitting

- `example.com/page1.html` lists resources `A`, `B`, `C`
  - client requests all these resources and caches them
- `example.com/page2.html` lists resources `A`, `D`, `E`
  - client only needs to request `D` and `E`
- As the client may request a subset of the listed resources, it is able to retrieve those it already has from its cache

Responsiveness

- Web devs may prioritize some resources to shorten the time until the page is ready for user interaction
  - e.g. declaratively, when the page is loaded
- Less urgent resources may be fetched after the page has completed its first load
  - e.g. imperatively, on `window.onload`

Avoid cachebusting

- Prevent the need to download a whole bundle when only some of its resources have changed
- Devs can update resources with fine granularity:

1. append a version to the resource's name ("revving")
2. update the preload resource list in the document

Summary

- API for **resource preloading with bundled responses**
  - declarative (HTML) and imperative (JavaScript)
  - specific request headers and response behaviour
- Preserves user privacy and freedom
- Backwards compatibility for servers, intermediaries
- Remove overhead from resource "virtualization", etc.
- No significant changes to dev workflow

Bundling and the JS ecosystem

JS Modules today

- JS APIs require modules to be in separate source files:
  - static `import` and dynamic `import()`
  - Web workers
  - worklets (paint, audio, animation, layout)
- This causes several problems:
  - APIs are more inconvenient to use
  - ergonomic solutions are insecure or non-standard
  - importing a source file from another origin affects relative paths, CORS, etc.

Goal of module fragments

- Allow bundling of multiple JS modules in a single file
  - can be used as output format for bundlers
  - also convenient to write for developers
- Complementary to resource bundle preloading
  - number of JS files tends to blow up in web apps
- Complementary to *code splitting*, *tree shaking*, etc.
  - optimize number and composition of JS source files
  - more efficient for chunks to be more granular, to provide exactly the ones needed

Standardization track

- This proposal at TC39:
  - **module fragments** (Stage 1)

Core idea

Declare a module inline, inside another source file:

```
module myModule {
  export function myFunction(...) {
    ...
  }
};
```

- Related proposal: **module blocks** (stage 2)

Importing

These modules can then be imported dynamically:

```
let myExports = await import(myModule);
myExports.myFunction();
```

Or statically (from within other modules):

```
module combined {
  import { myFunction } from myModule;
  ...
}
```

Exporting

Can be exported, so they are accessible from outside:

```
// Only accessible within this file
module priv {
  export function doInPrivate() { ... }
}

// Accessible from outside this file
export module pub {
  import { doInPrivate } from priv;
  export function doInPublic() { ... }
}
```

With resource preloading

- Resource bundle preloading
  - works at the network level
  - supports resources of any MIME type
  - supports additional metadata as HTTP headers
  - each resource can be cached individually
- JS module fragments
  - limited to JS source files
  - work at the JS module loading level
  - only affect how JavaScript is loaded
  - equivalent to what bundlers do today

Wrap up

- High-performance, standard bundling for the Web
  - reduce and reorganize the number of source files
  - distribute these and other resources efficiently
- Next steps:
  - module fragments is at Stage 1 in TC39
  - prototype module fragments together with bundle preloading

Contact

Felipe Erias felipeerias@igalia.com

Daniel Ehrenberg dehrenberg@igalia.com

Thank you!