Créez en JavaScript votre fichier polyglotte

HTML / ZIP / PNG

Introduction

- Développeur Web depuis 20+ ans
- Auteur de :
  - zip.js : bibliothèque pour lire et écrire des fichiers zip
  - SingleFile : extension et outil en ligne de commande pour sauver une page web dans un simple fichier HTML
    - utilisation de Data URIs pour référencer et stocker les contenus binaire
    - taille du fichier de sortie plus large que la taille de toutes les ressources additionnées
- Peut-on combiner SingleFile et zip.js ?
- Peut-on aller plus loin ?

Projet web

1 - Création d'un projet de test (HTML, CSS, JS) dans `/project`

Inclusion de ressources externes :

- `<link href=style.css>`
- `<img src=image.png>`
- `<script src=script.js></script>`
- `@import url(properties.css)`
- `url(background.png)`

Contenus encodés en `UTF-8` :

- `<figcaption>Communauté des développeurs ...</figcaption>`
- `alert("Hello la communauté RennesJS !  ")`
- `content: "  "`

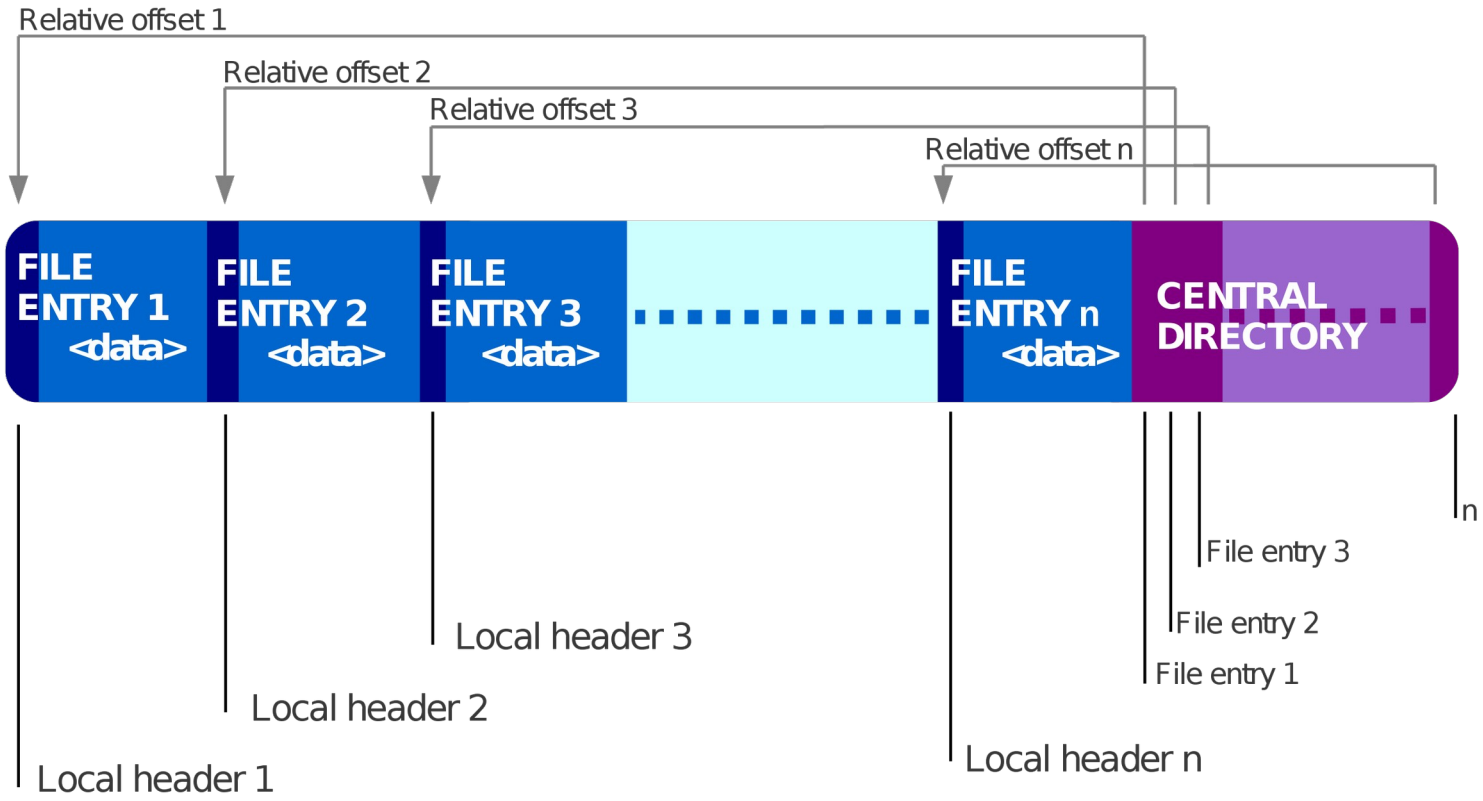
Format ZIP

- Créé en 1989 par Phil Katz chez PKWARE (éditeur de PKZIP)
- Support :
  - De la compression au format DEFLATE depuis la version 2.0 (1993)
  - Du chiffrage AES depuis la version 5.2 (2003)
  - De la version 2.0+ sur la plupart des systèmes d'exploitation
- Exemples de formats basés sur le format ZIP :
  - Documents LibreOffice/MS Office `.ODT`, `.DOCX`, ...
  - Archives Java `.JAR`
  - Packages Android `.APK` et Archives iOS `.IPA`
  - Web Extensions `.CRX` et `.XPI`

Format ZIP

- Entrées (file entries) suivies de l'annuaire central (central directory)
- Adapté pour la lecture/écriture en streaming mais avec quelques limitations en lecture
- Certaines métadonnées sont stockées en double dans les en-têtes locaux (local headers) et l'annuaire central : nom du fichier, date de dernière modification, taille des données ...
- L'annuaire central est requis et fait autorité
- L'annuaire central contient les positions (relative offsets) de chacune des entrées dans le fichier ZIP
- Les données des en-têtes locaux peuvent être utilisées pour réparer un ZIP

Structure d'un fichier ZIP



Source : https://en.wikipedia.org/wiki/ZIP_(file_format)

JavaScript & Format ZIP

- Support natif dans les navigateurs, Node.js, Deno, Bun, etc. :
  - ZIP : non
  - DEFLATE : oui (via gzip/zlib)
- Riche écosystème de bibliothèques tierces :
  - zip.js
  - fflate
  - client-zip
  - yazl/yauzl
  - JSZip
  - ...

Exemple avec zip.js

Ecriture d'un fichier ZIP

```
import { ZipWriter, Uint8ArrayWriter } from "@zip-js/zip-js"

const zipDataWriter = new Uint8ArrayWriter()
const zipWriter = new ZipWriter(zipDataWriter)

for await (const { name } of readDirectory(inputFolder)) {
    const readableStream = await readFileStream(name)
    await zipWriter.add(name, readableStream)
}

await zipWriter.close()
const zipData = zipDataWriter.getData() // Uint8Array
console.log("zip file data:", zipData)
```

Exemple avec zip.js

Lecture d'un fichier ZIP

```
import { ZipReader, BlobReader, BlobWriter } from "@zip-js/zip-js"

const zipReader = new ZipReader(new BlobReader(blob))
const entries = await zipReader.getEntries()

for (const entry of entries) {
    const blob = await entry.getData(new BlobWriter())
    console.log("file:", entry.filename, "blob:", blob)
}

await zipReader.close()
```

2 - Ecriture et lecture du fichier zip en JavaScript

Ajout de `/scripts` :

- `create-zip.js` et `read-zip.js` : fichiers de test
- `lib/utils-fs.js` :
  - `readDirectory(folder)` : retourne les noms de fichiers d'un répertoire
  - `readBinaryFile(filename)` : retourne un fichier sous forme de `Uint8Array`
  - `createFile(filename)` : crée un fichier et retourne le stream en écriture
- `lib/utils.js` :
  - `encodeText(text)` : encode du texte `UTF-8` sous forme de `Uint8Array`
- `lib/utils-zip.js` :
  - `getZipData(inputFolder)` : retourne le contenu sous forme de `Uint8Array` d'un ZIP contenant les fichiers lus depuis un répertoire

Format ZIP (suite)

- Format extensible :
  - Ajout de 64Ko de données après le fichier ZIP (commentaire)
  - Offset supérieur à zéro pour la première entrée dans le fichier zip
- Structure de page HTML auto-extractible :
  - Contenu HTML jusqu'à une balise ouvrante `<!--`
  - Contenu du fichier ZIP
  - Balise fermante `-->` et fin du contenu HTML
- Le contenu HTML inclut :
  - Script de la bibliothèque zip.js (`zip.min.js`)
  - Script pour extraire et afficher le contenu du fichier zip

Template de la page HTML auto-extractible

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- ZIP data -->
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Fichier HTML auto-extractible HTML/ZIP (1/2)

3 - Lecture du fichier ZIP depuis la page HTML

Ajout de `/assets` :

- `zip.min.js` : lecture de fichier ZIP
- `page-extraction.js` : extraction et affichage de la page

Ajouts dans `/scripts` :

- `lib/html-template.js` : template HTML de la page auto-extractible
- `index.js` : génération du fichier auto-extractible HTML/ZIP

Fichier HTML auto-extractible HTML/ZIP (2/2)

Evolutions dans `/scripts` :

- Ajout dans `lib/utils.js` :
  - `minifyScript(script)` : retourne le script minifié
- Ajout dans `lib/utils-fs.js` :
  - `readTextFile(filename)` : retourne le contenu d'un fichier sous forme de texte

Fichier HTML auto-extractible HTML/ZIP

4 - Extraction et affichage de la page `/index.html` depuis le fichier ZIP

Ajout dans `/assets/page-extraction.js` :

- `getExtension(filename)` : retourne l'extension d'un nom de fichier (e.g. `".txt"`)

Evolution dans `/assets/page-extraction.js` :

- `extractResources` : ajout du support des fichiers de type texte
- `displayPage` : implémentation de l'affichage de la page HTML seule

Fichier HTML auto-extractible HTML/ZIP

5 - Affichage de la page HTML complète

Ajout dans `/assets/page-extraction.js` :

- `resolveDependencies(resources)` : remplace les chemins des ressources externes par les ressources extraites du ZIP

6 - Prise en charge des scripts dans la page

Evolution dans `/assets/page-extraction.js` :

- `displayPage` : remplacement dans le DOM des scripts par leurs clones

Lecture de la page depuis le système de fichier

Lecture du fichier ZIP depuis le DOM (1/2)

7 - Affichage en hexadécimal des données ZIP lues sous forme de texte via `Node#textContent`

Retraits temporaires dans `/assets/page-extraction.js` :

- `extractResources`
- `resolveDependencies`
- `getZipData`

Evolution temporaire dans `/assets/page-extraction.js` :

- Remplacement de `displayPage(resources)` par `displayData()`

Ajout de styles temporaires pour afficher le texte avec une police à espacement fixe dans `/scripts/lib/html-template.js`

Lecture du fichier ZIP depuis le DOM (2/2)

8 - Comparaison de l'impact des encodages `UTF-8` et `windows-1252`

Changement de l'attribut `charset` de la balise `<meta>`

Lecture du fichier ZIP depuis le DOM

9 - Test de la corruption de données engendrée par les encodages et détermination du meilleur encodage

Corruption de données :

- Encodage des caractères sur 1 octet (e.g. `windows-1252`) contre plusieurs octets (e.g. `UTF-8`)
- Remplacement par un passage à la ligne `\n` de tout :
  - retour chariot `\r`
  - retour chariot suivi immédiatement d'un passage à la ligne `\r\n`
- Caractères dont le code est supérieur à `127`, i.e. au delà de la table ASCII 7-bit

Lecture du fichier ZIP depuis le DOM (1/2)

10 - Contournement de l'appel à `await fetch("")`

Evolutions dans `/scripts/lib/html-template.js` :

- Remplacement de l'encodage `utf-8` par `windows-1252`
- Ajout du texte ■ `--><script type=text/json>`
- Ajout des ■ données de consolidation (encodées en JSON)
- Remplacement de ■ `-->` par `</script>` dans la partie HTML finale

Template de la page HTML auto-extractible

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=windows-1252>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- ZIP data -->
    <script type=text/json> ZIP extra data </script>
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Lecture du fichier ZIP depuis le DOM (2/2)

Ajout dans `/scripts/lib/utils-zip.js` :

- `getExtraData(zipData)` : génère les données de consolidation

Evolution dans `/scripts/index.js` :

- `writeFile` : ajout des donnés de consolidation

Evolution dans `/assets/page-extraction.js` :

- `getZipData` : lecture du contenu via le commentaire du DOM (données ZIP) et le script JSON (donnés de consolidation)

Lecture du fichier ZIP depuis le DOM

11 - Correction des problèmes de type MIME des ressources externes

Evolutions `/assets/page-extraction.js` :

- `resolveDependencies` : prise en compte des types MIME des ressources (texte ou binaire)
- `extractResources` : modification du code testant l'extension du nom de fichier d'une ressource

La page reste encodée en `windows-1252`

Format PNG

- Créé en 1996
- Format d'image standard et libre de droits
- Compression sans perte (`DEFLATE`)
- Fichier composé d'une suite de blocs (chunk)
- Structure d'un bloc :
  - Taille des données du bloc (4 octets)
  - Type de bloc (4 octets) : `IHDR`, `IDAT`, `IEND`, `tEXt` ...
  - Données du bloc (taille variable)
  - Code redondant cyclique (CRC32) calculé à partir de l'ensemble des données du bloc

Format PNG

- Structure minimale d'un fichier PNG :
  - Signature PNG (8 octets) - obligatoire

    `89 50 4E 47  0D 0A 1A 0A`
  - Bloc `IHDR` pour l'en-tête (13 octets) - obligatoire

    `00 00 00 0D  49 48 44 52  ...`
  - Bloc(s) `IDAT` pour les données de l'image
  - Bloc `IEND` pour la fin des données (12 octets)

    `00 00 00 00  49 45 4E 44  AE 42 60 82`
- Bloc(s) optionnel(s) :
  - Bloc `tEXt` pour stocker des données textes ou binaires quelconques

    `xx xx xx xx  74 45 58 74  ...`

Format PNG

Type de données	Données en hexadécimal	Obligatoire	Taille (octets)
Signature PNG	89 50 4E 47 0D 0A 1A 0A	✓	8
Bloc d'en tête IHDR	00 00 00 0D **49 48 44 52** ...	✓	13
...			
Bloc de données IDAT	xx xx xx xx **49 44 41 54** ...		12 + n
...			
Bloc de fin IEND	00 00 00 00 **49 45 4E 44** AE 42 60 82	✓	12

Format PNG polyglotte (1/3)

12 - Encapsulation du fichier HTML/ZIP dans un fichier PNG

Structure de fichier polyglotte PNG :

Signature PNG

`IHDR` - Bloc d'en-tête

`tEXt` - Bloc de texte

Contenu HTML jusqu'à la balise ouvrante `<!--`

`IDAT` - Bloc(s) de données

`tEXt` - Bloc de texte

Balise fermante `-->` et fin du contenu HTML

`IEND` - Bloc de fin

Format PNG polyglotte (2/3)

Type de données	Données en hexadécimal	Obligatoire	Taille (octets)
Signature PNG	89 50 4E 47 0D 0A 1A 0A	✓	8
Bloc d'en tête IHDR	00 00 00 0D **49 48 44 52** ...	✓	13
Bloc de texte tEXt	xx xx xx xx **74 45 58 74** ...		12 + n
...			
Bloc de données IDAT	xx xx xx xx **49 44 41 54** ...		12 + n
...			
Bloc de texte tEXt	xx xx xx xx **74 45 58 74** ...		12 + n
Bloc de fin IEND	00 00 00 00 **49 45 4E 44** AE 42 60 82	✓	12

Template de la page HTML polyglotte

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=windows-1252>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- PNG data -->
    <!-- ZIP data -->
    <script type=text/json> ZIP extra data </script>
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Format PNG polyglotte (3/3)

Evolution dans `/assets/html-template.js` :

- Ajout du commentaire HTML pour stoker les données de l'image (blocs `IDAT`)

Ajout de `utils-png.js` dans `/scripts/lib` :

- `createTextChunk(keyword, payloadData)`: retourne un nouveau bloc `tEXt`
- `getHeaderData(imageData)` : retourne la signature PNG + bloc `IHDR`
- `getDataChunks(imageData)` : retourne le(s) bloc(s) `IDAT`
- `getTrailerChunk(imageData)` : retourne le bloc `IEND`
- `getChunkDataOffset()` : retourne l'offset des données dans un bloc `tEXt`

Ajout dans `/scripts/lib/utils.js` :

- `mergeData(...data)` : fusionne plusieurs instances de `Uint8Array`

Evolution dans `/scripts/index.js` :

- `writeFile` : intégration du format PNG

Format PNG polyglotte

Améliorations dans `/assets/page-extraction.js` :

13 - Suppression des nœuds texte induits par le format PNG

Ajout de la fonction `cleanupPNGData()`

14 - Correction du mode de rendu de la page HTML et meilleure prise en charge des scripts

Utilisation dans la fonction `displayPage()` de `document.open()`, `document.write()` et `document.close()`

15 - Réutilisation de l'image (polyglotte) dans la page HTML

Suppression de l'image dans `DEPENDENCIES`

Déplacement de l'image `image.png` dans `/assets`

Remplacement de `image.png` par # dans `/project/index.html`

Conclusion

- Limitations de l'implémentation finale :
  - Résolution des dépendances manuelle
  - Dépassement de la limite de 64Ko de données après le ZIP
  - Présence de `-->` dans les données binaires ZIP ou PNG
  - Utilisation de `String#replaceAll()` pour remplacer les chemins dans les fichiers texte au lieu de s'appuyer sur l'analyse syntaxique
  - Absence de tag `<meta>` contenant la `CSP`
  - Non prise en charge des frames ...
- Formats alternatifs : MHTML, Web Bundle, WARC/WACZ, MAFF ...
- Est-ce dangereux ?
  - GIFAR