Comment créer des fichiers polyglottes HTML/ZIP/PNG

Cet article résume la présentation disponible ici. Le fichier démo résultant peut être téléchargé à la fin de l'article.

Introduction

SingleFile, un outil d'archivage web, stocke habituellement les ressources des pages web sous forme d'URI de données. Cependant, cette approche peut être inefficace pour les ressources volumineuses. Une solution plus élégante combine la structure flexible du format ZIP avec HTML. Nous allons pousser cette idée plus loin en encapsulant cette structure entière dans un fichier PNG.

La puissance du ZIP

Le format ZIP fournit une structure organisée pour stocker plusieurs fichiers. Il repose sur une structure composée d'entrées de fichiers suivies d'un répertoire central. Le répertoire central agit comme une table des matières, contenant des en-têtes avec des métadonnées pour chaque entrée de fichier. Ces en-têtes incluent des informations essentielles telles que les noms de fichiers, les tailles, les sommes de contrôle et les décalages des entrées de fichiers. Ce qui rend le ZIP particulièrement polyvalent, c'est sa flexibilité dans le placement des données. Le format permet de préfixer des données avant le contenu ZIP en définissant un décalage supérieur à 0 pour la première entrée de fichier, tout en autorisant l'ajout de jusqu'à 64 Ko de données après (commentaire ZIP). Cette caractéristique le rend particulièrement adapté à la création de fichiers polyglottes.

Création de fichiers polyglottes HTML/ZIP

Avec ces connaissances, nous pouvons créer une archive auto-extractible fonctionnant dans les navigateurs. La page à afficher et ses ressources sont stockées dans un fichier ZIP. En plaçant les données ZIP dans un commentaire HTML, nous obtenons une page auto-extractible qui extrait et affiche le contenu du fichier ZIP.

Structure de base de la page auto-extractible :

```
<!doctype html>
<html>
  <head>
    <meta charset=utf-8>
    <title>Please wait...</title>
    <script> Contenu de lib/zip.min.js </script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- [Données ZIP] -->
    <script> Contenu de assets/main.js </script>
  </body>
</html>
```

Le script `assets/main.js` sur cette "page de démarrage" lit les données ZIP avec `fetch("")` et utilise la bibliothèque JavaScript `lib/zip.min.js` pour les extraire. Cependant, à cause de la politique de même

origine, la récupération des données ZIP avec `fetch("")` échoue lorsque la page est ouverte depuis le système de fichiers (sauf dans Firefox).

Lecture des données ZIP depuis le DOM

Pour contourner cette limitation, nous pouvons lire les données ZIP directement depuis le DOM. Cela nécessite de gérer attentivement l'encodage des caractères. La page de démarrage est encodée en `windows-1252`, permettant une lecture avec une dégradation minimale. Quelques défis d'encodage apparaissent :

1. Le contenu texte du DOM est décodé en `UTF-16` au lieu de `windows-1252`
2. Le caractère NULL (`U+0000`) est décodé en caractère de remplacement (`U+FFFD`)
3. Les retours chariot (`\r`) et les retours chariot + saut de ligne (`\r\n`) sont décodés en saut de ligne (`\n`)

Les deux premiers points peuvent être résolus en utilisant une table d'association pour convertir les caractères en windows-1252. Pour le dernier point, des "données de consolidation" dans une balise de script JSON sont ajoutées à la page de démarrage. Ces données suivent les décalages des retours chariot et des retours chariot suivis de sauts de ligne, permettant une reconstruction précise du contenu d'origine lors de l'extraction des données ZIP.

Voici la structure résultante :

```
<!doctype html>
<html>
  <head>
    <meta charset=windows-1252>
    <title>Please wait...</title>
    <script> Contenu de lib/zip.min.js </script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- [Données ZIP] -->
    <script text=application/json>
    [Données de consolidation]
    </script>
    <script> Contenu de assets/main.js </script>
  </body>
</html>
```

Ajouter PNG au mélange

Le format PNG contient une signature suivie de chunks, chacun ayant :

- Longueur (4 octets)
- Identifiant de type (4 octets), ex. `IHDR` (en-tête), `IDAT` (données), `IEND` (fin), `tEXt` (données personnalisées)...
- Contenu des données (n octets)
- Somme de contrôle CRC32 (4 octets)

Structure minimale d'un fichier PNG :

1. Signature PNG (8 octets)
2. Chunk `IHDR` (13 octets)
3. Un ou plusieurs chunks `IDAT`
4. Chunk `IEND` (12 octets)

La forme finale : fichiers polyglottes HTML/ZIP/PNG

La mise en œuvre ultime combine les trois formats dans un seul fichier. La tolérance du format HTML permet cette complexité. Cependant, cela pose des défis :

1. La signature, les chunks `IHDR` et `IEND` deviennent brièvement visibles comme des nœuds texte et doivent être supprimés dès que la page est analysée
2. La page affichée est rendue en mode quirks, nécessitant un traitement spécifique avec `document.write()` pour analyser la page affichée

Structure vue comme des chunks PNG :

```
[Signature PNG]
[Chunk IHDR]
[Chunk tEXt
  <!doctype html>
  <html>
    <head>
      <meta charset=windows-1252>
      <title>Please wait...</title>
      <script> Contenu de lib/zip.min.js </script>
    </head>
    <body>
      <p>Please wait...</p>
      <!--
]
[Chunks IDAT]
[Chunk tEXt
      -->
      <!-- [Données ZIP] -->
      <script text=application/json>
      [Données de consolidation]
      </script>
      <script> Contenu de assets/main.js </script>
    </body>
  </html>
]
[Chunk IEND]
```

Optimisation grâce à la réutilisation d'images

L'optimisation finale retire l'image principale (i.e. le logo de RennesJS) du fichier ZIP et réutilise la page, interprétée comme un fichier PNG, pour la remplacer dans la page affichée.

Fichier résultant

Téléchargez demo.png.zip.html (un bug dans "Archive Utility" sur macOS empêche la décompression du fichier résultant ; vous pouvez utiliser `unzip` pour contourner ce problème). Vous pouvez également afficher le fichier demo.png.zip.html.