Engineer

UTF-8

T↕ codage Unicode d'un à quatre octets par valeur scalaire

Ce que c’est

UTF-8 est le codage qui transforme le texte Unicode en octets. Le standard Unicode le définit comme la forme de codage qui stocke chaque valeur scalaire Unicode, c’est-à-dire tout point de code sauf les substituts réservés à UTF-16, sous forme d’une séquence d’un à quatre octets. Un caractère perçu comme tel par l’utilisateur peut contenir plusieurs valeurs scalaires et donc nécessiter davantage d’octets. C’est l’une des trois formes de codage du standard, avec UTF-16 et UTF-32, et celle qu’imposent les standards du Web. Comme étiquette, dans une page HTML ou un en-tête HTTP, son nom s’écrit généralement en minuscules, utf-8, parfois sans trait d’union, utf8 ; l’Encoding Standard accepte les deux et ignore la casse des étiquettes.

Ken Thompson a conçu UTF-8 en septembre 1992 pour le système d’exploitation Plan 9, selon des critères définis par Rob Pike. Au cours de sa normalisation, le codage a porté les noms FSS-UTF puis UTF-2 avant de recevoir son nom actuel. La RFC 3629, définition de l’IETF publiée en 2003, le limite aux points de code U+0000 à U+10FFFF, le même espace que celui d’Unicode. À la rédaction de la RFC, la description d’UTF-8 dans ISO/IEC 10646 autorisait encore des séquences allant jusqu’à six octets ; selon la RFC et le standard Unicode, aucune valeur scalaire Unicode n’en nécessite plus de quatre.

Les bits de tête de chaque octet indiquent de quel type d’octet il s’agit. Un octet dont le bit de poids fort est 0 constitue un caractère à lui seul, l’un des 128 caractères ASCII. Un octet commençant par 110, 1110 ou 11110 ouvre une séquence de deux, trois ou quatre octets. Chaque octet suivant commence par 10 et porte six bits supplémentaires du point de code. Tout le format tient en quatre lignes, x représentant les bits du point de code :

```
U+0000..U+007F     0xxxxxxx
U+0080..U+07FF     110xxxxx 10xxxxxx
U+0800..U+FFFF     1110xxxx 10xxxxxx 10xxxxxx
U+10000..U+10FFFF  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

La ligne de trois octets exclut les substituts U+D800 à U+DFFF : ce ne sont pas des caractères, et toute séquence d'octets qui en coderait un est mal formée.

Quatre propriétés découlent de cette structure, et le standard Unicode donne un nom à chacune. UTF-8 est transparent pour ASCII : les octets 00 à 7F ont le même sens qu'en ASCII et n'apparaissent jamais à l'intérieur d'un autre caractère. Il est autosynchronisant : un octet initial se distingue facilement d'un octet de continuation. Trier des chaînes UTF-8 octet par octet donne le même ordre que les trier par point de code. Et il n'a pas d'ordre des octets : la question grand-boutiste ou petit-boutiste à laquelle UTF-16 et UTF-32 doivent répondre ne se pose donc pas.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'UTF-8 ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Un logiciel qui se lit naturellement sur chaque marché où vous vendez, y compris les dates, les devises et le classement des noms.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Système i18n anglais/danois — 12,027 messages par langue, en anglais et en danois

Système i18n anglais/danois Contactez-nous →

Pourquoi c'est important

Le Web l'a adopté. L'Encoding Standard de WHATWG, la spécification implémentée par les navigateurs pour lire et écrire du texte, désigne UTF-8 comme moyen privilégié d'échanger du texte Unicode. Il l'impose aux nouveaux protocoles et formats ainsi qu'aux formats existants déployés dans de nouveaux contextes. Il demande aux auteurs d'utiliser UTF-8 et de l'identifier par l'étiquette utf-8. HTML suit cette règle : le codage d'un document doit être UTF-8, qu'il le déclare ou non, et les outils de création doivent présélectionner UTF-8 pour les nouveaux documents. Un produit qui publie des pages, envoie du JSON ou reçoit des envois de formulaires est déjà censé utiliser ce codage.

Sa conception explique pourquoi ce choix a tenu. Les octets ASCII n'apparaissant jamais à l'intérieur d'un autre caractère, un programme écrit pour ASCII continue de fonctionner avec du texte UTF-8 : un analyseur qui coupe aux virgules, barres obliques ou sauts de ligne les trouve dans une phrase danoise ou ukrainienne exactement là où ils sont, jamais au milieu d'un å ou d'un ľ. Un programme placé à n'importe quelle position retrouve le début d'une valeur scalaire codée en remontant d'au plus quatre octets, généralement moins. Le décodage peut donc reprendre à une séquence suivante bien formée sans perdre le reste du fichier. Cela ne promet pas exactement un caractère de remplacement pour chaque octet endommagé. Une recherche d'une valeur scalaire entièrement codée ne correspond jamais à une partie d'une autre. L'ordre binaire étant celui des points de code, une base de données ou un programme peut trier des octets UTF-8 sans les décoder. Cet ordre reste technique : il place å avant æ et ø, alors qu'un dictionnaire danois termine l'alphabet par æ, ø et å.

La contrepartie est la taille. Le standard Unicode qualifie UTF-8 de raisonnablement compact : pour la syntaxe ASCII et les langues occidentales, il prend bien moins de place qu'UTF-16, tandis que les textes dans des écritures asiatiques comme l'hindi, le thaï, le chinois, le japonais et le coréen deviennent nettement plus volumineux. Dans l'alphabet danois de base, les lettres occupent un octet sauf æ, ø et å, qui en prennent deux ; d'autres lettres accentuées peuvent en nécessiter davantage. L'ukrainien utilise deux octets pour chaque lettre cyrillique et un pour les espaces, chiffres et signes de ponctuation ASCII, mais d'autres signes de ponctuation peuvent en demander plus. Un champ mesuré en octets contient donc environ deux fois moins de lettres ukrainiennes que de lettres anglaises.

À quoi cela ressemble en pratique

Les octets sont faciles à rendre visibles. Node.js, comme un navigateur, possède le `TextEncoder` de l'Encoding Standard, qui écrit une chaîne en UTF-8 :

```
const hex = (bytes) =>
  [...bytes].map((b) => b.toString(16).padStart(2, "0")).join(" ");
const encoder = new TextEncoder();
for (const text of ["A", "å", "ľ", "€", "𐍈"]) {
  console.log(text, hex(encoder.encode(text)).toUpperCase());
}
```

Exécuté avec Node.js 25, il affiche chaque caractère et ses octets en hexadécimal :

```
A 41
å C3 A5
Ґ D2 90
€ E2 82 AC
𐐀 F0 90 90 80
```

La lettre A est en ASCII et conserve son unique octet. Le å danois est U+00E5, qui nécessite huit bits, un de plus que les sept offerts par un seul octet : il prend donc le motif de deux octets. Sa valeur sur onze bits, 00011100101, est séparée en 00011 et 100101 ; l'octet initial devient 110 suivi de 00011, soit C3, et l'octet de continuation devient 10 suivi de 100101, soit A5. Le Ґ ukrainien, U+0490, prend deux octets de la même façon, le signe euro trois et la lettre vieil-italique 𐐀, au-delà de U+FFFF, quatre. Le premier octet de chaque séquence annonce sa longueur : C3 et D2 commencent par 110, E2 par 1110 et F0 par 11110.

Ce site écrit toutes ses pages en UTF-8 et l'annonce là où HTML le demande. Le premier élément de l'en-tête de chaque page est `<meta charset="utf-8">`, avec l'étiquette utf-8 imposée par l'Encoding Standard, bien à l'intérieur des 1 024 premiers octets où HTML exige la déclaration. La page des crédits contient une section de contrôles externes, dont aucun ne vient de nous, chacun exécutant un test en direct plutôt que de nous croire sur parole. L'un d'eux est le W3C Internationalization Checker, qui lit les déclarations de langue et de codage rendant chaque version linguistique de ce site lisible pour son public visé.

Où les problèmes surviennent

Une limite en octets coupe un caractère en deux. Une colonne, un champ de message ou un tampon mesuré en octets peut s'arrêter au milieu d'une séquence. Le nom Ærø compte trois caractères et cinq octets ; une limite de quatre conserve Æ, r et le premier octet de ø :

```
const bytes = new TextEncoder().encode("Ærø");
const cut = new TextDecoder().decode(bytes.slice(0, 4));
console.log(bytes.length, [...cut].map((c) => c.codePointAt(0).toString(16)));
console.log(new TextEncoder().encodeInto("Ærø", new Uint8Array(4)));
```

```
5 [ 'c6', '72', 'fffd' ]
{ read: 2, written: 3 }
```

L'octet initial orphelin est décodé en U+FFFD, le caractère de remplacement : le nom stocké se termine donc par un signe de remplacement à la place du ø. Il faut plutôt couper à une frontière de valeur scalaire Unicode. Sur la dernière ligne, `encodeInto` le fait seul : avec quatre octets de place, il n'écrit que les trois octets de Æ et r et indique dans `read` la portion de chaîne lue, comptée en unités UTF-16. Un code qui tronque des octets qu'il possède déjà peut remonter les octets de continuation, ceux qui commencent par 10, jusqu'à l'octet initial et couper là. Cela conserve un UTF-8 bien formé ; préserver un caractère entier tel que le perçoit l'utilisateur exige aussi de respecter les frontières des groupes de graphèmes.

Une marque d'ordre des octets ouvre le fichier. UTF-8 n'a pas d'ordre des octets et n'a donc pas besoin de marque. Pourtant, un convertisseur qui transforme de l'UTF-16 avec marque d'ordre des octets en UTF-8 transforme cette marque en trois octets EF BB BF. Le standard Unicode ne les exige pas, indique que leur présence n'affecte pas la conformité et ne les recommande pas comme signature d'UTF-8. Les lecteurs les traitent différemment. Ici, le fichier bom.json contient ces trois octets suivis de `{"name":"Ærø"}` :

```
import { readFileSync } from "node:fs";

const text = readFileSync("bom.json", "utf8");
console.log(text.codePointAt(0).toString(16));
try {
  JSON.parse(text);
} catch (error) {
  console.log(error.name);
}
console.log(JSON.parse(new TextDecoder().decode(readFileSync("bom.json"))));
```

```
feff
SyntaxError
{ name: 'Ærø' }
```

Le `readFileSync` de Node conserve la marque sous forme du caractère U+FEFF, et `JSON.parse` rejette le texte à cause d'elle. `TextDecoder` suit l'Encoding Standard, retire la marque et permet d'analyser ces mêmes octets sans erreur. Dans un navigateur, la marque va plus loin : l'Encoding Standard lui donne priorité sur tout, y compris l'en-tête HTTP `Content-Type`. Il faut écrire UTF-8 sans marque et, pour les fichiers provenant d'autres systèmes, les décoder avec un décodeur qui la retire ou supprimer le U+FEFF initial avant l'analyse.

Un contrôle lit les octets avant leur décodage. La RFC 3629 décrit un analyseur qui vérifie son entrée sous forme codée et refuse les octets 2F 2E 2E 2F, l'étape de chemin "/../", mais laisse passer 2F C0 AE 2E 2F, où C0 AE est une écriture illégale sur deux octets du point que l'analyseur lit ensuite comme tel. La RFC relate qu'un virus largement répandu sur les serveurs web en 2001 employait exactement cette astuce. Depuis Unicode 3.1, ces formes qui ne sont pas les plus courtes sont mal formées, et les octets C0, C1 et F5 à FF n'apparaissent jamais dans un UTF-8 valide. Un décodeur conforme, dont `TextDecoder`, transforme ces cinq octets en une barre oblique, deux U+FFFD et "./" : il ne reste donc aucun "/../" capable de franchir le contrôle. Le standard Unicode est tout aussi strict sur ce qui suit une erreur : un décodeur qui avale les octets valides après un octet initial erroné n'est pas conforme et s'expose à des attaques de sécurité. `TextDecoder` transforme les octets C2 41 42 en U+FFFD suivi de AB. Il faut décoder d'abord avec le décodeur de la plateforme, puis vérifier le texte décodé, sans jamais écrire à la main un décodeur UTF-8 ou un motif d'octets qui en tient lieu.

Le remplacement masque les dégâts. Sans réglage particulier, un décodeur remplace chaque séquence erronée par U+FFFD et continue. C'est adapté à l'affichage d'une page, mais pas à l'importation de données : le texte se lit presque normalement, la copie stockée a perdu les octets d'origine et l'expéditeur ne sait jamais où le fichier était endommagé. L'Encoding Standard nomme deux modes d'erreur de décodage, « replacement » et « fatal », et permet à d'autres standards de choisir un décodeur qui échoue plutôt que de remplacer pour les identifiants et les séquences d'octets à l'intérieur d'un format ou protocole. En JavaScript, `new TextDecoder("utf-8", { fatal: true })` lève un `TypeError` pour le nom tronqué ci-dessus au lieu de le renvoyer. Il faut utiliser le mode fatal partout où le texte est stocké, signé ou comparé, rejeter l'entrée avec un message permettant à l'expéditeur d'agir et réserver le remplacement à l'affichage.

La déclaration est tardive, absente ou erronée. HTML exige que l'élément déclarant le codage soit entièrement compris dans les 1 024 premiers octets du document. Il exige une déclaration même si tous les caractères sont ASCII, car le

navigateur a besoin d'un codage pour la saisie des formulaires et pour les URL construites par les scripts. Une étiquette d'apparence inoffensive peut aussi être incorrecte : dans l'Encoding Standard, "ascii", "latin1" et "iso-8859-1" sont toutes des étiquettes de windows-1252. Une page déclarée ASCII est donc lue en windows-1252. Il faut placer `<meta charset="utf-8">` au début de l'en-tête, envoyer `Content-Type: text/html; charset=utf-8` avec chaque page et traiter toute autre étiquette à l'un ou l'autre endroit comme un défaut.

Pour aller plus loin

- The Unicode Standard, chapitre 3 : Conformance — la définition d'UTF-8, la table des séquences d'octets bien formées, la marque d'ordre des octets et le traitement obligatoire des entrées mal formées par un convertisseur.
- The Unicode Standard, chapitre 2 : General Structure — pourquoi UTF-8 est transparent pour ASCII, autosynchronisant et dépourvu d'ordre des octets, et comment sa taille se compare à celle d'UTF-16.
- RFC 3629 : UTF-8, a transformation format of ISO 10646 — la définition de l'IETF, son histoire depuis Plan 9 et les considérations de sécurité derrière l'attaque par codage non minimal du chemin.
- Encoding Standard — la spécification WHATWG suivie par les navigateurs : pourquoi UTF-8 est obligatoire, comment le décodage traite une marque d'ordre des octets et les erreurs, et les interfaces `TextEncoder` et `TextDecoder`.
- HTML : Specifying the document's character encoding — ce qu'une déclaration doit indiquer, où elle doit figurer et pourquoi une page écrite uniquement en ASCII en a besoin.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'UTF-8 ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/utf-8/