Engineer

UTF-8

T↕ スカラー値を一〜四バイトで表すUnicodeの符号化方式

どのようなものか

UTF-8は、Unicodeのテキストをバイトへ変換する符号化方式です。Unicode標準は、各Unicodeスカラー値を、一〜四バイトの列として保存する符号化形式と定義しています。スカラー値とは、UTF-16用に確保したサロゲートを除く、すべてのコードポイントです。人が一文字と認識する単位が、複数のスカラー値から成ることもあり、その分多くのバイトが必要になります。UTF-8は、UTF-16、UTF-32と並ぶ標準の三つの符号化形式の一つであり、Webの標準が要求する形式です。HTMLページやHTTPヘッダーのラベルでは、通常、小文字のutf-8と書き、ハイフンなしのutf8と書く場合もあります。Encoding Standardは両方を認め、ラベルの大文字と小文字を区別しません。

Ken Thompsonは、Rob Pikeが定めた条件に従い、Plan 9 OSのために1992年9月にUTF-8を設計しました。標準化の過程では、現在の名前になる前に、FSS-UTF、次にUTF-2と呼ばれました。2003年のIETFによる定義RFC 3629は、コードポイントをU+0000からU+10FFFFまでに制限し、Unicodeと同じコード空間にしています。RFCの執筆時点では、ISO/IEC 10646のUTF-8の記述は最大六バイトの列を認めていました。RFCとUnicode標準では、どのUnicodeスカラー値も四バイトを超えません。

各バイトの先頭のビットが、そのバイトの種類を示します。最上位ビットが0なら、それだけでASCIIの128文字の一つを表します。110、1110、11110で始まるバイトは、それぞれ二、三、四バイトの列の先頭です。後続の各バイトは10で始まり、コードポイントの残りのビットを六つずつ運びます。形式全体は四行に収まり、xはコードポイントのビットを表します。

```
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
```

三バイトの行からは、サロゲートのU+D800からU+DFFFまでを除きます。これらは文字ではなく、符号化しようとするバイト列は不正です。

この形から四つの性質が導かれ、Unicode標準はそれぞれに名前を付けています。UTF-8はASCIIに対して透過的です。00から7FのバイトはASCIIと同じ意味を持ち、別の文字の内部には現れません。自己同期性もあり、先頭バイトと継続バイトを簡単に区別できます。また、UTF-8の文字列をバイト単位で並べると、コードポイント順と同じ順序になります。そしてバイト順序を持たないため、UTF-16やUTF-32で必要になるビッグエンディアンかリトルエンディアンかの判断がありません。

概念を判断に生かす

「UTF-8とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 日付、通貨、名前の並び順まで含め、販売先のそれぞれの市場で自然に読めるソフトウェアにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 英語・デンマーク語のi18nシステムを構築 — 12,027 英語・デンマーク語の言語ごとのメッセージ

英語・デンマーク語のi18nシステムを構築 お問い合わせ →

なぜ重要なのか

Webでは、すでにUTF-8を使うと決まっています。ブラウザーがテキストの読み書きで従うWHATWG Encoding Standardは、Unicodeテキストの交換にUTF-8を優先し、新しいプロトコルや形式、既存の形式を新しい文脈で使う場合にも要求しています。著者にはUTF-8を使い、utf-8というラベルで示すよう求めています。HTMLも同様です。文書は宣言の有無にかかわらずUTF-8でなければならず、作成ツールも新規文書の既定値をUTF-8にしなければなりません。ページを公開し、JSONを送り、フォームの送信を受ける製品は、すでにUTF-8を扱うことが求められています。

UTF-8が定着した理由は、その設計にあります。ASCIIのバイトは別の文字の内部に現れないため、ASCII用に書いたプログラムはUTF-8でも動きます。コンマ、スラッシュ、改行で区切るパーサーは、デンマーク語やウクライナ語の文でも、その位置を正確に見つけ、åやґの途中を区切りません。任意の位置からでも、最大四バイト、通常はそれより短い範囲を前へさかのぼって調べれば、符号化したスカラー値の先頭が分かります。そのため、後に続く正しい列から復号を再開でき、ファイルの残りを失わずに済みます。ただし、壊れた一バイトにつき必ず一つの置換文字になるという約束ではありません。符号化済みのスカラー値一つの完全な列を検索しても、別の列の一部分には一致しません。バイナリー順がコー

ドポイント順なので、データベースやプログラムは復号せずにUTF-8のバイトを並べ替えられます。ただし、その順序は技術上のものです。åをæ、øより前に置きますが、デンマーク語の辞書では、アルファベットの末尾はæ、ø、åです。

容量との兼ね合いもあります。Unicode標準はUTF-8を十分コンパクトと説明しています。ASCIIの構文や西欧の言語ではUTF-16より大幅に小さくなりますが、ヒンディー語、タイ語、中国語、日本語、韓国語などのアジアの文字体系では、かなり大きくなります。基本的なデンマーク語のアルファベットでは、æ、ø、åが二バイトで、それ以外の文字は一バイトです。ほかのアクセント付き文字は、もっと必要な場合があります。ウクライナ語はキリル文字一つにつき二バイトで、ASCIIの空白、数字、句読点は一バイトですが、ほかの句読点にはもっと必要な場合があります。そのため、バイト数で長さを測る欄には、英語の約半分のウクライナ語の文字しか入りません。

実際の活用例

バイトは簡単に確認できます。Node.jsには、ブラウザーと同じく、文字列をUTF-8で書き出すEncoding Standardの `TextEncoder` があります。

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

Node.js 25で動かすと、各文字と、そのバイトが十六進で出ます。

```
A 41
å C3 A5
Ĺ D2 90
€ E2 82 AC
𐌂 F0 90 8C 82
```

AはASCIIで、一バイトのままです。デンマーク語のåはU+00E5で、八ビットが必要です。一バイト形式が使える七ビットより一つ多いため、二バイトの形式になります。十一ビットで表した値00011100101を、00011と100101に分けます。先頭バイトは110の後に00011を付けたC3、継続バイトは10の後に100101を付

けたA5です。ウクライナ語のґ、U+0490も同じ方法で二バイト、ユーロ記号は三バイト、U+FFFFを超える古イタリア文字𐌊は四バイトです。各列の最初のバイトが長さを示します。C3とD2は110、E2は1110、F0は11110で始まります。

このサイトは、すべてのページをUTF-8で書き、HTMLが要求する場所で宣言しています。各ページのhead内の最初の要素は `<meta charset="utf-8">` です。Encoding Standardが指定するutf-8ラベルを使い、HTMLが宣言の位置として求める最初の1,024バイト以内に十分収まります。クレジットページには外部の検査を集めた節があり、いずれも外部の検査サービスで、こちらの説明をうのみにせず、実際のサイトをその場で検査します。その一つがW3C Internationalization Checkerです。サイトの各言語版を適切な読者が読めるようにする、言語と符号化方式の宣言を調べます。

問題が起きるところ

バイト数の上限で、文字の途中を切る。 データベースの列、メッセージの欄、バッファーをバイト数で測ると、バイト列の途中で終わる場合があります。Ærøは三文字、五バイトなので、四バイトの上限ではÆ、r、øの最初のバイトだけが残ります。

```
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 }
```

孤立した先頭バイトは置換文字U+FFFDに復号され、保存した名前の末尾はøの代わりの記号になります。代わりに、Unicodeスカラー値の境界で切ります。最後の行の `encodeInto` は、それを自身で行います。四バイト分の領域を与えると、Æとrの三バイトだけを書き、文字列のどこまで読んだかをUTF-16単位で `read` に報告します。すでに保持しているバイトを切るコードなら、10で始まる継続バイトを前へさかのぼり、先頭バイトまで戻って、そこで切れます。これでUTF-8は正しい形式を保ちます。人が一文字と認識する単位全体を保つには、さらに書記素クラスターの境界も守る必要があります。

バイト順序マークがファイルの先頭に入る。 UTF-8にはバイト順序がなく、マークは不要です。しかし、バイト順序マーク付きのUTF-16をUTF-8へ変換すると、マークがEF BB BFの三バイトになります。Unicode標準はそれを必須とせず、有無は適合性に影響しないとし、UTF-8の署名としては推奨していません。読み手の扱いも異なります。ここではbom.jsonが、その三バイトに続けて `{"name":"Ærø"}` を保存しています。

```
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' }
```

Nodeの `readFileSync` はマークをU+FEFFとして残し、そのため `JSON.parse` はテキストを拒否します。 `TextDecoder` はEncoding Standardに従ってマークを取り除くので、同じバイトが問題なく解析されます。ブラウザーでは、さらに優先度が高くなります。Encoding Standardは、HTTPの `Content-Type` ヘッダーも含む、ほかの指定すべてよりマークを優先します。UTF-8はマークなしで書き、他システムから来るファイルは、それを除くデコーダーで復号するか、解析前に先頭のU+FEFFを除去します。

復号前のバイトを検査する。 RFC 3629には、符号化された入力を調べ、パスの"/../"を表す2F 2E 2E 2Fを拒否しながら、2F C0 AE 2E 2Fを通すパーサーの例があります。そこでは、不正な二バイトの表記C0 AEを、後でピリオド一つとして読みます。RFCは、2001年にWebサーバーへ広く感染したウイルスが、この方法を使ったと記録しています。Unicode 3.1以降、このような最短でない形式は不正で、C0、C1、F5からFFのバイトは、正しいUTF-8には現れません。 `TextDecoder` などの適合したデコーダーは、その五バイトを、スラッシュ、二つのU+FFFD、"./"に変えるので、検査をすり抜ける"/../"は残りません。エラー後の扱いについても、Unicode標準は厳格です。不正な先頭バイトの後ろの正しいバイトまで飲み込むデコーダーは不適合で、セキュリティー上の悪用を許します。

`TextDecoder` はC2 41 42をU+FFFDに続くABへ変えます。まずプラットフォーム自身のデコーダーで復号し、それからテキストを検査します。UTF-8デコーダーや、その代わりとなるバイトパターンを手書きしてはいけません。

置換が破損を隠す。 デコーダーは既定で、不正な列をU+FFFDへ置き換えて続行します。ページの表示には適していても、データの取り込みには不向きです。テキストはほぼ正常に読め、保存したコピーから元のバイトは失われ、送信者はどこが壊れたか分かりません。Encoding Standardは、復号のエラーモードを"replacement"と"fatal"と呼びます。識別子や、形式・プロトコル内のバイト列については、別の標準が、置換せず失敗するデコーダーを選べるようにしています。JavaScriptでは、`new TextDecoder("utf-8", { fatal: true })` は、上で途中切断した名前を返す代わりに `TypeError` を投げます。保存、署名、比較を行う場所ではfatalモードを使い、送信者が対処できるメッセージとともに入力を拒否し、置換は表示用に限ります。

宣言が遅い、ない、または間違っている。 HTMLは、符号化方式を宣言する要素全体が、文書の最初の1,024バイト以内に収まるよう求めています。すべてASCIIでも宣言は必要です。フォームへ入力する内容や、スクリプトが組み立てるURLにも、ブラウザーは符号化方式を必要とするからです。無害に見えるラベルも、違う方式を表す場合があります。Encoding Standardでは、“ascii”、“latin1”、“iso-8859-1"はすべてwindows-1252のラベルです。そのため、ASCIIと宣言したページはwindows-1252で読まれます。headの最初に `<meta charset="utf-8">` を置き、どのページでも `Content-Type: text/html; charset=utf-8` を送り、どちらかに別のラベルがあれば欠陥として扱います。

参考資料

- Unicode標準、第3章：適合性 — UTF-8の定義、正しいバイト列の表、バイト順序マーク、不正な入力に対する変換器の扱いを定めています。
- Unicode標準、第2章：全体構造 — ASCII透過性、自己同期性、バイト順序がない理由、UTF-16との容量の違いを説明します。
- RFC 3629：UTF-8, a transformation format of ISO 10646 — IETFの定義、Plan 9に始まる歴史、パス内の文字を過長形式で符号化する攻撃のセキュリティー上の考察です。
- Encoding Standard — ブラウザーが従うWHATWG仕様です。UTF-8を要求する理由、復号時のバイト順序マークやエラーの扱い、`TextEncoder` と `TextDecoder` のインターフェースを定めます。
- HTML：文書の文字エンコーディングの指定 — 宣言の内容、置く場所、ASCIIだけのページにも必要な理由です。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「UTF-8とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/utf-8/