Engineer

UTF-8

кодування Unicode: 1–4 байти на скалярне значення

Що це таке

UTF-8 — це кодування, яке перетворює текст Unicode на байти. Стандарт Unicode визначає його як форму кодування, що записує кожне скалярне значення Unicode, тобто будь-яку кодову точку, крім сурогатів, відведених для UTF-16, як послідовність від одного до чотирьох байтів. Символ, який користувач сприймає як один, може містити кілька скалярних значень і тому потребувати більше байтів. Це одна з трьох форм кодування стандарту, поряд з UTF-16 і UTF-32, і саме її вимагають власні стандарти вебу. Як позначку кодування на сторінці HTML чи в заголовку HTTP його зазвичай пишуть малими літерами, utf-8, а часом і без дефіса, utf8; Encoding Standard приймає обидва варіанти й не зважає на регістр у позначках.

UTF-8 придумав Кен Томпсон у вересні 1992 року за критеріями, які сформулював Роб Пайк, для операційної системи Plan 9, а до нинішньої назви воно встигло зватися FSS-UTF і UTF-2. RFC 3629, визначення IETF від 2003 року, обмежує його кодовими точками від U+0000 до U+10FFFF, тобто тим самим кодовим простором, що й у Unicode. Коли писали RFC, опис UTF-8 у стандарті ISO/IEC 10646 ще допускав послідовності до шести байтів; за RFC і стандартом Unicode жодне скалярне значення Unicode не займає більше чотирьох.

Старші біти кожного байта кажуть, що це за байт. Байт, у якого старший біт дорівнює 0, сам є символом, одним із 128 символів ASCII. Байт, що починається з 110, 1110 чи 11110, відкриває послідовність із двох, трьох чи чотирьох байтів, а кожен наступний байт починається з 10 і несе ще шість бітів кодової точки. Увесь формат уміщується в чотири рядки, де x позначає біти кодової точки:

```
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
```

Трибайтовий рядок оминає сурогати, від U+D800 до U+DFFF: вони не є символами, і будь-яка послідовність байтів, яка кодувала б один із них, неправильно сформована.

З такої будови випливають чотири властивості, і стандарт Unicode називає кожну з них. UTF-8 прозоре для ASCII: байти від 00 до 7F означають те саме, що в ASCII, і ніколи не трапляються всередині іншого символу. Воно самосинхронізується: початковий байт легко відрізнити від байта продовження. Двійкове сортування рядків UTF-8 дає той самий порядок, що й сортування за кодовими точками. І в UTF-8 немає порядку байтів, тож питання прямого чи зворотного порядку, big-endian чи little-endian, на яке мусять відповідати UTF-16 і UTF-32, тут не постає взагалі.

Застосуйте поняття до рішення

Застосуйте «Що таке UTF-8?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Програма, що природно читається на кожному ринку, де ви продаєте, — з правильними датами, валютами й сортуванням імен.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Створила англо-данську систему i18n — 12,027 повідомлень на локаль — англійською та данською

Створила англо-данську систему i18n Зв'язатися з нами →

Чому це важливо

Веб уже зробив свій вибір. Encoding Standard від WHATWG, специфікація, яку браузери реалізують для читання й запису тексту, називає UTF-8 найдоречнішим кодуванням для обміну текстом Unicode і вимагає його для нових протоколів і форматів, а також для наявних форматів, які застосовують у нових контекстах. Авторам цей стандарт велить писати в UTF-8 і позначати його як utf-8. HTML іде тим самим шляхом: кодуванням документа мусить бути UTF-8, оголошує це документ чи ні, а інструменти для створення документів мусять типово пропонувати UTF-8 для нових документів. Продукт, що публікує сторінки, надсилає JSON або приймає дані з форм, уже мусить працювати в UTF-8.

Будова кодування пояснює, чому цей вибір закріпився. Оскільки байти ASCII ніколи не трапляються всередині іншого символу, програма, написана для ASCII, і далі працює з текстом UTF-8: парсер, що ділить рядок за комами, скісними рисками чи розривами рядків, знаходить їх в українському чи данському реченні саме там, де вони стоять, і ніколи посередині Ґ чи å. Оскільки програма, що опинилася в довільній позиції, знаходить початок закодованого скалярного значення, відступивши назад щонайбільше на чотири байти, а здебільшого менше, декодування може продовжитися з наступної коректної послідовності без втрати решти файлу. Це не обіцяє рівно одного символу заміни на кожен пошкоджений байт. Пошук одного повного закодованого скалярного значення ніколи не збігається з частиною іншого. Оскільки двійковий порядок збігається з порядком кодових точок, база даних чи програма може сортувати байти UTF-8, не декодуючи їх. Проте цей порядок суто технічний: він ставить ґ, є, і та ї після я, хоча в абетці ґ стоїть одразу за г, є за е, а і та ї за и.

Ціна за це — розмір. Стандарт Unicode називає UTF-8 доволі компактним: значно меншим за UTF-16 для синтаксису ASCII і західних мов і значно більшим для письма азійських мов, як-от гінді, тайської, китайської, японської та корейської. Українська займає два байти на кожну кириличну літеру й один на пробіли, цифри та розділові знаки ASCII, але інші розділові знаки можуть потребувати більше. Тож поле, розмір якого рахують у байтах, уміщує приблизно вдвічі менше українських літер, ніж англійських. У базовій данській абетці літери займають один байт, крім æ, ø і å, які займають по два; інші літери з діакритикою можуть потребувати більше.

Як це виглядає на практиці

Байти легко побачити. У Node.js, як і в браузері, є `TextEncoder` з Encoding Standard, який записує рядок в 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", "ґ", "€", "𐍈"]) {
  console.log(text, hex(encoder.encode(text)).toUpperCase());
}
```

Запущений у Node.js 25, код виводить кожен символ і його байти в шістнадцятковому записі:

```
A 41
å C3 A5
Ґ D2 90
€ E2 82 AC
𐌀 F0 90 8C 82
```

Літера A належить до ASCII й зберігає свій єдиний байт. Скандинавська літера å має кодову точку U+00E5, для якої потрібні вісім бітів, на один більше, ніж сім, які може нести один байт, тож вона бере двобайтовий шаблон. Її одинадцятибітове значення, 00011100101, ділиться на 00011 і 100101; початковий байт стає 110 і 00011, тобто C3, а байт продовження — 10 і 100101, тобто A5. Українська Ґ, U+0490, так само займає два байти, знак євро — три, а літера староіталійського письма 𐌀, що лежить вище за U+FFFF, — чотири. Перший байт кожної послідовності оголошує її довжину: C3 і D2 починаються з 110, E2 — з 1110, а F0 — з 11110.

Цей сайт пише кожну сторінку в UTF-8 і повідомляє про це там, де цього вимагає HTML. Першим елементом у head кожної сторінки йде `<meta charset="utf-8">` з позначкою utf-8, яку приписує Encoding Standard, і він стоїть задовго до кінця перших 1024 байтів, у яких HTML вимагає розмістити оголошення. На сторінці подяк є розділ із перевірками сторонніх сервісів, жоден із яких не наш: кожне посилання там запускає живу перевірку й не вірить нам на слово. Одна з них — Перевірка інтернаціоналізації W3C, яка читає оголошення мови й кодування, завдяки яким кожна мовна версія сайту правильно відображається для свого читача.

Де виникають проблеми

Обмеження в байтах розрізає символ навпіл. Стовпець, поле повідомлення чи буфер, розмір яких рахують у байтах, можуть закінчитися посередині послідовності. Датська назва острова Ærø має три символи й п’ять байтів, і обмеження в чотири байти лишає Æ, r і перший байт 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 }
```

Одинокий початковий байт декодується як U+FFFD, символ заміни, тож збережена назва закінчується заповнювачем там, де було ø. Обрізайте натомість на межі скалярних значень Unicode. `encodeInto` в останньому рядку коду робить це сам: отримавши чотири байти місця, він записує лише три байти Æ і r і повідомляє в `read`, яку частину рядка взяв, рахуючи в одиницях UTF-16. Код, що обрізає байти, які вже має, може відступити через байти продовження, ті, що починаються з 10, до початкового байта й обрізати там. Це зберігає коректність UTF-8; щоб також зберегти цілий символ у сприйнятті користувача, треба дотримуватися меж графемних кластерів.

Файл починається з мітки порядку байтів. UTF-8 не має порядку байтів, тож мітка йому не потрібна, але конвертер, що перетворює UTF-16 з міткою порядку байтів (BOM) на UTF-8, перетворює мітку на три байти EF BB BF. Стандарт Unicode їх не вимагає, каже, що їхня наявність не впливає на відповідність стандарту, і не радить уживати їх як сигнатуру UTF-8. Програми, що читають текст, поводяться з ними по-різному. Ось файл bom.json, що містить ці три байти, а за ними `{"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' }
```

`readFileSync` у Node залишає мітку як символ U+FEFF, і `JSON.parse` через неї відхиляє текст. `TextDecoder` дотримується Encoding Standard, вилучає мітку, і ті самі байти розбираються без помилок. У браузері мітка важить ще більше: Encoding Standard ставить її вище за все інше, зокрема й за заголовок HTTP `Content-Type`. Записуйте UTF-8 без мітки, а файли, що надходять з інших систем, декодуйте декодером, який її вилучає, або вилучайте початковий U+FEFF перед розбором.

Перевірка читає байти до декодування. RFC 3629 описує парсер, що перевіряє вхідні дані в закодованому вигляді й відкидає байти 2F 2E 2E 2F, тобто крок шляху «/../», але пропускає 2F C0 AE 2E 2F, де C0 AE — недопустимий двобайтовий запис крапки, який парсер далі все одно прочитає як крапку. RFC зазначає, що саме цим трюком скористався поширений вірус, який 2001 року атакував вебсервери. Починаючи з Unicode 3.1 такі не найкоротші форми вважаються неправильно сформованими, а байти C0, C1 і від F5 до FF ніколи не трапляються в коректному UTF-8. Декодер, що відповідає стандарту, зокрема `TextDecoder`, перетворює ці п’ять байтів на скісну риску, два U+FFFD і «./», тож жодного «/../», який міг би прослизнути, не лишається. Стандарт Unicode так само суворий щодо того, що йде після помилки: декодер, який поглинає коректні байти після хибного початкового байта, не відповідає стандарту й вразливий до атак, а `TextDecoder` перетворює байти C2 41 42 на U+FFFD і літери AB. Спершу декодуйте вбудованим декодером платформи, потім перевіряйте декодований текст і ніколи не пишіть власноруч декодер UTF-8 чи шаблон байтів, що його підміняє.

Заміна приховує пошкодження. Типово декодер замінює кожну хибну послідовність на U+FFFD і працює далі. Для показу сторінки це правильно, а для імпорту даних — ні: текст читається майже нормально, збережена копія втратила початкові байти, а відправник так і не дізнається, де файл пошкоджено. Encoding Standard називає два режими обробки помилок під час декодування, «replacement» і «fatal», і для ідентифікаторів та послідовностей байтів усередині формату чи протоколу дозволяє іншим стандартам вибрати декодер, що зупиняється з помилкою замість заміни. У JavaScript `new TextDecoder("utf-8", { fatal: true })` для обрізаної назви вище кидає `TypeError`, а не повертає її. Вмикайте режим fatal скрізь, де текст зберігають, підписують чи порівнюють, відхиляйте такі вхідні дані з повідомленням, за яким відправник зможе їх виправити, а заміну залиште для показу.

Оголошення запізнюється, бракує його або воно хибне. HTML вимагає, щоб елемент, який оголошує кодування, повністю вміщувався в перші 1024 байти документа. Оголошення потрібне навіть тоді, коли всі символи належать до ASCII, бо браузеру потрібне кодування для того, що люди вводять у форми, і для URL, які будують скрипти. Позначка, що здається безневинною, теж може бути хибною: в Encoding Standard «ascii», «latin1» та «iso-8859-1» — це позначки windows-1252, тож сторінку, оголошену як ASCII, прочитають як windows-1252. Ставте `<meta charset="utf-8">` першим у head, надсилайте `Content-Type: text/html; charset=utf-8` з кожною сторінкою, а будь-яку іншу позначку в будь-якому з цих двох місць вважайте дефектом.

Що почитати далі

- The Unicode Standard, Chapter 3: Conformance — визначення UTF-8, таблиця коректних послідовностей байтів, мітка порядку байтів і те, як конвертер мусить поводитися з неправильно сформованими вхідними даними.
- The Unicode Standard, Chapter 2: General Structure — чому UTF-8 прозоре для ASCII, самосинхронізується й не має порядку байтів, і як його розмір співвідноситься з UTF-16.
- RFC 3629: UTF-8, a transformation format of ISO 10646 — визначення IETF, історія кодування від Plan 9 і міркування безпеки, що стоять за атакою з надмірно довгими записами символів у шляху.
- Encoding Standard — специфікація WHATWG, якої дотримуються браузери: чому UTF-8 обов'язкове, як декодування поводиться з міткою порядку байтів і помилками, та інтерфейси `TextEncoder` і `TextDecoder`.
- HTML: Specifying the document's character encoding — що мусить сказати оголошення, де воно має стояти і чому воно потрібне навіть сторінці, написаній лише символами ASCII.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке UTF-8?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/utf-8/