**Binary Data Services**

Шаблико Катерина

***struct — Інтерпретувати байти як упаковані двійкові дані***

Модуль struct виконує перетворення між значеннями Python і структурами C, представленими як об'єкти байтів Python. Це можна використовувати для обробки двійкових даних, що зберігаються у файлах або з мережевих з’єднань, серед інших джерел. Він використовує форматні рядки як компактні описи макета структур C і передбачуваного перетворення у або зі значень Python.

*Примітка*

За замовчуванням, результат упаковки структури C включає в себе байти заповнення для підтримки належного вирівнювання для залучених типів C; так само при розпаковуванні враховується вирівнювання. Ця поведінка вибирається таким чином, щоб байти упакованої структури точно відповідали макету в пам'яті відповідної структури C. Щоб обробляти незалежні від платформи формати даних або пропускати неявні байти прокладок, використовуйте стандартний розмір і вирівнювання замість власного розміру та вирівнювання.

**Функції структури та винятки**

Модуль визначає такі виключення та функції:

|  |  |
| --- | --- |
| exception struct.error | Виняток, поставлений з різних приводів; аргумент — це рядок, що описує, що не так. |
| struct.pack(fmt, v1, v2, ...) | Повертає об’єкт байтів, що містить значення v1, v2 тощо, запаковані відповідно до рядка формату fmt. Аргументи повинні точно відповідати значенням, які вимагає формат. |
| struct.pack\_into(fmt, buffer, offset, v1, v2, ...) | Запакує значення v1, v2 тощо відповідно до рядка формату fmt і запише упаковані байти в буфер для запису, починаючи зі зміщення позиції. Зауважте, що зміщення є обов’язковим аргументом. |
| struct.unpack(fmt, buffer) | Розпакує з буфера (ймовірно, запакований pack(fmt, ...)) відповідно до рядка формату fmt. Результатом є кортеж, навіть якщо він містить рівно один елемент. Буфер повинен містити саме ту кількість даних, яку вимагає формат (len(байти) має дорівнювати calcsize(fmt)). |
| struct.unpack\_from(fmt, buffer, offset=0) | Розпакує з буфера, починаючи зі зміщення позиції, відповідно до рядка формату fmt. Результатом є кортеж, навіть якщо він містить рівно один елемент. Буфер повинен містити принаймні ту кількість даних, яку вимагає формат (len(buffer[offset:]) має бути не менше calcsize(fmt)). |
| struct.iter\_unpack(fmt, buffer) | Ітеративно розпакує з буфера відповідно до форматного рядка fmt. Ця функція повертає ітератор, який зчитує фрагменти однакового розміру з буфера, доки не буде використано весь його вміст. Розмір буфера в байтах має бути кратним кількості даних, які вимагає формат, як це відображає calcsize().  Кожна ітерація дає кортеж, визначений рядком форматування. |
| struct.calcsize(fmt) | Повертає розмір структури (і, отже, об’єкта bytes, створеного pack(fmt, ...)), що відповідає рядку формату fmt. |

* fmt– рядок формату.
* v1, v2, … – значення (об’єкти), які потрібно упакувати;
* buffer – буфер, в якому записаний об’єкт, який був попередньо упакований функцією pack(). Розмір цього об’єкту повинен співпадати з розміром, заданим у рядку формату fmt;

**Рядки структурованого формату**

Рядки формату – це механізм, який використовується для визначення очікуваного макета під час пакування та розпакування даних. Вони створені з символів формату, які визначають тип даних, які пакуються/розпаковуються. Крім того, існують спеціальні символи для керування порядком байтів, розміром і вирівнюванням.

**Структура порядку байтів, розмір і вирівнювання**

Типи C за замовчуванням представлені в оригінальному форматі машини та порядку байтів і правильно вирівняні шляхом пропуску байтів панелей, якщо необхідно (відповідно до правил, які використовуються компілятором C).

Крім того, перший символ рядка форматування може вказувати порядок байтів, розмір і вирівнювання запакованих даних відповідно до наступної таблиці:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Порядок байт | Розмір | Вирівнювання |
| **@** | природній | природній | природній |
| **=** | little-endian | стандартний | відсутній |
| **<** | little-endian | стандартний | відсутній |
| **>** | big-endian | стандартний | відсутній |
| **!** | мережевий (= big-endian) | стандартний | відсутній |

Якщо перший символ не є одним із цих, передбачається codecs@codecs.

Значення порядку байт може бути одним з 4-х:

* природній порядок (native). Цей порядок може бути або little-endian або big-endian. Даний порядок визначається залежно від хост-системи;

Наприклад, Intel x86 і AMD64 (x86-64) мають порядковий байт; Motorola 68000 і PowerPC G5 мають тенденцію до великого порядку; ARM і Intel Itanium мають можливість перемикання порядкового рядка (bi-endian). Використовуйте sys.byteorder, щоб перевірити порядковість байтів у вашій системі.

* порядок типу little-endian. При такому порядку першим обробляється молодший байт, а потім вже старший байт;
* порядок типу big-endian. У цьому випадку першим обробляється старший байт, а потім вже молодший байт;
* мережний порядок (network), який за замовчуванням рівний порядку big-endian.

Власний розмір і вирівнювання визначаються за допомогою виразу sizeof компілятора C. Це завжди поєднується з власним порядком байтів.

Немає способу вказати не рідний порядок байтів (примусова заміна байтів); використовуйте відповідний вибір «<» або «>».

*Примітки:*

* Заповнення автоматично додається лише між послідовними членами структури. Заповнення не додається на початку або в кінці закодованої структури.
* Заповнення не додається, якщо використовується не рідний розмір і вирівнювання, напр. з «<», «>», «=» та «!».
* Щоб узгодити кінець структури з вимогою вирівнювання певного типу, закінчіть формат кодом цього типу з нульовим числом повторів.

Стандартний розмір залежить лише від **формату символу:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Формат** | **Тип у мові C** | **Тип у мові Python** | **Стандартний розмір** |
| x | pad byte | без значення |  |
| c | char | байти довжиною 1 | 1 |
| b | signed char | integer | 1 |
| B | unsigned char | integer | 1 |
| ? | \_Bool | bool | 1 |
| codecs | short | integer | 2 |
| CODECS | unsigned short | integer | 2 |
| i | int | integer | 4 |
| I | unsigned int | integer | 4 |
| l | long | integer | 4 |
| L | unsigned long | integer | 4 |
| q | long long | integer | 8 |
| Q | unsigned long long | integer | 8 |
| n | ssize\_t | integer |  |
| N | size\_t | integer |  |
| e | float (експоненційний формат) | float | 2 |
| f | float | float | 4 |
| d | double | float | 8 |
| s | char[] | bytes |  |
| p | char[] | bytes |  |
| P | void\* | integer |  |

*Примітки:*

Код перетворення "?" відповідає типу \_Bool, визначеному в C99. Якщо цей тип недоступний, він моделюється за допомогою char. У стандартному режимі він завжди представлений одним байтом.

Коди перетворення 'q' і 'Q' доступні в рідному режимі, лише якщо компілятор платформи C підтримує C long long, або, у Windows, \_\_int64. Вони завжди доступні в стандартних режимах.

Під час спроби запакувати неціле число за допомогою будь-якого з кодів цілого перетворення, якщо неціле число має метод \_\_index\_\_(), тоді цей метод викликається для перетворення аргументу в ціле число перед пакуванням.

Коди перетворення 'n' та 'N' доступні лише для вихідного розміру (вибраного за замовчуванням або із символом порядку байтів codecs@codecs). Для стандартного розміру ви можете використовувати будь-який з інших форматів цілих чисел, який підходить для вашої програми.

Для кодів перетворення «f» і «d» упаковане представлення використовує формат IEEE 754 binary32 (для «f») або binary64 (для «d»), незалежно від формату, який використовується платформою.

Символ формату «P» доступний лише для власного впорядкування байтів (вибраного за замовчуванням або зі символом порядку байтів «@»). Символ порядку байтів codecs=codecs вибирає використання порядку малого або великого порядку на основі хост-системи. Модуль struct не інтерпретує це як власне впорядкування, тому формат 'P' недоступний.

**Приклади**

Усі приклади передбачають внутрішній порядок байтів, розмір і вирівнювання за допомогою машини з великим порядком байтів.

Базовий приклад пакування/розпакування трьох цілих чисел:

**>>> from** **struct** **import** \*

**>>>** pack('hhl', 1, 2, 3)

b'\x00\x01\x00\x02\x00\x00\x00\x03'

**>>>** unpack('hhl', b'**\x00\x01\x00\x02\x00\x00\x00\x03**')

(1, 2, 3)

**>>>** calcsize('hhl')

8

Розпакованим полями можна назвати, призначивши їх змінним або загорнувши результат в іменований кортеж:

**>>>** record = b'raymond **\x32\x12\x08\x01\x08**'

**>>>** name, serialnum, school, gradelevel = unpack(codecs<10sHHb', record)

**>>> from** **collections** **import** namedtuple

**>>>** Student = namedtuple('Student', 'name serialnum school gradelevel')

**>>>** Student.\_make(unpack(codecs<10sHHb', record))

Student(name=b'raymond codecs, serialnum=4658, school=264, gradelevel=8)

Упорядкування символів форматування може вплинути на розмір, оскільки заповнення, необхідне для задоволення вимог вирівнювання, відрізняється:

**>>>** pack('ci', b'\*codecs, 0x12131415)

b'\*\x00\x00\x00\x12\x13\x14\x15'

**>>>** pack('ic', 0x12131415, b'\*codecs)

b'\x12\x13\x14\x15\*codecs

**>>>** calcsize('ci')

8

**>>>** calcsize('ic')

5

Наступний формат 'llh0l'  визначає два байти заповнення в кінці, припускаючи, що longs вирівняні за 4-байтовими межами:

**>>>** pack('llh0l', 1, 2, 3)

b'\x00\x00\x00\x01\x00\x00\x00\x02\x00\x03\x00\x00'

Це працює лише тоді, коли діють рідний розмір і вирівнювання; стандартний розмір і вирівнювання не забезпечує жодного вирівнювання.

**Модуль codecs** обробляє стиснення та декомпресію деяких стандартних форматів даних.

Цей модуль визначає базові класи для стандартних кодеків Python (кодери та декодери) і надає доступ до внутрішнього реєстру кодеків Python, який керує процесом пошуку кодеків і обробки помилок.

**Базові класи кодеків**

Модуль кодеків визначає набір базових класів, які визначають інтерфейс, а також можуть використовуватися для легкого написання кодеків для використання в Python.

Кожен кодек має визначити чотири інтерфейси, щоб його можна було використовувати як кодек у Python: кодер без стану, декодер без стану, зчитувач потоку та записувач потоку. Зчитувач потоків і записи часто повторно використовують кодер/декодер без стану для реалізації протоколів файлів.

Клас Codec визначає інтерфейс для кодерів/декодерівs без стану.

Щоб спростити та стандартизувати обробку помилок, методи encode() і decode() можуть реалізувати різні схеми обробки помилок шляхом надання аргументу рядка помилок. Усі стандартні кодеки Python визначають і реалізують такі рядкові значення:

|  |  |
| --- | --- |
| Значення | Значення |
| 'strict' | Викликати помилку UnicodeError; це за замовчуванням. |
| 'ignore' | Ігнорувати та переходити до наступного символу. |
| 'replace' | Замінити відповідним символом заміни; Python використовує офіційний символ заміни для вбудованих кодеків Unicode для декодування та «?» для кодування. |
| 'xmlcharrefreplace' | Замінити відповідним посиланням на символ XML (лише для кодування). |
| 'backslashreplace' | Замінити escape-послідовністю з зворотною косою рискою (тільки для кодування). |
| 'surrogatepass' | Дозволити кодування та декодування сурогатних кодів у всьому кодуванні Unicode схеми. (codec: utf-8, utf-16, utf-32, utf-16-be, utf-16-le, utf-32-be, utf-32-le) |

Набір дозволених значень можна розширити за допомогою register\_error().

**Об'єкти кодека**

Клас Codec визначає ці методи, які також визначають інтерфейси функцій кодера та декодера без стану:

|  |  |
| --- | --- |
| Codec.encode(input [, errors]) | Кодує вхідний об’єкт і повертає кортеж (об’єкт виведення, споживана довжина). Кодування перетворює рядковий об’єкт в об’єкт байтів за допомогою певного кодування набору символів (наприклад, cp1252 або iso-8859-1).  Помилки визначають обробку помилок, які потрібно застосувати. За замовчуванням це «суворе» оброблення.  Метод може не зберігати стан у екземплярі кодека. Використовуйте StreamCodec для кодеків, які повинні зберігати стан, щоб зробити кодування/декодування ефективним.  У цій ситуації кодер повинен мати можливість обробляти введення нульової довжини та повертати порожній об’єкт типу вихідного об’єкта. |
| Codec.decode(input[, errors]) | Декодує вхідний об’єкт і повертає кортеж (вихідний об’єкт, спожита довжина). Декодування перетворює об’єкт байтів, закодований за допомогою кодування певного набору символів, у рядковий об’єкт.  Вхідним має бути об’єкт байтів або той, який забезпечує інтерфейс буфера символів лише для читання – наприклад, об’єкти буфера та файли, відображені в пам’яті.  Помилки визначають обробку помилок, які потрібно застосувати. За замовчуванням це «суворе» оброблення.  Метод може не зберігати стан у екземплярі кодека. Використовуйте StreamCodec для кодеків, які повинні зберігати стан, щоб зробити кодування/декодування ефективним.  У цій ситуації декодер повинен мати можливість обробляти вхідні дані нульової довжини та повертати порожній об’єкт типу вихідного об’єкта. |

Класи IncrementalEncoder і IncrementalDecoder забезпечують базовий інтерфейс для інкрементного кодування та декодування. Кодування/декодування вхідних даних виконується не одним викликом функції кодера/декодера без стану, а декількома викликами методу encode()/decode() інкрементного кодера/декодера. Інкрементний кодер/декодер відстежує процес кодування/декодування під час викликів методів.

Об’єднаний вихід викликів методу encode()/decode() такий самий, як якщо б усі окремі входи були об’єднані в один, і цей вхід був закодований/декодований за допомогою кодера/декодера без стану.

**Кодеки кодування та Юнікод**

Рядки зберігаються всередині як послідовності кодових точок в діапазоні 0 - 10FFFF (див. PEP 393 для отримання додаткової інформації про реалізацію). Після того, як об’єкт рядка використовується за межами ЦП і пам’яті, стає проблемою байтове розташування ЦП і те, як ці масиви зберігаються у вигляді байтів. Перетворення рядкового об’єкта в послідовність байтів називається кодуванням, а відтворення рядкового об’єкта з послідовності байтів – декодуванням. Існує багато різних методів для того, як це перетворення можна зробити (ці методи також називаються кодуваннями). Найпростіший метод — зіставити кодові точки 0-255 з байтами 0x0-0xff. Це означає, що об’єкт рядка, що містить кодові точки вище U+00FF, не може бути закодований цим методом (який називається 'latin-1' або 'iso-8859-1'). str.encode() викликає UnicodeEncodeError, який виглядає так:

UnicodeEncodeError: кодек 'latin-1' не може кодувати символ

"\u1234" у позиції 3: порядковий номер не в діапазоні (256).

Існує ще одна група кодувань (так звані кодування charmap), які вибирають різну підмножину всіх кодових точок Unicode і як ці кодові точки відображаються в байтах 0x0-0xff. Щоб побачити, як це робиться, відкрийте, наприклад, encodings/cp1252.py (яке є кодуванням, яке використовується переважно в Windows). Існує рядкова константа з 256 символів, яка показує, який символ зіставляється зі значенням кожного байта.

Без зовнішньої інформації неможливо надійно визначити, яке кодування було використано для кодування рядка. Кожне кодування charmap може декодувати будь-яку випадкову послідовність байтів. Однак це неможливо з UTF-8, оскільки послідовності байтів UTF-8 мають структуру, яка не дозволяє довільні послідовності байтів. Щоб підвищити надійність, з якою можна виявити кодування UTF-8, Microsoft винайшла варіант UTF-8 (який Python 2.5 називає "utf-8-sig") для своєї програми Блокнот: до того, як будь-який із символів Unicode буде записаний до файлу записується специфікація, закодована UTF-8 (яка виглядає так як послідовність байтів: 0xef, 0xbb, 0xbf).

**Стандартні кодування**

Python постачається з низкою вбудованих кодеків, реалізованих як функції C, або зі словниками як таблиці зіставлення. У наступній таблиці перелічено кодеки за іменами разом із кількома поширеними псевдонімами та мовами, для яких, ймовірно, використовується кодування. Ні список псевдонімів, ні список мов не є вичерпними. Зверніть увагу, що варіанти написання, які відрізняються лише регістром або використовують дефіс замість підкреслення, також є дійсними псевдонімами; тому, напр. 'utf-8' є дійсним псевдонімом для кодека 'utf\_8'.

| **Кодек** | **Псевдоніми** | **Мови** |
| --- | --- | --- |
| ascii | 646, us-ascii | Англійська |
| big5 | big5-tw, csbig5 | Традиційна китайська |
| big5hkscs | big5-hkscs, hkscs | Традиційна китайська |
| cp037 | IBM037, IBM039 | Англійська |
| cp273 | 273, IBM273, csIBM273 | Німецька |
| cp424 | EBCDIC-CP-HE, IBM424 | Іврит |
| cp437 | 437, IBM437 | Англійська |
| cp500 | EBCDIC-CP-BE, EBCDIC-CP-CH, IBM500 | Західноєвропейський |
| cp720 |  | Арабська |
| cp737 |  | Грецька |
| cp775 | IBM775 | Балтійські мови |
| cp850 | 850, IBM850 | Західноєвропейський |
| cp852 | 852, IBM852 | Центральної та Східної Європи |
| cp855 | 855, IBM855 | Болгарська, Білоруська, Македонська, Російська, Сербська |
| cp856 |  | Іврит |
| cp857 | 857, IBM857 | Турецька |
| cp858 | 858, IBM858 | Західноєвропейський |
| cp860 | 860, IBM860 | Португальська |
| cp861 | 861, CP-IS, IBM861 | Ісландська |
| cp862 | 862, IBM862 | Іврит |
| cp863 | 863, IBM863 | Канадська |
| cp864 | IBM864 | Арабська |
| cp865 | 865, IBM865 | Датська , Норвезька |
| cp866 | 866, IBM866 | Російська |
| cp869 | 869, CP-GR, IBM869 | Грецька |
| cp874 |  | Тайська |
| cp875 |  | Грецька |
| cp932 | 932, ms932, mskanji, ms-kanji | Японська |
| cp949 | 949, ms949, uhc | Корейська |
| cp950 | 950, ms950 | Традиційна китайська |
| cp1006 |  | Мова урду |
| cp1026 | ibm1026 | Турецька |
| cp1125 | 1125, ibm1125, cp866u, ruscii | Українська |
| cp1140 | ibm1140 | Західноєвропейський |
| cp1250 | windows-1250 | Центральної та Східної Європи |
| cp1251 | windows-1251 | Болгарська, Білоруська, Македонська, Російська, Сербська |
| cp1252 | windows-1252 | Західноєвропейський |
| cp1253 | windows-1253 | Грецька |
| cp1254 | windows-1254 | Турецька |
| cp1255 | windows-1255 | Іврит |
| cp1256 | windows-1256 | Арабська |
| cp1257 | windows-1257 | Балтійські мови |
| cp1258 | windows-1258 | В’єтнамська |
| euc\_jp | eucjp, ujis, u-jis | Японська |
| euc\_jis\_2004 | jisx0213, eucjis2004 | Японська |
| euc\_jisx0213 | eucjisx0213 | Японська |
| euc\_kr | euckr, korean, ksc5601, ks\_c-5601, ks\_c-5601-1987, ksx1001, ks\_x-1001 | Корейська |
| gb2312 | chinese, csiso58gb231280, euc-cn, euccn, eucgb2312-cn, gb2312-1980, gb2312-80, iso-ir-58 | Спрощена Китайська |
| gbk | 936, cp936, ms936 | Єдина китайська |
| gb18030 | gb18030-2000 | Єдина китайська |
| hz | hzgb, hz-gb, hz-gb-2312 | Спрощена Китайська |
| iso2022\_jp | csiso2022jp, iso2022jp, iso-2022-jp | Японська |
| iso2022\_jp\_1 | iso2022jp-1, iso-2022-jp-1 | Японська |
| iso2022\_jp\_2 | iso2022jp-2, iso-2022-jp-2 | Японська, Корейська, Спрощена Китайська, Західноєвропейський, Грецька |
| iso2022\_jp\_2004 | iso2022jp-2004, iso-2022-jp-2004 | Японська |
| iso2022\_jp\_3 | iso2022jp-3, iso-2022-jp-3 | Японська |
| iso2022\_jp\_ext | iso2022jp-ext, iso-2022-jp-ext | Японська |
| iso2022\_kr | csiso2022kr, iso2022kr, iso-2022-kr | Корейська |
| latin\_1 | iso-8859-1, iso8859-1, 8859, cp819, latin, latin1, L1 | Західноєвропейський |
| iso8859\_2 | iso-8859-2, latin2, L2 | Центральної та Східної Європи |
| iso8859\_3 | iso-8859-3, latin3, L3 | Есперанто, Мальтійська |
| iso8859\_4 | iso-8859-4, latin4, L4 | Балтійські мови |
| iso8859\_5 | iso-8859-5, cyrillic | Болгарська, Білоруська, Македонська, Російська, Сербська |
| iso8859\_6 | iso-8859-6, arabic | Арабська |
| iso8859\_7 | iso-8859-7, greek, greek8 | Грецька |
| iso8859\_8 | iso-8859-8, hebrew | Іврит |
| iso8859\_9 | iso-8859-9, latin5, L5 | Турецька |
| iso8859\_10 | iso-8859-10, latin6, L6 | Скандинавські мови |
| iso8859\_11 | iso-8859-11, thai | Тайські мови |
| iso8859\_13 | iso-8859-13, latin7, L7 | Балтійські мови |
| iso8859\_14 | iso-8859-14, latin8, L8 | Кельтські мови |
| iso8859\_15 | iso-8859-15, latin9, L9 | Західноєвропейський |
| iso8859\_16 | iso-8859-16, latin10, L10 | Південно-Східна Європа |
| johab | cp1361, ms1361 | Корейська |
| koi8\_r |  | Російська |
| koi8\_t |  | Таджицька |
| koi8\_u |  | Українська |
| kz1048 | kz\_1048, strk1048\_2002, rk1048 | Казахська |
| mac\_cyrillic | maccyrillic | Болгарська, Білоруська, Македонська, Російська, Сербська |
| mac\_greek | macgreek | Грецька |
| mac\_iceland | maciceland | Ісландська |
| mac\_latin2 | maclatin2, maccentraleurope, mac\_centeuro | Центральної та Східної Європи |
| mac\_roman | macroman, macintosh | Західноєвропейський |
| mac\_turkish | macturkish | Турецька |
| ptcp154 | csptcp154, pt154, cp154, cyrillic-asian | Казахська |
| shift\_jis | csshiftjis, shiftjis, sjis, s\_jis | Японська |
| shift\_jis\_2004 | shiftjis2004, sjis\_2004, sjis2004 | Японська |
| shift\_jisx0213 | shiftjisx0213, sjisx0213, s\_jisx0213 | Японська |
| utf\_32 | U32, utf32 | усі мови |
| utf\_32\_be | UTF-32BE | усі мови |
| utf\_32\_le | UTF-32LE | усі мови |
| utf\_16 | U16, utf16 | усі мови |
| utf\_16\_be | UTF-16BE | усі мови |
| utf\_16\_le | UTF-16LE | усі мови |
| utf\_7 | U7, unicode-1-1-utf-7 | усі мови |
| utf\_8 | U8, UTF, utf8, cp65001 | усі мови |
| utf\_8\_sig |  | усі мови |