5. RFC7049：Concise Binary Object Representation (CBOR)
RFC7049：簡明二進制物件表示法（CBOR）

網際網路工程任務組（IETF）
意見請求：7049
類別：標準跟蹤
ISSN：2070-1721

C. Bormann
Universitaet Bremen TZI
P. Hoffman
VPN Consortium

2013 年 10 月

簡明二進制物件表示法（CBOR）

摘要

簡明二進制物件表示法（CBOR）是一種資料格式，其設計目標包括極小的編碼大小，和相當小的訊息大小和可擴展性的可能性，而無需版本協商. 這些設計目標使其與早期的二進制序列化方式（如 ASN.1 和 MessagePack）不同。

本文的狀態

這是網際網路標準跟蹤文件。

本文是網際網路工程任務組（IETF）的產品。它代表了 IETF 社區的共識。它已經過公眾審查，並已獲得網際網路工程指導組（IESG）的批准發布。有關網際網路標準的更多訊息，請參見 RFC 5741 的第 2 節。

有關本文當前狀態，任何勘誤以及如何提供反饋的訊息，請訪問 http://www.rfc-editor.org/info/rfc7049。

版權聲明

起生效。請仔細閱讀這些文件，因為它們描述了您對本文的權利和限制。從本文中提取的編碼元件必須包含信任法律規定第 4.e 節中所述的簡化 BSD 授權條款，並且不提供簡化 BSD 授權條款中所述的保證。

目錄

1. 前言 3
   1.1. 目標 4
   1.2. 術語 5
2. CBOR編碼的規範 6
   2.1. 主要類型 7
   2.2. 某些主要類型的無限長度 9
   2.3. 沒有內容的浮點數和值 12
   2.4. 可選的項目標籤 14
3. 創建基於CBOR的協定 19
   3.1. 串流應用中的CBOR 20
   3.2. 通用編碼器和解碼器 21
   3.3. 語法錯誤 21
   3.4. 其他解碼錯誤 23
   3.5. 處理未知的簡單值和標籤 23
   3.6. 數字 24
   3.7. 指定映射的鍵 25
   3.8. 未定義的值 26
   3.9. 典型CBOR 26
   3.10. 嚴格的模式 27
4. 在CBOR和JSON之間轉換資料 29
   4.1. 從CBOR轉換為JSON 29
   4.2. 從JSON轉換為CBOR 30
5. CBOR的未來演變 31
   5.1. 擴展點 32
   5.2. 策劃附加訊息空間 32
6. 偵錯表示法 33
   6.1. 編碼指標 34
7. IANA考量 35
   7.1. 簡單值註冊表 35

7.2. 標籤註冊表 35
7.3. 媒體類型（“MIME類型”） 36
7.4. CoAP內容格式 36
7.5. + cbor結構化語法尾綴註冊 37
8. 安全考量 37
9. 致謝 38
10. 參考文獻 39
10.1. 規範性參考文獻 39
10.2. 訊息參考 40
附錄A. 範例 41
附錄B. 跳越表 45
附錄C. 偽碼 48
附錄D. 半精確度 50
附錄E. 其他二進制格式與CBOR設計的比較 51
E.1. ASN.1 DER，BER和PER 52
E.2. MessagePack 52
E.3. BSON 52
E.4. UBJSON 53
E.5. MSDTP: RFC 713 53
E.6. 簡潔性電線 53

1. 前言

結構化資料的二進制表示有數百種標準化格式（也稱為二進制序列化格式）。其中一些是針對特定的訊息領域，而另一些是廣義的針對任意資料進行。在 IETF 中，後一類中最著名的格式可能是 ASN.1 的 BER 和 DER [ASN.1]。

此處定義的格式遵循當前格式無法很好地滿足一些特定設計目標。底層資料模型是 JSON 資料模型的擴展版本[RFC4627]。重要的是要注意，這不是一般性地擴展 RFC 4627 中的語法方案，因為這樣做會導致與已部署的 JSON 文件發生嚴重的向後不兼容。相反，本文只是定義了自己的資料模型，它從 JSON 開始。

附錄 E 列出了一些現有的二進制格式，並討論了它們與簡明二進制物件表示（CBOR）的設計目標的相符程度。

1.1. 目標

CBOR 的目標大致依重要性高低排列：

1. 表示必須能夠明確地編碼網際網路標準中使用的大多數常見資料格式。

   * 它必須使用二進制編碼表示一組合理的基本資料類型和結構。這裡的“合理”很大程度上受到 JSON 功能的影響，主要是增加了二進制位元組字串。支援的結構僅限於陣列和樹；不支援循環和格狀樣式圖。

   * 不要求所有資料格式都是唯一編碼；也就是說，可以以多種不同的方式編碼數字“7”。

2. 編碼器或解碼器的編碼必須能夠緊湊，以支援具有非常有限的記憶體，處理器能力和指令集的系統。

   * 編碼器和解碼器需要以非常少量的編碼實施（例如，在 [CNN-TERMS]中定義的 1 類受約束節點中）。

   * 格式應該使用資料的當代機器表示（例如，不要求二進制到十進制的轉換）。

3. 必須能夠在沒有模式描述的情況下解碼資料。

   * 與 JSON 類似，編碼資料應該是自描述的，以便可以編寫通用解碼器。

4. 序列化必須相當緊湊，但資料緊湊性是編碼器和解碼器的編碼緊湊性的次要因素。

   * 這裡的“合理”是 JSON 作為大小的上限，並且通過實施複雜性保持下限。使用通用壓縮方案或廣泛的位功能違反了複雜性目標。

5. 該格式必須適用於受約束的節點和高容量應用程式。

   * 這意味著編碼和解碼的 CPU 使用率必須合理節儉。這對於受限節點和具有非常高資料量的應用程式中的潛在用途都是相關的。

6. 該格式必須支援所有 JSON 資料類型，以便與 JSON 進行轉換。

   * 只要所表示的資料在 JSON 的能力範圍內，它就必須支援合理的轉換級別。必須可以為所有類型的資料定義針對 JSON 的單向映射。

7. 格式必須是可擴展的，並且擴展資料必須可由早期解碼器解碼。

   * 該格式設計使用數十年。

   * 格式必須支援一種可擴展的形式，允許回退，以便不理解擴展的解碼器仍然可以解碼該訊息。

   * 以後的 IETF 標準必須能夠擴展該格式。

1.2. 術語

本文件中的關鍵詞"必須(MUST)"、"不得(MUST NOT)"，"必要(REQUIRED)"，"必須(SHALL)"，"不得(SHALL NOT)"，"應該(SHALL NOT)"，"不應該(SHOULD NOT)"，"建議(RECOMMENDED)"，"不建議(RECOMMENDED)"，"可以(MAY)"，"可選(OPTIONAL)"按照 RFC 2119，BCP 14 [RFC2119] 中的描述進行解釋，並指出符合要求的 CBOR 實施的要求等級。

術語“位元組”在其現在習慣的意義上用作“八位元組”的同義詞。所有多位元組值都以網路位元組順序編碼（即最重要的位元組優先，也稱為“big-endian”）。

該規範使用以下術語：

資料項：單一 CBOR 資料。資料項的結構可以包含零個，一個或多個巢狀資料項。該術語既用於表示格式的資料項，也用於可由解碼器從中導出的抽象概念。

解碼器:解碼一個CBOR資料項,並使其可用於應用程式的方法。正式地說，解碼器包含一個解析器分割使用 CBOR 的語法規則，以及語意處理器以適合應用的形式準備資料的輸入端。

編碼器：產生從應用訊息的 CBOR 資料項表示格式的方法。

資料串流：零個或多個資料項的序列，不能進一步組裝成更大的資料項。構成一個資料串流的獨立資料項有時也被稱為“頂層資料項”。

形式完善：遵循 CBOR 的語法結構資料項。形式完善的資料項使用初始位元組和位元組字串 and/or 資料項，這些位元組字串 and/or 資料項由 CBOR 中隱含的值定義，不會依據無關的資料。

有效：形式完善的資料項，也遵循語意限制可應用於 CBOR 資料項。

流解碼器：對資料串流進行解碼並使應用程式中的每個資料項在接收時可用的過程。

在解釋位元算術或資料類型的情況下，本文使用語言 C 中熟悉的符號，除了“**”表示指數。與十六進制數字的“0x”表示法類似，二進制表示法中的數字以“0b”為前綴。為了便於閱讀，可以將底線添加到這樣的數字中，因此可以寫入 0b00100001（0x21）0b001_00001 以強調對位元組中位元的所需解釋；
在這種情況下，它被分成三位元和五位元。

2．CBOR 編碼的規範

CBOR 編碼的資料項結構和編碼如本節所述。編碼總結在表 5 中。

每個資料項的初始字元包含主要類型（高階 3 位元，在 2.1 節中描述）和附加訊息（低階 5 位元）的訊息。當附加訊息的值小於 24 時，它直接作為小的無符號整數。當它是 24 到 27 時，附加位元組依據可變長度整數來立即表示；附加訊息的值為 24 到 27 時分別指定其長度是 1 位元組，2 位元組，4 位元組或 8 位元組無符號整數。附加訊息值為 31 用於無限長的項目，如第 2.2 節所述。保留其他訊息值 28 到 30 以供將來擴展。
在所有附加訊息值中，根據主要類型解釋生成的整數。它可以表示實際資料：例如，在整數型態中，整數用於表示值本身結果。它可以改為提供長度訊息：例如，在位元組字串中，它給出了後面的位元組字串資料的長度

CBOR 解碼器實施是基於所有的跳越表，初始位元組的 256 個定義值（表 5）。受約束實施中的解碼器可以使用初始位元組的結構和後續位元組來獲得更緊湊的編碼（請參閱附錄 C，以獲得對其外觀的粗略印象）。

2.1. 主要類型

下面列出了主要類型以及與該類型相關的附加訊息和其他位元組。

主要類型 0：無符號整數。5 位元附加訊息是整數本身（對於附加訊息值 0 到 23）或附加資料的長度。附加訊息 24 表示值在另外的 uint8_t 中表示，25 表示 uint16_t，26 表示 uint32_t，27 表示 uint64_t。例如，整數 10 表示為一個位元組 0b000_01010（主要類型 0，附加訊息 10）。整數 500 為 0b000_11001（主要類型 0，附加訊息 25）後跟兩個位元組 0x01f4，即十進制 500 。

主要類型 1：負整數。編碼遵循無符號整數（主要類型 0）的規則，除了值為-1 減去編碼的無符號整數。例如，整數-500 為

0b001_11001（主要類型 1，附加訊息 25），後跟兩個位元組 0x01f3，即十進制的 499 。

主要類型 2：位元組字串。字串的長度（以位元組為單位）遵循正整數規則（主要類型 0）表示。例如，長度為 5 的位元組字串將具有初始位元組 0b010_00101（主要類型 2，長度的附加訊息 5），後跟 5 個位元組的二進制內容。長度為 500 的位元組字串將包含 3 個的初始位元組 0b010_11001（主要類型 2，附加訊息 25 表示兩個位元組的長度），後跟兩個位元組 0x01f4，長度為 500，後跟 500 位元組的二進制內容。

主要類型 3：文字字串，特別是編碼為 UTF-8 [RFC3629]的 Unicode 字串。此類型的格式與位元組字串（主要類型 2）的格式相同，即與主要類型 2 一樣，長度給出位元組數。此類型是為需要解釋或顯示人類可讀文本的系統提供的，並允許區分非結構化位元組和具有指定指令表和編碼的文本。與 JSON 等格式相比，此類型的 Unicode 字元永遠不會被逸出。因此，換行字元（U＋000A）始終在字串中表示為位元組 0x0a，而從不作為位元組 0x5c6e（字元"\"和"n"）或 0x5c7530303061（字元"\", "u", "0", "0", "0",和"a"）。

主要類型 4：資料項陣列。陣列也稱為列表，序列或元組。陣列的長度遵循位元組字串（主要類型 2）的規則。陣列中的項不需要都是相同的類型。例如，包含 10 個任意類型項的陣列的初始位元組為 0b100_01010（主要類型為 4，長度為 10 的附加訊息），其後是 10 個剩餘項。

主要類型 5：資料項對的映射。映射也稱為表格，字典，散列或物件（以 JSON 格式）。映射由成對的資料項組成，每對資料項由一個鍵緊跟著一個值組成。映射的長度遵循位元組字串（主要類型 2）的規則，除了長度表示對的數量，而不是映射的位元組長度佔用。例如，包含 9 對的映射將具有 0b101_01001 的初始位元組（主要類型為 5，附加訊息為對的數量 9），其後是剩餘的 18 個項目。第一個項是第一個鍵，第二個項是第一個值，第三個項是第二個鍵，依此類推。具

有重複鍵的映射可能是格式良好的，但無效，因此會導致不確定的解碼；另見第 3.7 節。

主要類型 6：其他主要類型的可選語意標籤。見 2.4 節。

主要類型 7：不需要內容的浮點數和簡單資料類型，以及“中斷”停止編碼。見 2.3 節。

這八種主要類型導致一個簡單的表格，顯示資料項初始位元組的 256 個可能值中的哪一個（表 5）。

在主要類型 6 和 7 中，許多可能的值保留用於將來的規範。有關這些值的更多訊息，請參見第 7 節。

2.2. 某些主要類型的無限長度

可以使用附加訊息值 31 以無限長度編碼四個 CBOR 項（陣列，映射，位元組字串和文本字串）。如果項目的編碼需要在知道陣列或映射中的項目數或字符串的總長度之前開始，則這很有用。（應用程式通常在資料項中稱為“串流”。）

無限長陣列和映射的處理方式與無限長位元組字串和文本字串不同。

2.2.1. 無限長陣列和映射

使用附加訊息值 31，簡單地打開無限長陣列和映射，而不指示將包含在陣列或映射中的資料項的數量。初始主要類型和附加訊息位元組後面是陣列的元素或映射，就像它們在其他陣列或映射中一樣。通過在通常包含下一個資料項的位置編碼“中斷”停止編碼來指示陣列或映射的結尾。“break”使用主要類型 7 和附加訊息值 31（0b111_11111）進行編碼，但它本身不是資料項：它只是關閉陣列或映射的語法功能。也就是說，“中斷”位於陣列或映射中的最後一項之後停止編碼，並且不能在其他任何地方代替資料項。以這種方式，無限長陣列和映射看起來與其他陣列和映射相同，除了以附加訊息值 31 開始並以“中斷”停止編碼結束。

具有不確定長度的陣列和映射允許在“中斷”停止編碼之前給出任意數量的項（對於陣列）和鍵/值對（對於映射）。巢狀無限長的陣列或映射項沒有限制。“中斷”僅終止單個項目，因此巢狀無限長的項目需要完全相同 “中斷”停止編碼，因為有一個類型位元組開始一個無限長的項目。

例如，假定編碼器想表示抽象陣列 [1，[2，3]，[4，5]。限定長編碼為 x8301820203820405：

```
83        -- Array of length 3
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   82     -- Array of length 2
      04  -- 4
      05  -- 5
```

無限長編碼可以根據需要獨立應用於此資料項中編碼的三個陣列中的每一個，從而產生如下表示：

```
0x9f018202039f0405ffff
   9F        -- Start indefinite-length array
      01     -- 1
      82     -- Array of length 2
         02  -- 2
         03  -- 3
      9F     -- Start indefinite-length array
         04  -- 4
         05  -- 5
         FF  -- "break" (inner array)
      FF     -- "break" (outer array)
```

```
0x9f01820203820405ff
9F        -- Start indefinite-length array
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   82     -- Array of length 2
      04  -- 4
      05  -- 5
   FF     -- "break"


0x83018202039f0405ff
83        -- Array of length 3
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   9F     -- Start indefinite-length array
      04  -- 4
      05  -- 5
      FF  -- "break"

0x83019f0203ff820405
83        -- Array of length 3
   01     -- 1
   9F     -- Start indefinite-length array
      02  -- 2
      03  -- 3
      FF  -- "break"
   82     -- Array of length 2
      04  -- 4
      05  -- 5
```

無限長映射（碰巧有兩個鍵/值對）的範例可能是：

```
0xbf6346756ef563416d7421ff
BF           -- Start indefinite-length map
   63        -- First key, UTF-8 string length 3
      46756e --   "Fun"
   F5        -- First value, true
   63        -- Second key, UTF-8 string length 3
      416d74 --   "Amt"
   21        -- -2
   FF        -- "break"
```

2.2.2. 無限長位元組字串和文本字串

無限長的位元組字串和文本字串實際上是零個或多個定長位元組或文本字串（“塊”）的串聯，它們一起被視為一個連續的字串。使用主要類型和附加訊息值 31 打開無限長字串，但接下來是一系列具有確定長度（塊）的位元組或文本字串。通過在將發

生系列中的下一個塊的位置編碼“中斷”停止編碼（0b111_11111）來指示該系列塊的結束。這些塊的內容連接在一起，無限長字串的總長度將是所有塊的長度之和。總之，無限長字串的編碼方式類似於如何編碼塊的無限長陣列，除了無限長字串的主要類型是（文本或位元組）字串的主要類型並匹配主要字串塊的類型。

對於無限長的位元組字串，無限長指示符和“中斷”之間的每個資料項（塊）必須是一個確定長度的位元組字串項目；如果解析器看到除了之外的任何項類型在看到“中斷”之前的位元組字串，這是一個錯誤。

例如，假設序列：

```
0b010_11111 0b010_00100 0xaabbccdd 0b010_00011 0xeeff99 0b111_11111
5F              -- Start indefinite-length byte string
   44           -- Byte string of length 4
      aabbccdd  -- Bytes content
   43           -- Byte string of length 3
      eeff99    -- Bytes content
   FF           -- "break"
```

解碼後，這會產生一個包含七個位元組的單一位元組字串：0xaabbccddeeff99。

具有無限長的文本字串與具有不確定長度的位元組字串相同，除了它們的所有塊必須是定義長文本字串之外。請注意，這意味著單個 UTF-8 字元的位元組不能在塊之間傳播：新的塊只能在字元邊界處開始。

2.3. 沒有內容的浮點數和值

主要類型 7 用於兩種類型的資料：浮點數和不需要任何內容的“簡單值”。初始位元組中的 5 位附加訊息的每個值都有其獨立的含義，如表 1 中所定義。與整數的主要類型一樣，此主要類型的項不攜帶內容資料；所有訊息都在初始位元組中。

5-Bit Value	Semantics
0..23	Simple value (value 0..23)
24	Simple value (value 32..255 in following byte)
25	IEEE 754 Half-Precision Float (16 bits follow)
26	IEEE 754 Single-Precision Float (32 bits follow)
27	IEEE 754 Double-Precision Float (64 bits follow)
28-30	(Unassigned)
31	"break" stop code for indefinite-length items

表 1：主要類型 7 的附加訊息值

與所有其他主要類型一樣，5 位元值為 24 表示單個位元組擴展：跟隨一個額外的位元組來表示簡單值。（為了最大限度地減少混淆，值只使用 32 到 255.）這樣可以保持初始位元組的結構：對於其他主要類型，這些類型的長度始終取決於第一個位元組中的附加訊息。表 2 列出了簡單類型分配和可用的值。

Value	Semantics
0..19	(Unassigned)
20	False
21	True
22	Null
23	Undefined value
24..31	(Reserved)
32..255	(Unassigned)

表 2：簡單值

25，26 和 27 中的 5 位元值分別用於 16 位元，32 位元和 64 位元 IEEE 754 二進制浮點數值。這些浮點數值被編碼在適當大小的附加位元組。（有關 16 位元浮點數的一些訊息，請參見附錄 D。）

2.4. 可選的項目標籤

在 CBOR 中，資料項可以選擇在標籤之後為其提供額外的語意，同時保留其結構。標籤是主要類型 6，並且表示由標籤的整數值指示的整數；（唯一的）資料項作為內容資料攜帶。如果標籤需要結構化資料，則此結構將編碼到巢狀資料項中。標籤的定義通常限制標籤可以攜帶哪種巢狀資料項。

標籤的初始位元組遵循正整數規則（主要類型 0）。標籤後跟任意類型的單個資料項。例如，假設標籤了長度為 12 的位元組字串帶有標籤以表明它是正面的大數（第 2.4.2 節）。這將被標籤為 0b110_00010（主要類型 6，標籤的附加訊息 2），然後是 0b010_01100（主要類型 2，長度為 12 的附加訊息），接著是大數的 12 個位元組。

解碼器不需要理解標籤，因此在創建特定 CBOR 資料項的實施和完成解碼該流實施解碼的應用中，標籤可能沒什麼價值，知道資料串流中每個項的語意含義。它們在本規範中的主要目的是定義通用資料類型比如日期。次要目的是當解碼器是通用 CBOR 解碼器時允許可選標籤，該解碼器可能受益於關於項目內容的提示。理解語意標籤對於解碼器是可選的；它可以跳過標籤的初始位元組並解釋標籤的資料項本身。

標籤始終應用於直接跟隨其後的項目。因此，如果標籤 A 後為標籤 B，然後是資料項 C，則標籤 A 適用於將標籤 B 應用於資料項 C 的結果。也就是說，標籤項是由標籤和標籤組成的資料項值。被標籤項的內容是被標籤的資料項（值）。

IANA 維護標籤值的註冊表，如第 7.2 節所述。表 3 提供了初始值列表，其中包含本節其餘部分中的定義。

Tag	Data Item	Semantics
0	UTF-8 string	Standard date/time string; see Section 2.4.1
1	multiple	Epoch-based date/time; see Section 2.4.1
2	byte string	Positive bignum; see Section 2.4.2
3	byte string	Negative bignum; see Section 2.4.2
4	array	Decimal fraction; see Section 2.4.3
5	array	Bigfloat; see Section 2.4.3
6..20	(Unassigned)	(Unassigned)
21	multiple	Expected conversion to base64url encoding; see Section 2.4.4.2
22	multiple	Expected conversion to base64 encoding; see Section 2.4.4.2
23	multiple	Expected conversion to base16 encoding; see Section 2.4.4.2
24	byte string	Encoded CBOR data item; see Section 2.4.4.1
25..31	(Unassigned)	(Unassigned)
32	UTF-8 string	URI; see Section 2.4.4.3
33	UTF-8 string	base64url; see Section 2.4.4.3
34	UTF-8 string	base64; see Section 2.4.4.3
35	UTF-8 string	Regular expression; see Section 2.4.4.3
36	UTF-8 string	MIME message; see Section 2.4.4.3
37..55798	(Unassigned)	(Unassigned)
55799	multiple	Self-describe CBOR; see Section 2.4.5
55800+	(Unassigned)	(Unassigned)

表3：標籤的值

2.4.1. 日期和時間

標籤值 0 用於遵循 [RFC3339]中描述的標準格式的日期/時間字串，由[RFC4287]的第 3.3 節改進。

標籤值 1 用於以數字形式表示，相對於 UTC 時間 1970-01-01T00：00Z 的秒數。（對於可移植操作系統介面（POSIX）定義的非負值，秒數的計數方式與 POSIX“自紀元以來的秒數”[TIME_T]相同。）標籤的項目可以是正數或負整數（主要類型 0 和 1），或浮點數（具有附加訊息 25,26 或 27 的主要類型 7）。注意這個數字可以是負數（1970-01-01T00：00Z 之前的時間），如果是浮點數，則表示小數秒。

2.4.2. 大數

大數是不適合由基本整數中主要類型 0 和 1 提供表示形式的整數。它們被編碼為位元組字串資料項，其被解釋為網路位元組順序中的無符號整數 n。對於標籤值 2，大數的值為 n。對於標籤值 3，大數的值為-1-n。理解這些標籤的解碼器必須能夠解碼前導數含零的大數 。

例如，編號 18446744073709551616（2 ** 64）表示為 0b110_00010（主要類型 6，標籤 2），後跟 0b010_01001（主要類型 2，長度 9），後跟 0x0100000000000000（一個位元組 0x01 和八個位元組 0x00）。以十六進制表示：

```
C2                      -- Tag 2
   29                   -- Byte string of length 9
      0100000000000000  -- Bytes content
```

2.4.3. 十進制小數和大浮點數

十進制小數將整數尾數與以 10 為底的比例因子結合在一起。如果應用程式需要十進制小數的精確表示（例如 1.1），它們是最有用的，因為在二進制浮點數中沒有精確表示多個十進制小數。

大浮點數將整數尾數與以 2 為基底的比例因子結合在一起。它們是二進制浮點值，可能超出 CBOR 支援的三種 IEEE 754 格式的

範圍或精度（第 2.3 節）。大浮點數也可以用於需要一些基本二進制浮點功能而無需支援 IEEE 754 的受限應用程式中。

十進制小數或大浮點數表示為包含正好兩個整數的標籤陣列：指數 e 和尾數 m。十進制分數（標籤 4）使用基數為 10 的指數；十進制小數資料項的值是 m *（10 ** e）。大浮點數（標籤 5）使用基數為 2 的指數； 大浮點數資料項的值是 m *（2 ** e）。指數 e 必須用主要類型 0 或 1 的整數表示，而尾數也可以是大數（第 2.4.2 節）。

小數的一個例子是，數字 273.15 可以表示為 0b110_00100（標籤的主要類型為 6，標籤類型的附加訊息為 4），然後通過用於陣列 0b100_00010（陣列的主要類型為 4，陣列長度的附加訊息為 2），接著為第一個整數 0b001_00001（第一個整數的主要類型為 1，值為-2 的附加訊息為 1），然後 0b000_11001（第二個整數的主要類型為 0，兩個位元組值的主要類型為 25）中，隨後 0b0110101010110011（27315 在兩個位元組）。在十六進制：

```
C4             -- Tag 4
   82          -- Array of length 2
      21       -- -2
      19 6ab3  -- 27315
```

一個例子是有一個大浮點數，數字 1.5 可以表示 0b110_00101（標籤的主要類型為 6，標籤類型的附加信息為 5），其次是 0b100_00010（陣列的主要類型為 4，陣列長度的附加訊息為 2），後跟 0b001_00000（第一個整數的主要類型為 1，值為-1 的其他類型為 0），後跟 0b000_00011（第二個整數的主要類型為 0，值為 3 的附加訊息為 3）。十六進制：

```
C5             -- Tag 5
   82          -- Array of length 2
      20       -- -1
      03       -- 3
```

十進制小數和大浮點數不提供無窮大，負無窮大或 NaN 的表示；如果需要這些代替小數或大浮點數，可以使用第 2.3 節中的 IEEE 754 半精確度表示。對於受約束的應用程式，可以選擇將特定數字表示為整數，還是作為十進制小數或大浮點數（例如當指數很小且非負數）時， 期望實施質量直接使用整數表示。

2.4.4. 內容提示

本節中的標籤用於通用可能由 CBOR 處理器使用的內容提示。

2.4.4.1. 編碼的CBOR資料項

有時攜帶嵌入式CBOR資料項是有益的，該資料項不是在解析封閉資料項時立即解碼的。標籤24（CBOR資料項）可用於將嵌入的位元組字串標籤為以CBOR格式編碼的資料項。

2.4.4.2. CBOR到JSON轉換器的預期後期編碼

標籤 21 到 23 表示當與基於文本的表示相互操作時，位元組字串可能需要特定的編碼。當編碼器知道它正在寫入的位元組字串資料可能稍後轉換為特定的基於JSON的用法時,這些標籤很有用。該用法指定某些字串編碼為 base64，base64url 等。編碼器使用位元組字串而不是編碼本身來減少訊息大小，減少編碼器的編碼大小，或兩者兼而有之。編碼器不知道轉換器是否是通用的，因此想要說出它認為將二進制字串轉換為 JSON 的正確方法。

標籤的資料項可以是位元組字串或任何其他資料項。在後一種情況下，標籤適用於資料項中包含的所有位元組字串資料項，但標籤有預期轉換的巢狀資料項中包含的資料項除外。

這三種標籤類型建議轉換為[RFC4648]中定義的三種基本資料編碼。對於 base64url 編碼，不使用填充（參見 RFC 4648 的第 3.2 節）；也就是說，所有尾隨等於標誌（“=”）將從 base64url 編碼的字串中刪除。可以為 RFC 4648 的其他資料編碼定義後來的標籤，或者為字串中的二進制資料編碼的其他方式定義。

2.4.4.3. 編碼文本

一些文本字串保存具有在網際網路上廣泛使用的格式的資料，並且有時這些格式可以由解碼器以適當的形式被驗證並呈現給應用程式。其中一些格式有標籤。

- 標籤 32 用於 URI，如[RFC3986]中所定義；

- 標籤 33 和 34 用於 base64url 和 base64 編碼的文本字串，如[RFC4648]中所定義；

- 標籤 35 用於 Perl 相容正規表達式（PCRE）/ JavaScript 語法[ECMA262]中的正規表達式。

- 標籤 36 用於 MIME 訊息（包括所有標題），如[RFC2045]中所定義；

注意，標籤 33 和 34 與 21 和 22 的不同之處在於，前者以基本編碼形式傳輸資料，後者以原始位元組字串形式傳輸資料。

2.4.5. CBOR 的自我描述

在許多應用中，從上下文中可以清楚地看出，CBOR 被用於編碼資料項。例如，特定協定可以指定 CBOR 的使用，或者指示指定其使用的媒體類型。然而，可能存在這樣的上下文訊息不可用的應用，例如當 CBOR 資料存儲在文件中並且消除歧義元資料未被使用時。在這裡，它可能有助於為資料本身提供一些區別特徵。為此定義了標籤 55999。它不會在隨後的資料項上賦予任何特殊的語意；也就是說，標記為 55799 的資料項的語意與資料項本身的語意完全相同。

此標籤的序列化為 0xd9d9f7，它似乎不用作常用文件類型的區分標籤。特別是，如果後面跟隨有效的 CBOR 資料項，則它不採用任何 Unicode 編碼的 Unicode 文本。

例如，解碼器可能能夠解析 CBOR 和 JSON。這種解碼器需要機械地區分這兩種格式。編碼器幫助解碼器的一種簡單方法是使用標籤 55799 標籤整個 CBOR 項，其序列化將永遠不會在 JSON 文本的開頭。

3. 創建基於 CBOR 的協定

諸如 CBOR 之類的資料格式通常用於沒有格式協商的環境中。CBOR 的一個特定設計目標是不需要任何包含或假定的模式：解碼器可以採用 CBOR 項目並在沒有其他知識的情況下對其進行解碼。

當然，在實際實施中，編碼器和解碼器將具有 CBOR 資料項中應該具有的共享視圖。例如，一致同意的格式可能是“該項是一個陣列，其第一個值是 UTF-8 字串，第二個值是一個整數，後續值是零個或多個浮點數”或“該項是一個具有鍵的位元組字串的映射，並且包含至少一對鍵為“0xab01”的對。

該規範對基於 CBOR 的協定沒有限制。編碼器能夠編碼使用它的協定所需的多種或幾種類型的值；解碼器能夠理解使用它的協定所需的多種或幾種類型的值。缺乏限制使得 CBOR 可以在極其受限的環境中被使用。

本節討論創建基於 CBOR 的協定的一些考量。它只是建議性的，並且明確地排除 RFC 2119 中的任何語言，而不是 RFC 2119 意義上可以解釋為“MAY”的單詞。

3.1. 串流應用中的 CBOR

在串流應用程式中，資料串流可以由一系列背靠背連接的 CBOR 資料項組成。在這樣的環境中，如果在先前資料項結束之後找到資料，則解碼器立即開始解碼新資料項。

並非構成資料項的所有位元組都可以立即用於解碼器；一些解碼器將緩衝其他資料，直到可以向應用程式呈現完整的資料項。其他解碼器可以向應用程式呈現關於頂級資料項的部分訊息，例如可能已經被解碼的巢狀資料項，或者甚至尚未完全到達的位元組字串的部分。

請注意，某些應用程式和協定不會想用無限長編碼。使用無限長編碼允許編碼器不需要編組用於計數的所有資料，但是它需要解碼器在等待項目結束時分配越來越多的內存。這對於某些應用程式可能是好的，但對於其他應用程式則不是。

3.2. 通用編碼器和解碼器

通用 CBOR 解碼器可以解碼所有格式良好的 CBOR 資料並將它們呈現給應用程式。如果 CBOR 資料以 CBOR 定義的方式使用初始位元組以及由其值隱含的位元組字串和/或資料項，則 CBOR 資料格式良好，並且不會出現無關資料（附錄 C）。

儘管 CBOR 試圖最小化這些情況，但並非所有格式良好的 CBOR 資料都是有效的：例如，
該格式不包括使用擴展位元組編碼的 32 以下的簡單值。此外，特定標籤可能會構成可能會違反的語意約束，例如通過在大數標籤中包括標籤或者通過跟隨日期標籤內的位元組字串。最後，資料可能無效，例如無效的 UTF-8 字串或不符合[RFC3339]的日期字串。通用編碼器和解碼器不需要為其應用程式介面做出不自然的選擇，以便能夠處理無效資料。即使在寫入編碼器/解碼器時未註冊其特定碼點，通用編碼器和解碼器也應轉發簡單值和標籤（第 3.5 節）。

通用解碼器提供了向應用程式呈現格式良好的 CBOR 值（有效和無效）的方法。偵錯符號（第 6 節）可用於向人類呈現良好形成的 CBOR 值。

通用編碼器提供了一個應用程式介面，允許應用程式指定任何格式良好的值，包括編碼器未知的簡單值和標籤。

3.3. 語法錯誤

遇到格式不正確的 CBOR 資料項的解碼器通常可以選擇完全讓解碼失敗（發出錯誤和/或完全停止處理），使用明確指示有問題的資料和資料項解碼器以特定約定來替換。明確表示存在問題或採取了其他措施。

3.3.1. 不完整的 CBOR 資料項

CBOR 資料項的表示具有特定長度，由其初始位元組和資料項中包含的任何資料項的結構確定。如果可用的資料較少，則可將其視為語法錯誤。解碼器還可以實施增量解析，即在資料項可用時對資料項進行解碼並呈現到目前為止找到的資料（例如在基於事件的介面中），並且可以選擇在進一步資料時繼續解碼。

不完整資料項的範例包括：

- 解碼器需要一定數量的陣列或映射條目，但遇到資料結尾。
- 解碼器處理它期望成為映射中的最後一對並到達資料末尾的內容。
- 解碼器剛剛看到一個標籤然後遇到資料的結尾。
- 解碼器已經看到了一個無限長項的開始，但計數到資料的結尾，在看到“中斷”停止代碼之前

3.3.2. 格式無限長項

格式錯誤的無限長資料項的範例包括：

- 在無限長位元組字串或文本中，解碼器在找到“中斷”停止碼之前先找到不是適當主要類型的項目。
- 在無限長的映射中，解碼器在讀取密鑰後立即遇到“中斷”停止編碼（缺少值）。

另一個錯誤是在資料中的某個點找到“中斷”停止編碼，其中沒有立即封閉（未閉合）的無限長項。

3.3.3. 未知的附加訊息值

在撰寫本文時，未分配一些其他訊息值，並為本文的未來版本保留（參見第 5.2 節）。由於尚未定義這些附加訊息值的整體語法，因此看到其不理解的附加訊息值的解碼器不能繼續解析。

3.4. 其他解碼錯誤

CBOR 資料項可以在語法上良好地形成，但是在 CBOR 資料模型中解釋在其中編碼的資料存在問題。一般來說，找到具有此類問題的資料項的解碼器可能會發出警告，可能會完全停止處理，可能會處理錯誤並使問題值可用於應用程式，或採取其他類型的操作。

這些問題可能包括：

映射中的重複鍵：通用解碼器（第 3.2 節）使用本機 CBOR 資料模型使應用程式可以使用資料。該資料模型包括映射（具有唯一鍵的鍵值映射），而不是多映射（鍵值映射，其中多個條目可以具有相同的鍵）。因此，獲得具有重複鍵的 CBOR 映射項的通用解碼器將解碼為僅具有該鍵的一個實例的映射，或者它可能完全停止處理。另一方面，“流解碼器”甚至可能無法注意到（第 3.7 節）。

標籤後面的值不允許使用類型：標籤（第 2.4 節）指定應該在標籤後面跟什麼類型的資料項；例如，正或負 bignums 的標籤應該放在位元組字串上。將標籤資料項解碼為本機表示（在該範例中為本機大整數）的解碼器期望檢查被標籤的資料項的類型。即使在其環境中沒有這種本機表示的解碼器也可以對它們已知的那些標籤執行檢查並且適當地做出反應。

無效的 UTF-8 字串：解碼器可能或可能不想驗證 UTF-8 字串（主要類型 3）中的位元組序列是否實際上是有效的 UTF-8 並作出適當的反應。

3.5. 處理未知的簡單值和標籤

遇到無法識別的簡單值（第 2.3 節）的解碼器，例如在部署解碼器後添加到 IANA 註冊表的值或解碼器選擇不實施的值，可能會發出警告，可能會完全停止處理，可能會通過使應用程式可以使

用未知值（如通用解碼器所期望的那樣）來處理錯誤，或採取其他類型的操作。

遇到無法識別的標籤（第 2.4 節）的解碼器，例如在部署解碼器後添加到 IANA 註冊表的標籤或解碼器選擇不執行的標籤，可能會發出警告，完全停止處理，可能會處理錯誤並將未知標籤值與包含的資料項一起呈現給應用程式（如通用解碼器所期望的那樣），可能會忽略標籤並僅將包含的資料項僅呈現給應用程式，或者一些其他類型的行動。

3.6. 數字

出於本說明書的目的，相同數值的所有數字表示是等效的。這意味著編碼器可以將浮點值 0.0 編碼為整數 0.但是，這也意味著，如果編碼器決定這些值是合乎需要的，那麼期望找到整數值的應用程式可能會找到浮點值,例如當浮點值比 64 位整數更緊湊時。

使用 CBOR 的應用程式或協定可能會限制數字的表示。例如，僅處理整數的協定可能會說可能不會使用浮點數，並且該協定的解碼器不需要能夠處理浮點數。類似地，使用 CBOR 的協定或應用程式可能會說解碼器需要能夠處理任何類型的數字。

基於CBOR的協定應考慮到不同的語言環境對可表示的數字的範圍和精確度有不同的限制。例如，JavaScript 數字系統將所有數字視為浮點數，這可能導致解碼具有超過 53 個有效位元的整數時的靜默精確度損失。使用數字的協定應該定義其對解碼器和接收應用程式中非普通數字處理的期望。

包含浮點數的基於CBOR的協定可以限制支援三種格式中的哪一種（半精確度，單精確度和雙精確度）。對於僅整數的應用程式，協定可能希望完全排除浮點值的使用。

專為緊湊而設計的基於CBOR的協定可能希望排除比應用程式所需時間更長的特定整數編碼，例如節省實施 64 位整數的需要。期望編碼器將使用可以表示給定值的最緊湊的整數表示。但是，只要應用程式可以解碼給定大小的整數，緊湊型應用程式就應該

接受使用長於所需編碼的值（例如將“0”編碼為 0b000_11101，後跟兩個位元組 0x00）。

3.7. 指定映射的鍵

編碼和解碼應用程式需要就將在映射中使用哪些類型的密鑰達成一致。在需要與基於 JSON 的應用程式交互的應用程式中，密鑰可能僅限於 UTF-8 字串；否則，必須存在從其他 CBOR 類型到 Unicode 字元的指定映射，這通常會導致實施錯誤。在鍵本質上是數字的應用程式中，鍵的數字排序對應用程式很重要，直接使用鍵的數字是很有用的。

如果要使用多種類型的密鑰，則應考慮如何在要使用的特定編程環境中表示這些類型。例如，在 JavaScript 對像中，整數 1 的鍵不能與字串“1”的鍵區分開。這意味著，如果使用整數鍵，則需要同時使用看起來像數字的字串鍵。同樣，這導致了鍵應該是單個 CBOR 類型的結論。

在解碼它們時立即傳送巢狀在 CBOR 資料項內的資料項的解碼器（“串流解碼器”）通常不保持確定映射中的鍵的唯一性所必需的狀態。類似地，可以在封閉資料項完全可用之前開始編碼資料項的編碼器（“串流編碼器”）可能希望通過依賴其資料源來保持唯一性來顯著降低其開銷。

當接收應用程式確實在映射中看到多個相同的密鑰時，基於 CBOR 的協定應該做出有意識的決定。協定中產生的規則應該遵循 CBOR 資料模型：它不能規定條目的特定處理。

使用相同的鍵，除了它可能有一個規則，在映射中具有相同的鍵表示格式錯誤的映射，並且解碼器必須停止並出現錯誤。使用嚴格模式的 CBOR 解碼器也禁止重複鍵（第 3.10 節）。.

映射的 CBOR 資料模型不允許將語意歸因於映射表示中的鍵/值對的順序。因此，以這樣一種方式定義基於 CBOR 的協定將是一種非常糟糕的做法：除了普通方面（快取使用等）之外，改變映

射中的鍵/值對順序將改變語意。（基於 CBOR 的協定可以規定特定的序列化順序，例如規範化。）

具有 24 個或更少常用鍵的映射所受約束設備的應用需考慮使用小的整數（並且具有多達 48 個常用鍵的也應考慮使用小的負整數），因為可以在單個位元組中對密鑰進行編碼。

3.8. 未定義的值

在基於一些CBOR的協定中，編碼器可以使用未定義的簡單值（第2.3 節）作為具有編碼問題的資料項的替代，以便允許對其餘的封閉資料項進行編碼而不會造成傷害。

3.9. 典型 CBOR

某些協定可能希望編碼器僅以特定的規範格式發出 CBOR；這些協定也可能讓解碼器檢查它們的輸入是否合乎規範。這些協定可以通過規範格式自由定義它們的含義以及預期編碼器和解碼器應該做什麼。本節列出了此類協定的一些建議。

如果協定認為“規範”意味著以相同輸入資料開始的兩個編碼器實施將產生相同的 CBOR 輸出，則以下四個規則就足夠了：

- 整數必須盡可能小。
  - 0 至 23 和-1 至-24 必須在與主要類型相同的位元組中表示；
  - 24 至 255 和-25 至-256 必須僅使用額外的 uint8_t 表示；
  - 256 至 65535 和-257 至-65536 必須僅使用額外的 uint16_t 表示；
  - 65536 至 4294967295 和-65537 至-4294967296 必須僅使用額外的 uint32_t 表示。

- 主要類型 2 到 5 的長度表達必須盡可能短。這些長度的規則遵循上述整數規則。

- 每個映射中的鍵必須從最低值到最高值排序。對關鍵資料項表示的位元組執行排序，而無需注意主要類型的 3/5 位元拆分。（請注意，此規則允許具有不同類型的鍵的映射，即使這可能是一種可能導致某些規格化中發生出錯的錯誤做法。）排序規則是：

  * 如果兩個鍵的長度不同，則較短的鍵會更早排序；

  * 如果兩個鍵的長度相同，則按（逐位元組）詞法順序值較低的鍵會更早排序。

- 無限長項必須做成定長度項。

如果協定允許 IEEE 浮點數，則可能需要添加其他規範化規則。一個範例規則可能是將所有浮點數作為 64 位元浮點數啟動，然後將測試轉換為 32 位元浮點數；如果結果是相同的數值，請使用較短的值並重複該過程，並將測試轉換為 16 位元浮點數。（此規則也為正負無窮大選擇 16 位元浮點數。）此外，NaN 有許多表示形式。如果 NaN 是允許值，則必須始終表示為 0xf97e00。

CBOR 標籤為規範化提供了額外的考量。規範格式標籤的存在與否由協定中標籤的可選性決定。在基於 CBOR 的協定中，允許在任何地方使用可選標籤，規範格式不得允許它們。在某些需要標籤的協定中，標籤需要以規範格式顯示。使用規範化的基於 CBOR 的協定可能會反過來說，無論訊息是否是可選的，都必須保留訊息中出現的所有標籤。

3.10. 嚴格的模式

CBOR 的某些應用領域不需要規範化（第 3.9 節），但可能要求不同的解碼器達到相同（語意上等效）的結果，即使存在潛在的惡意資料。如果一個應用程式（例如防火牆或其他保護實體）根

據另一個獨立解碼資料的應用程式所依賴的資料做出決策，則可能需要這樣做。

通常，發件人有責任避免模糊解碼資料。但是，發送方可能是特別編制 CBOR 資料的攻擊者，因此不同的解碼器將對其進行不同的解釋，以試圖將其作為漏洞利用。在可能存在問題的應用中使用的通用解碼器需要支援嚴格模式，其中接收器也有責任拒絕模糊可解碼資料。預計解碼 CBOR 的防火牆和其他安全系統只能在嚴格模式下解碼。

嚴格模式的解碼器將可靠地拒絕任何可由其他解碼器以不同方式解釋的資料。它將可靠地拒絕具有語法錯誤的資料項（第 3.3 節）。它還將花費精力來可靠地檢測其他解碼錯誤（第 3.4 節）。特別是，嚴格的解碼器需要有一個 API，它報告包含以下任何內容的 CBOR 資料項的錯誤（並且不返回資料）：

- 不只一個具有相同鍵的入口點的映射（主要類型 5）
- 在不正確類型的資料項上使用的標籤
- 為給定的類型格式不正確的資料項，例如無效的 UTF-8 或無法使用已標籤的特定標籤解釋的資料

為給定的類型格式不正確的資料項，例如無效的 UTF-8 或無法使用已標籤的特定標籤解釋的資料

- 它可以報告錯誤（而不是返回資料）。
- 它可以向調用解碼器的應用程式發出未知項（類型，值和標籤，解碼的標籤資料項），並指示解碼器不識別該標籤或簡單值。

後一種方法也適用於非嚴格的解碼器，支援與新註冊的標籤和簡單值的向前兼容，而無需與調用應用程式同時更新編碼器。（為此，解碼器的 API 需要有一種標籤未知項的方法，以便調用應用程式可以以適合該程式的方式處理它們。）

由於某些處理可能具有可觀的成本（特別是對於映射的重複檢測），因此不要求對所有 CBOR 解碼器提供嚴格模式的支援。

一些編碼器將依賴其應用程式以明確可解碼的CBOR結果的方式提供輸入資料。通用編碼器還可能希望提供一種嚴格的模式，在該模式下，它將可靠地將其輸出限制為明確可解碼的 CBOR，而與其應用程式是否提供符合 API 的資料無關。

4. 在 CBOR 和 JSON 之間轉換資料

本節提供有關 CBOR 和 JSON 之間轉換的非規範性建議。轉換器的實施可以隨意使用他們想要的任何建議。

值得注意的是，JSON 文本是字元序列，而不是編碼的位元組序列，而 CBOR 資料項由位元組而不是字元組成。

4.1. 從 CBOR 轉換為 JSON

CBOR 中的大多數類型都有 JSON 中的直接類比。但是，有些沒有，而實施 CBOR-to-JSON 轉換器的人必須考慮在這種情況下該怎麼做。以下非規範性建議通過將它們轉換為單個替換值（例如 JSON null）來處理這些建議。

- 整數（主要類型 0 或 1）成為 JSON 編號。
- 未嵌入指定建議編碼的標籤中的位元組字串（主要類型 2）在 base64url 中編碼而沒有填充，並成為 JSON 字串。
- UTF-8 字串（主要類型 3）成為 JSON 字串。請注意，JSON 需要轉譯某些字元（RFC 4627，第 2.5 節）：引號（U + 0022），反向實線（U + 005C）和“C0 控制字元”（U + 0000 到 U + 001F）。所有其他字元將未更改地複製到 JSON UTF-8 字串中。
- 陣列（主要類型 4）成為 JSON 陣列。

- 映射（主要類型 5）成為 JSON 物件。僅當所有鍵都是 UTF-8 字串時才可以直接執行此操作。轉換器也可以將其他鍵值轉換為 UTF-8 字串（例如通過將整數轉換為包含其十進制表示的字串）；然而，這樣做會帶來鍵值碰撞的危險。

- False（主要類型 7，附加訊息 20）變為 JSON false。

- True（主要類型 7，附加訊息 21）變為 JSON true。

- Null（主要類型 7，附加訊息 22）變為 JSON null。

- 浮點值（主要類型 7，附加訊息 25 到 27）如果是有限的（即，它可以用 JSON 數字表示），則變為 JSON 數字；如果該值是非有限的（NaN，或正或負無窮大），則由替換值表示。

- 任何其他簡單值（主要類型 7，尚未討論的任何其他訊息值）由替換值表示。

- 大數（主要類型 6，標籤值 2 或 3）通過在 base64url 中編碼其位元組字串而不填充來表示，並成為 JSON 字串。對於標籤值 3（負大數），在基本編碼值之前插入“～”（ASCII '~' 字元）。（轉換為二進制大型物件而不是數字是為了防止 JSON 解碼器可能出現數字溢位。）

- 帶有編碼提示的位元組字串（主要類型 6，標籤值 21 到 23）按照描述進行編碼，並成為 JSON 字串。

- 對於所有其他標籤（主要類型 6，任何其他標籤值），嵌入的 CBOR 項表示為 JSON 值；標籤值被忽略。

- 轉換前確定無限長項。

4.2. 從 JSON 轉換為 CBOR

在建議的轉換中：所有的 JSON 值，一旦解碼，直接映射到一個或多個 CBOR 值。與任何類型的 CBOR 生成一樣，必須就數字表示形式做出決策。

- 沒有十進制小數的 JSON 數字（整數）表示為整數（主要類型 0 和 1，可能是主要類型 6 標籤值 2 和 3），選擇最短形式；長度大於實施定義的閾值（通常為 32 位元或 64 位元）的整數可以表示為浮點值。（如果 JSON 是從 JavaScript 生成的，則其精確度已限制為最多 53 位元。）

- 帶十進制小數的數字表示為浮點值。優先地，使用最短精確浮點數表示；例如，1.5 以 16 位元浮點值表示（但並非所有實施都能夠有效地找到最小形式）。可能存在實施定義的精確度限制，這將限制所表示的值的精確度。只有在協定中指定的十進制表示才可使用。

CBOR 的設計通常提供比 JSON 更緊湊的編碼。可能會想到的一種實施策略是在單個緩衝區中執行 JSON 到 CBOR 編碼。這種策略需要仔細考慮許多病態情況，例如，當在 CBOR 中編碼為 UTF-8 字串時，某些字串表示沒有或很少有逸出，並且比 255 位元組長（或更長）可能會擴展。類似地，一些二進制浮點表示可能會導致 JSON 中的一些短十進制表示（1.1,1e9）擴展。這可能很難做到，任何隨之而來的漏洞都可能被攻擊者利用。

5. CBOR 的未來演變

成功的協定隨著時間而演變。出現了新的想法，改進了實施平台，開發和發展了相關協定，並增加了應用程式和協定的新要求。因此，促進協定演進是任何協定開發的重要設計考量。

對於將使用 CBOR 的協定，CBOR 提供了一些有用的機制來促進它們的發展。對此的最佳實踐是眾所周知的，尤其是基於 JSON 的協定的 JSON 格式開發。因此，此類最佳實踐超出了本規範的範圍。

然而，促進 CBOR 本身的發展非常適合其範圍。CBOR 旨在為基於 CBOR 的協定開發提供穩定的基礎並且能夠發展。

由於成功的協定可能存在數十年，因此 CBOR 需要設計數十年才能使用和發展。本節為 CBOR 的演變提供了一些指導。它必然比本文的其他部分更主觀。它也必然是不完整的，以免它變成協定開發的教科書。

5.1. 擴展點

在協定設計中，進化的機會通常以擴展點的形式包括在內。例如，可能存在未從一開始就完全分配的碼點空間，並且該協定旨在容忍並接受開始使用比最初分配的碼點更多的碼點的實施

確定碼點空間的大小可能很困難，因為所需的範圍可能難以預測。應該嘗試使碼點空間足夠大，以便可以在協定的預期生命週期內慢慢填充它。

CBOR 有三個主要延伸點：

- “簡單”空間（主要類型 7 中的值）。在 24 個有效（和 224 個效率稍低）值中，只分配了一小部分。接收未知簡單資料項的實施可能能夠如此處理它，因為值的結構確實很簡單。第 7.1 節中的 IANA 註冊表是解決此碼點空間可擴展性的適當方法。

- “標籤”空間（主要類型 6 中的值）。同樣，只分配了一小部分碼點空間，並且空間很大（儘管早期數字比後者更有效）。接收未知標籤的實施可以選擇簡單地忽略它或將其作為包裝以下資料項的未知標籤進行處理。第 7.2 節中的 IANA 註冊表是解決此碼點空間可擴展性的適當方法。

- 附加訊息空間。接收到未知附加訊息值的實施無法繼續解析，因此將碼點分配給此空間是一個重要步驟。還剩下很少的碼點。

5.2. 策劃附加訊息空間

人類的思想有時被吸引到填補一點點感知的空隙，以使一些東西整潔。我們希望碼點空間中的剩餘空白可以作為新想法的吸引子，因為它們就在那裡。

本規範不管理 IANA 註冊中心的附加訊息碼點空間。相反，只能通過更新此規範來完成此空間之外的分配。

對於 n >= 24 的附加訊息值，附加資料的大小通常是 2 **（n-24）位元組。因此，如果需要將附加訊息值 28 和 29 添加到協定中，則應將其視為 128 位元和 256 位元的候選值。然後，附加訊息值 30 是可用於一般分配的唯一附加訊息值，並且在通過更新該協定進行分配之前應該有非常好的理由來分配它。

6. 偵錯表示法

CBOR 是二進制交換格式。為了便於記錄和除錯，特別是為了促進在除錯中合作的實體之間的通信，本節定義了一種簡單的人類可讀偵錯符號。所有實際的交換總是以二進制格式發生。

請注意，這確實是一種偵錯格式；它並不意味著被解析。因此，本文中未給出正式定義（如 ABNF 中所述）。（尋找基於文本的格式以在配置文件中表示 CBOR 資料項的實施者也可能想要考慮 YAML [YAML]。）

偵錯符號大致上是基於 RFC 4627 中定義的 JSON，並在需要時進行擴展。

符號借用了 JSON 語法的數字（整數和浮點），True（> true <），False（> false <），Null（> null <），UTF-8 字串，陣列和映射（映射稱為物件）在 JSON 中；偵錯符號通過允許鍵位置中的任何資料項來擴展 JSON。未定義是在 JavaScript 中編寫的 > undefined <。非有限浮點數無窮大，負無窮大和 NaN 與這句話完全相同(這也是他們可以用 JavaScript 編寫的一種方式，儘管 JSON 不允許它們)。標籤的項目寫為標籤的整數，後跟括號中的項目；例如，RFC 3339（ISO 8601）日期可以標籤為：

0("2013-03-21T20:04:00Z")

或等效的相對時間
1(1363896240)

位元組字串在其中一個基本編碼中標籤，沒有填充，用單引號括起來，前綴為> h <表示 base16，> b32 <表示 base32，> h32 <表示 base32hex，> b64 <表示 base64 或 base64url（實際編碼）不重疊，所以字串保持明確）。例如，位元組字串 0x12345678 可寫為 h'12345678'，b32'CI2FM6A'或 b64'EjRWeA'。

未分配的簡單值以“simple（）”的形式給出，括號中有適當的整數。例如，“simple（42）”表示主要類型 7，值 42。

6.1. 編碼指標

有時在偵錯符號中指出實際使用了哪種替代表示形式中的哪一種很有用；例如，偵錯解碼器寫入> 1.5 <的資料項可能已被編碼為半精確度，單精確度或雙精確度浮點數。

編碼指標的規約是，以下劃線開頭的所有後續字母數字或下劃線字符都是編碼指標，並且對此訊息不感興趣的任何人都可以忽略。編碼指標始終是可選的。

可以在映射的左括號或陣列的左括號之後寫入單個下劃線，以指示資料項以無限長格式表示。例如，[_ 1,2]包含一個指標，表示使用無限長表示來表示資料項[1,2]。

下劃線後跟十進制數字 n 表示前一項（或對於數組和映射，該項從前括號或大括號開始）已使用附加訊息值 24 + n 進行編碼。例如，1.5_1 是一個半精確度浮點數，而 1.5_3 被編碼為雙精確度。這種編碼指示不在附錄 A 中出現（注意，編碼指標“_”是這樣完整形式“_7”，這是不使用的縮寫。）

作為一種特殊情況，無限長的位元組和文本字串可以以（_ h'0123'，h'4567'）和（_“foo”，“bar”）的形式表示。

7. IANA 考量

IANA 為新的 CBOR 價值創建了兩個註冊管理機構。註冊管理機構是獨立的，即不在傘式註冊管理機構之下，並遵循[RFC5226]中的規則。IANA 還分配了新的 MIME 媒體類型和相關的約束應用協定（CoAP）內容格式條目。

7.1. 簡單值註冊表

IANA 創建了“簡明二進制物件表示法(CBOR)簡單值”註冊表。初始值如表 2 所示。

標準操作分配 0 到 19 範圍內的新條目。建議這些標準操作分配從數字 16 開始的值，以便為連續的塊（如果有的話）保留較低的數字。

規範要求指定範圍為 32 到 255 的新條目。

7.2. 標籤註冊表

IANA 創建了“簡明二進制物件表示（CBOR）標籤”註冊表。初始值如表 3 所示。

標準操作分配 0 到 23 範圍內的新條目。24 到 255 範圍內的新條目由“需要規範”指定。256 至 18446744073709551615 範圍內的新條目由“先到先得”分配。註冊請求的模板是：

- 資料項
- 語意學（簡寫）

此外，先到先得的要求應包括：

- 聯繫點
- 語意描述（URL）此描述是可選的；該 URL 可以指向網際網路草案或網頁似的內容。

7.3. 媒體類型（“MIME 類型”）

用於 CBOR 資料的網際網路媒體類型[RFC6838]是應用程式/cbor。

型態名稱：應用程式
子類型名稱：cbor
必需參數：不適用
可選參數：n / a
編碼考量：二進制
安全考量：請參閱本文的第 8 節
互操作性考慮：不適用
發布規範：本文

使用此媒體類型的應用程式：尚未使用，但預計此格式將部署在協定和應用程式中。

附加訊息：
幻數：不適用
文件擴展名：.cbor
Macintosh 文件類型編碼：n / a
聯繫人和電子郵件地址以獲取更多訊息：
Carsten Bormann cabo@tzi.org
預期用途：COMMON
使用限制：無
作者：
Carsten Bormann <cabo@tzi.org>
變更控制器：
IESG iesg@ietf.org

7.4. CoAP 內容格式

媒體類型：application / cbor
編碼： -
Id：60
參考：[RFC7049]

7.5. + cbor 結構化語法尾綴註冊

名稱：簡明二進制物件表示（CBOR）

+尾綴：+ cbor

參考文獻：[RFC7049]

編碼考量：CBOR 是二進制格式。

互操作性考量：不適用

片段識別符的考量：

指定片段識別符的語法和語意

+ CBOR 應為“應用程式/ CBOR”指定。（在本文的出版物，對於“應用程式/ CBOR”未定義片段識別語法）。

對於片段識別符對於特定的語法和語意的“xxx / YYY + CBOR” 應如下處理：

對於+ cbor 中定義的情況，其中片段識別符根據+ cbor 規則進行解析，然後按照+ cbor 中指定的方式進行處理。

對於+ cbor 中定義的情況，其中片段識別符未按照+ cbor 規則解析，則按照“ xxx / yyy + cbor” 中的指定進行處理。

對於未在+ cbor 中定義的情況，請按照“ xxx / yyy + cbor” 中的指定進行處理。

安全考量：請參見本文的第 8 節

聯繫：應用領域工作組（apps-discuss@ietf.org）

作者/變更控制者：

應用領域工作組。

IESG 對此註冊進行了更改控制。

8. 安全考量

面向網路的應用程式可能會在其處理邏輯中顯示傳入資料的漏洞。眾所周知，複雜的解析器可能是此類漏洞的來源，例如遠程崩潰節點，甚至遠程執行任意編碼的能力。CBOR 試圖通過在可能的情況下賦予整個可編碼值範圍來減少解析器複雜性來縮小引入此類漏洞的機會。

資源耗盡攻擊可能會嘗試通過設置深層巢狀項來誘使解碼器分配非常大的資料項（字串，陣列，映射）或耗盡堆疊深度。解碼器需要具有適當的資源管理來緩解這些攻擊。（給出非常大的項目也可以嘗試利用整數溢位漏洞。）

當有可能對資料項進行多次解釋時，由閘管理者功能檢查 CBOR 資料項並稍後由不同應用程式使用的應用程式可能會出現漏洞。例如，攻擊者可能利用映射中的重複鍵和數字中的精確問題來使閘管理者的決定基於與第二個應用程序將使用的解釋不同的解釋。在安全上下文中使用的協定應該以這樣的方式定義，即這些多種解釋可靠地減少到單個解釋。為了促進這一點，在這種上下文中使用的編碼器和解碼器實施應該提供至少一種嚴格的操作模式（第 3.10 節）。

9. 致謝

CBOR 的靈感來自 MessagePack。MessagePack 由 Sadayuki Furuhashi（“frsyuki”）開發和推廣。對 MessagePack 的引用僅用於歸屬；CBOR 不是 MessagePack 的版本或替代品，因為它具有不同的設計目標和要求。

在 2012 年左右的同一時間，對於許多人來說，對原始 MessagePack 規範之外的功能的需求變得顯而易見。BinaryPack 是 Eric Zhang 為 binaryjs 項目開發的 MessagePack 的次要衍生產品。Tim Caswell 對他的 msgpack-js 和 msgpack-js-browser 項目進行了類似但不同的擴展。最近有很多人為擴展 MessagePack 來將文本字串表示形式與字節字串表示形式分開而做出了貢獻。

CBOR 中附加訊息的編碼受到了 Klaus Hartke 為 CoAP 設計的長度訊息編碼的啟發。

該文件還包含許多人提出的建議，特別是 Dan Frost，James Manger，Joe Hildebrand，Keith Moore，Matthew Lepinski，Nico Williams，Phillip Hallam-Baker，Ray Polk，Tim Bray，Tony Finch，Tony Hansen 和 Yaron Sheffer。

10. 參考文獻

10.1. 規範性參考文獻

[ECMA262] European Computer Manufacturers Association, "ECMAScript Language Specification 5.1 Edition", ECMA Standard ECMA-262, June 2011, <http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose InternetMail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March1997.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time onthe Internet: Timestamps", RFC 3339, July 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287,

December 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writingan IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Sincethe Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/ basedefs/V1_chap04.html#tag_04_15>.

10.2. 訊息參考

[ASN.1] International Telecommunication Union, "Information Technology -- ASN.1 encoding rules: Specification ofBasic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 1994.

[BSON] Various, "BSON - Binary JSON", 2013, <http://bsonspec.org/>.

[CNN-TERMS] Bormann, C., Ersue, M., and A. Keranen, "Terminologyfor Constrained Node Networks", Work in Progress, July2013.

[MessagePack] Furuhashi, S., "MessagePack", 2013, <http://msgpack.org/>.

[RFC0713] Haverty, J., "MSDTP-Message Services Data Transmission Protocol", RFC 713, April 1976.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July2006.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13,RFC 6838, January 2013.

[UBJSON] The Buzz Media, "Universal Binary JSON Specification", 2013, <http://ubjson.org/>.

[YAML] Ben-Kiki, O., Evans, C., and I. Net, "YAML Ain’tMarkup Language (YAML[TM]) Version 1.2", 3rd Edition, October 2009, <http://www.yaml.org/spec/1.2/spec.html>.

附錄A. 範例

下表提供了十六進制（右列）中的一些 CBOR 編碼值，以及這些值的偵錯表示法（左列）。請注意，字串“\u00fc”是 UTF-8 字串的一種偵錯表示形式，其中包含單個 Unicode 字元 U + 00FC，LATIN SMALL LETTER U WITH DIAERESIS（u umlaut）。類似地，“\u6c34”是偵錯表示法中的 UTF-8 字串，帶有單個字元 U + 6C34(CJK UNIFIED IDEOGRAPH-6C34，通常表示為“水”），“\ ud800 \ udd51” 是 UTF-8 字串帶有單個字元 U + 10151（GREEK ACROPHONIC ATTIC FIFTY STATERS)的偵錯符號。（請注意，所有這些單字元字串也可以用偵錯表示法中的原生 UTF-8 表示，而不是像現在這樣的僅 ASCII 規範。）在為大數提供的偵錯表示法中，顯示了它們的預期數值作為十進制數字（例如 18446744073709551616）而不是顯示標籤的位元組字串（例如 2（h'0100000000000000'））。

Diagnostic	Encoded
0	0x00
1	0x01
10	0x0a
23	0x17
24	0x1818
25	0x1819
100	0x1864
1000	0x1903e8
1000000	0x1a000f4240
1000000000000	0x1b000000e8d4a51000
18446744073709551615	0x1bffffffffffffffff
18446744073709551616	0xc249010000000000000000
-18446744073709551616	0x3bffffffffffffffff
-18446744073709551617	0xc349010000000000000000
-1	0x20
-10	0x29
-100	0x3863
-1000	0x3903e7
0.0	0xf90000
-0.0	0xf98000
1.0	0xf93c00
1.1	0xfb3ff199999999999a
1.5	0xf93e00
65504.0	0xf97bff
100000.0	0xfa47c35000
3.4028234663852886e+38	0xfa7f7fffff
1.0e+300	0xfb7e37e43c8800759c

5.960464477539063e-8	0xf90001
0.00006103515625	0xf90400
-4.0	0xf9c400
-4.1	0xfbc010666666666666
Infinity	0xf97c00
NaN	0xf97e00
-Infinity	0xf9fc00
Infinity	0xfa7f800000
NaN	0xfa7fc00000
-Infinity	0xfaff800000
Infinity	0xfb7ff0000000000000
NaN	0xfb7ff8000000000000
-Infinity	0xfbfff0000000000000
false	0xf4
true	0xf5
null	0xf6
undefined	0xf7
simple(16)	0xf0
simple(24)	0xf818
simple(255)	0xf8ff
0("2013-03-21T20:04:00Z")	0xc074323031332d30332d32315432303a30343a30305a
1(1363896240)	0xc11a514b67b0
1(1363896240.5)	0xc1fb41d452d9ec200000
23(h'01020304')	0xd74401020304
24(h'6449455446')	0xd818456449455446
32("http://www.example.com")	0xd82076687474703a2f2f7777772e6578616d706c652e636f6d
h''	0x40
h'01020304'	0x4401020304

""	0x60
"a"	0x6161
"IETF"	0x6449455446
"\"\\"	0x62225c
"\u00fc"	0x62c3bc
"\u6c34"	0x63e6b0b4
"\ud800\udd51"	0x64f0908591
[]	0x80
[1, 2, 3]	0x83010203
[1, [2, 3], [4, 5]]	0x8301820203820405
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]	0x98190102030405060708090a0b0c0d0e0f101112131415161718181819
{}	0xa0
{1: 2, 3: 4}	0xa201020304
{"a": 1, "b": [2, 3]}	0xa26161016162820203
["a", {"b": "c"}]	0x826161a161626163
{"a": "A", "b": "B", "c": "C", "d": "D", "e": "E"}	0xa5616161416162614261636143616461446165614 5
(_ h'0102', h'030405')	0x5f42010243030405ff
(_ "strea", "ming")	0x7f657374726561646d696e67ff
[_]	0x9fff
[_ 1, [2, 3], [_ 4, 5]]	0x9f018202039f0405ffff
[_ 1, [2, 3], [4, 5]]	0x9f01820203820405ff
[1, [2, 3], [_ 4, 5]]	0x83018202039f0405ff
[1, [_ 2, 3], [4, 5]]	0x83019f0203ff820405
[_ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]	0x9f0102030405060708090a0b0c0d0e0f101112131415161718181819ff
{_ "a": 1, "b": [_ 2, 3]}	0xbf61610161629f0203ffff

["a", {_ "b": "c"}]	0x826161bf61626163ff
{_ "Fun": true, "Amt": -2}	0xbf6346756ef563416d7421ff

表 4：編碼 CBOR 資料項的實例

附錄B. 跳越表

為簡潔起見，此跳越表不顯示為將來擴展保留的初始位元組。它僅顯示可用於可選功能的初始位元組的選擇。（所有無符號整數都按網路位元組順序排列。）

Byte	Structure/Semantics
0x00..0x17	Integer 0x00..0x17 (0..23)
0x18	Unsigned integer (one-byte uint8_t follows)
0x19	Unsigned integer (two-byte uint16_t follows)
0x1a	Unsigned integer (four-byte uint32_t follows)
0x1b	Unsigned integer (eight-byte uint64_t follows)
0x20..0x37	Negative integer -1-0x00..-1-0x17 (-1..-24)
0x38	Negative integer -1-n (one-byte uint8_t for n follows)
0x39	Negative integer -1-n (two-byte uint16_t for n follows)
0x3a	Negative integer -1-n (four-byte uint32_t for n follows)
0x3b	Negative integer -1-n (eight-byte uint64_t for n follows)
0x40..0x57	byte string (0x00..0x17 bytes follow)
0x58	byte string (one-byte uint8_t for n, and then n bytes follow)
0x59	byte string (two-byte uint16_t for n, and then n bytes follow)
0x5a	byte string (four-byte uint32_t for n, and then n bytes follow)
0x5b	byte string (eight-byte uint64_t for n, and then n bytes follow)
0x5f	byte string, byte strings follow, terminated by "break"
0x60..0x77	UTF-8 string (0x00..0x17 bytes follow)
0x78	UTF-8 string (one-byte uint8_t for n, and then n bytes follow)
0x79	UTF-8 string (two-byte uint16_t for n, and then n bytes follow)
0x7a	UTF-8 string (four-byte uint32_t for n, and then n bytes follow)
0x7b	UTF-8 string (eight-byte uint64_t for n, and then n bytes follow)

0x7f	UTF-8 string, UTF-8 strings follow, terminated by "break"
0x80..0x97	array (0x00..0x17 data items follow)
0x98	array (one-byte uint8_t for n, and then n data items follow)
0x99	array (two-byte uint16_t for n, and then n data items follow)
0x9a	array (four-byte uint32_t for n, and then n data items follow)
0x9b	array (eight-byte uint64_t for n, and then n data items follow)
0x9f	array, data items follow, terminated by "break"
0xa0..0xb7	map (0x00..0x17 pairs of data items follow)
0xb8	map (one-byte uint8_t for n, and then n pairs of data items follow)
0xb9	map (two-byte uint16_t for n, and then n pairs of data items follow)
0xba	map (four-byte uint32_t for n, and then n pairs of data items follow)
0xbb	map (eight-byte uint64_t for n, and then n pairs of data items follow)
0xbf	map, pairs of data items follow, terminated by "break"
0xc0	Text-based date/time (data item follows; see Section 2.4.1)
0xc1	Epoch-based date/time (data item follows; see Section 2.4.1)
0xc2	Positive bignum (data item "byte string" follows)
0xc3	Negative bignum (data item "byte string" follows)
0xc4	Decimal Fraction (data item "array" follows; see Section 2.4.3)
0xc5	Bigfloat (data item "array" follows; see Section 2.4.3)
0xc6..0xd4	(tagged item)

0xd5..0xd7	Expected Conversion (data item follows; see Section 2.4.4.2)
0xd8..0xdb	(more tagged items, 1/2/4/8 bytes and then a data item follow)
0xe0..0xf3	(simple value)
0xf4	False
0xf5	True
0xf6	Null
0xf7	Undefined
0xf8	(simple value, one byte follows)
0xf9	Half-Precision Float (two-byte IEEE 754)
0xfa	Single-Precision Float (four-byte IEEE 754)
0xfb	Double-Precision Float (eight-byte IEEE 754)
0xff	"break" stop code

表 5：起始位元組跳越表

附錄C. 偽碼

可以通過圖 1 中的偽碼檢查 CBOR 項的格式良好。僅當以下情況時，資料格式良好：

- 偽碼沒有“失敗”；
- 偽碼執行後，輸入中不留任何位元組（流應用程式中除外）

偽碼具有以下先決條件：

- take（n）從輸入資料中讀取 n 個位元組並將它們作為位元組字串返回。如果 n 個位元組不再可用，則 take（n）失敗。
- uint（）通過以網路位元組順序解釋位元組字串，將位元組字串轉換為無符號整數。
- 算術在 C 中起作用。

- 所有變數都是具有足夠範圍的無符號整數。

```
well_formed (breakable = false) {
  // process initial bytes
  ib = uint(take(1));
  mt = ib >> 5;
  val = ai = ib & 0x1f;
  switch (ai) {
    case 24: val = uint(take(1)); break;
    case 25: val = uint(take(2)); break;
    case 26: val = uint(take(4)); break;
    case 27: val = uint(take(8)); break;
    case 28: case 29: case 30: fail();
    case 31:
      return well_formed_indefinite(mt, breakable);
  }
  // process content
  switch (mt) {
    // case 0, 1, 7 do not have content; just use val
    case 2: case 3: take(val); break; // bytes/UTF-8
    case 4: for (i = 0; i < val; i++) well_formed(); break;
    case 5: for (i = 0; i < val*2; i++) well_formed(); break;
    case 6: well_formed(); break;     // 1 embedded data item
  }
  return mt;                    // finite data item
}

well_formed_indefinite(mt, breakable) {
  switch (mt) {
    case 2: case 3:
      while ((it = well_formed(true)) != -1)
        if (it != mt)           // need finite embedded
          fail();               //    of same type
      break;
    case 4: while (well_formed(true) != -1); break;
    case 5: while (well_formed(true) != -1) well_formed(); break;
    case 7:
      if (breakable)
        return -1;              // signal break out
      else fail();              // no enclosing indefinite
    default: fail();            // wrong mt
  }
  return 0;                     // no break out
}
```

圖 1：偽良構檢查

注意，完整 CBOR 解碼器的剩餘複雜性是關於以適當的形式呈現已解析到應用程式的資料。

主要類型 0 和 1 的設計方式是，它們可以從 C 中從一個有符號整數中編碼而無需實際對正/負進行 if-then-else（圖 2），這使用了（-1-n ），主要類型 1 的轉換與 C 無符號算術中的～n（按位補

碼）相同；然後，對於否定情況，～n 可以表示為（-1）^ n，而對於非否定的情況，0^n 保持 n 不變。通過將數字算術移位比該數字的位元短一位元(例如，對於 64 位元數字，減少至 63 位元)。

```
void encode_sint(int64_t n) {
  uint64t ui = n >> 63;     // extend sign to whole length
  mt = ui & 0x20;           // extract major type
  ui ^= n;                  // complement negatives
  if (ui < 24)
    *p++ = mt + ui;
  else if (ui < 256) {
    *p++ = mt + 24;
    *p++ = ui;
  } else
```

圖 2：偽碼用於編碼符號整數

附錄D. 半精確度

由於半精確度浮點數僅在 2008 年被添加到 IEEE 754，因此今天的編程平台通常僅對它們提供有限的支援。即使沒有這樣的支援，也很容易包括至少解碼支援。C 語言中用於半精確度浮點數的小解碼器的範例如圖 3 所示。用於 Python 的類似程式如圖 4 所示。此編碼假定 2 位元組值已經按網路位元組順序解碼為（無符號短整數）整數（如附錄 C 中的偽碼所示）。

```
#include <math.h>

double decode_half(unsigned char *halfp) {
  int half = (halfp[0] << 8) + halfp[1];
  int exp = (half >> 10) & 0x1f;
  int mant = half & 0x3ff;
  double val;
  if (exp == 0) val = ldexp(mant, -24);
  else if (exp != 31) val = ldexp(mant + 1024, exp - 25);
  else val = mant == 0 ? INFINITY : NAN;
  return half & 0x8000 ? -val : val;
}
```

圖 3：C 編碼為半精確度解碼器

```
import struct
from math import ldexp

def decode_single(single):
    return struct.unpack("!f", struct.pack("!I", single))[0]

def decode_half(half):
    valu = (half & 0x7fff) << 13 | (half & 0x8000) << 16
    if ((half & 0x7c00) != 0x7c00):
        return ldexp(decode_single(valu), 112)
    return decode_single(valu | 0x7f800000)
```

圖 4：Python 編碼用於一個半精確度解碼器

附錄E. 其他二進制格式與CBOR設計的比較

目標

關於 CBOR 的提議遵循二進制格式的歷史，該歷史與計算機本身的歷史一樣長。不同的格式有不同的目標。在大多數情況下，從未說明格式的目標，儘管有時可以通過首次使用格式的上下文來暗示它們。雖然歷史證明沒有二進制格式滿足所有協定和應用程式的需要，但有些格式可以普遍使用。

CBOR 與許多這些格式不同，因為它從一系列目標開始並試圖滿足這些目標。本節將幾十種格式中的一些與 CBOR 的目標進行比較，以幫助讀者決定是否要為特定協定或應用程式使用 CBOR 或不同的格式。

請注意，這裡的討論並不是對任何格式的批評：據我們所知，在 CBOR 之前沒有任何格式是為了在我們賦予它們的優先順序中涵蓋 CBOR 的目標。簡要回顧 1.1 節的目標是：

1. 從網際網路標準中對大多數常見資料格式進行明確編碼
2. 編碼器或解碼器的編碼緊湊性
3. 不需要架構描述
4. 合理緊湊的序列化
5. 適用於受約束和不受約束的應用程式
6. 良好的 JSON 轉換
7. 可擴展性

E.1. ASN.1 DER，BER 和 PER

[ASN.1]有許多序列化。在 IETF 中，DER 和 BER 是最常見的。對於許多項目，序列化輸出不是特別緊湊，並且在受約束的設備上解碼數字項所需的編碼可能很複雜。

很少（如果有的話）IETF 協定採用了壓縮編碼規則（PER）的幾種變體之一。這可能有很多原因，但通常會說的是 PER 甚至在解析資料串流的表面結構時也使用模式，需要大量的工具支援。正在使用不同版本的 ASN.1 模式語言，這也阻礙了採用。

E.2. MessagePack

[MessagePack]是一種簡潔，廣泛實施的計數二進制序列化格式，在許多屬性上與 CBOR 類似，儘管有些不那麼規律。雖然資料模型可用於表示 JSON 資料，但 MessagePack 也已用於許多遠程過程調用（RPC）應用程式以及資料的長期存儲。

MessagePack 一直基本上是穩定的，因為它是第一個在 2011 年左右公佈；它尚未過渡。要保持與現有存儲資料的完全向後兼容性，就必須限制 MessagePack 的發展，而只有少數的位元組碼可用於擴展。多年來，MessagePack 用戶社區重複提出了將二進制文件分離出來的請求並在編碼的文本字串最近編碼中的文本字串導致了一個擴展提議，它會使 MessagePack 的“原始”資料在其二進制和文本資料的用法之間不明確。 MessagePack 的擴展機制仍不清楚。

E.3. BSON

[BSON]是為在 MongoDB 資料庫中存儲類 JSON 物件（JSON 物件）而開發的資料格式。它的主要區別特徵是就地更新的能力，前面是一個緊湊的表示。BSON 使用計數表示法，但映射鍵除外，這些鍵以空字元組結尾。雖然 BSON 可以用於在線上表示類似 JSON 的物件，但其規範由資料庫應用程式的要求主導，並且變得有點巴洛克式。 BSON 擴展將如何實施的狀態仍不清楚。

E.4. UBJSON

[UBJSON]的設計目標是利用僅限於 JSON 使用的資料模型的二進制格式，使 JSON 更快，更小。因此，明確無意支援例如二進制資料；但是，有一個“高精確度數字”，表示為 JSON 語法中的字串。UBJSON 沒有針對編碼緊湊性進行優化，其類型位元組編碼針對人類識別進行了優化，而不是針對本機類型（如小整數）的緊湊表示。雖然 UBJSON 主要被計算在內，但它提供了一個保留的“未知長度”值來支援陣列和映射（JSON 物件）的流式傳輸。在這些容器中，UBJSON 還具有用於填充的“Noop”類型。

E.5. MSDTP: RFC 713

訊息服務資料傳輸（MSDTP）是緊湊訊息格式的一個非常早期的例子；它在 1976 年編寫的[RFC0713]中進行了描述。由於其歷史價值而將其包含在此處，並不是因為它曾經被廣泛使用。

E.6. 簡潔性電線

雖然CBOR的編碼器和解碼器的編碼緊湊性的設計目標優先於其在線路上的簡潔性，但許多人關注的是線纜尺寸。表 6 顯示了簡單巢狀陣列[1，[2,3]]的一些編碼範例；其中編碼支援某種形式的無限長編碼，[_ 1，[2,3]]（外部陣列上的不確定長度）也會顯示。

Format	[1, [2, 3]]	[_ 1, [2, 3]]
RFC 713	c2 05 81 c2 02 82 83	
ASN.1 BER	30 0b 02 01 01 30 06 02 01 02 02 01 03	30 80 02 01 01 30 06 02 01 02 02 01 03 00 00
MessagePack	92 01 92 02 03	
BSON	22 00 00 00 10 30 00 01 00 00 00 04 31 00 13 00 00 00 10 30 00 02 00 00 00 10 31 00 03 00 00 00 00 00	
UBJSON	61 02 42 01 61 02 42 02 42 03	61 ff 42 01 61 02 42 02 42 03 45
CBOR	82 01 82 02 03	9f 01 82 02 03 ff

表 6：為了簡明不同水平的範例

作者地址

Carsten Bormann Universitet Bremen TZI
Postfach 330440
D-28359 Bremen
德國

電話：+ 49-421-218-63921
電子郵箱：cabo@tzi.org

Paul Hoffman
VPN Consortium
電子郵件：paul.hoffman@vpnc.org