WebP の秘密

ゆずねり　著

2026-04-11 版　　NeIn 発行

はじめに

写真は JPEG、アイコンや透過画像は PNG、アニメーションは GIF。長い間、Web の画像はこの三つのフォーマットを使い分けるのが常識でした。

2010 年、Google がこの常識を覆します。写真も、イラストも、アニメーションも、すべてを一つでまかなう WebP の発表です。その中身を開けてみると、まったく異なる二つの圧縮技術が同居しています。

非可逆圧縮のベースは、動画コーデックの技術をそのまま静止画に持ち込んでいます。WebP は「1 フレームしかない動画」として、画像を圧縮するのです。一方、可逆圧縮には PNG の限界を超えるべく、画像専用のコーデックがゼロから設計されました。

本書では、RIFF コンテナの中に何が詰まっているのかを覗き、VP8 と VP8L という二つのコーデックがどうやって画像を縮めるのかを追いかけます。変換ツールの使いこなしにも踏み込み、各オプションと圧縮の仕組みが、どうつながっているのかを掘り下げます。さらに、Web サイトでの配信や既存フォーマットとの使い分けなど、実践的なヒントもお伝えします。

WebP の秘密を、一緒に解き明かしていきましょう。

本書サポートサイトのご案内

本書のサポートサイトは次の URL になります。

`https://neln.net/b/nt05/`

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

第 1 章

WebP とは何か

WebP は Google が開発した画像フォーマットです。JPEG、PNG、GIF が個別に担ってきた役割を、一つのフォーマットに統合します。その誕生の背景には、ビデオコーデックの技術転用という大胆な発想がありました。

1.1 画像フォーマットの歴史

Web で使われる画像フォーマットの世界は、長らく GIF、JPEG、PNG の三大フォーマットが支配してきました。

1987 年、Graphics Interchange Format (GIF) が登場します。LZW 圧縮を採用した可逆圧縮フォーマットですが、256 色に制限されていました。1989 年の GIF89a では、アニメーションと透過（特定のパレット色を透明に指定する方式）がサポートされました。

1992 年、Joint Photographic Experts Group (JPEG) が登場します。離散コサイン変換に基づく非可逆圧縮を採用し、フルカラーの写真に対して高い圧縮率を誇ります。Web の黎明期から写真の標準フォーマットとして広く普及します。

1996 年、Portable Network Graphics (PNG) が登場します。可逆（ロスレス）圧縮とアルファチャンネルをサポートしましたが、アニメーションには非対応でした。PNG は GIF の LZW 特許問題[*1]を契機に開発され、Deflate 圧縮による可逆圧縮を採用しました。

こうして「写真は JPEG、アニメーションは GIF、その他は PNG」という使い分けが、長い間 Web の常識となりました。しかし、JPEG には透過機能がなく、PNG は可逆圧縮のため写真には非効率でした。GIF は 256 色までしか表現できず、アニ

[*1] 1994 年末、Unisys が GIF の使用にライセンス料を要求したことで大きな論争に。米国特許は 2003 年、日本特許は 2004 年に失効。

メーション以外の用途では時代遅れとなっていました。

1.2 WebP の誕生

WebP の起源は、Google によるビデオコーデック企業 On2 Technologies の買収にあります。同社が開発した VP8 は、H.264 に対抗するビデオコーデックとして設計されました。Google は買収後に VP8 をオープンソース化し、WebM 動画フォーマットの基盤として公開しました。

そして、VP8 の技術を静止画像圧縮にも応用するというアイデアから、WebP が生まれます。

動画圧縮には二つの仕組みがあります。フレーム間の時間的相関を利用する「インターフレーム予測」と、1 枚のフレーム内の空間的相関を利用する「イントラフレーム予測」です。WebP の非可逆圧縮は、VP8 のイントラフレーム（キーフレームあるいは I フレーム）の技術を使用します。つまり、WebP は 1 フレームしかない動画のようなもので、ビデオコーデックの「1 枚の静止画を効率よく圧縮する」部分だけを取り出したフォーマットなのです。

2010 年 9 月 30 日、Google は WebP を正式に発表しました。当初は非可逆圧縮だけで、段階的に拡張されていきました。

▼表 1.1　WebP の主要な歴史

年月	出来事
2010 年 9 月	WebP 発表（非可逆圧縮のみ）
2011 年 11 月	可逆圧縮（VP8L）とアルファチャンネル対応
2013 年 3 月	アニメーション対応
2014 年 1 月	Chrome でアニメーション WebP 対応
2018 年 10 月	Edge が対応
2019 年 1 月	Firefox が対応
2020 年 9 月	Safari が対応
2024 年 11 月	RFC 9649 として IETF 文書化

1.3 WebP の特徴

WebP は、従来のフォーマットが個別に担っていた機能を、一つのフォーマットに統合した点が最大の特徴です。

1.3.1 非可逆圧縮と可逆圧縮の両対応

WebP は非可逆圧縮と可逆圧縮、二つの圧縮方式を同一フォーマット内でサポートします。

非可逆圧縮は VP8 コーデックの技術を基盤とし、同等の画質で JPEG より 25〜34% 小さくなるとされています。また、可逆圧縮は VP8L と呼ばれる専用の方式を採用し、PNG と比較して約 26% 小さくなるとされています[*2]。

1.3.2 アルファチャンネル

WebP は可逆圧縮、非可逆圧縮の両方でアルファチャンネル（透過）をサポートします。

非可逆圧縮の JPEG では透過が使えず、透過が必要な場合は可逆圧縮の PNG を使うしかありませんでした。WebP では非可逆圧縮でもアルファチャンネルを付加できるため、ファイルサイズと透過を両立できます。

1.3.3 アニメーション

WebP は GIF のようなアニメーションをサポートします。GIF が 256 色に制限されるのに対し、WebP はフルカラーでのアニメーションが可能です。

1.3.4 メタデータ

WebP は EXIF、XMP、ICC プロファイルといったメタデータの埋め込みに対応しています。撮影情報の保持や色再現が必要な用途にも利用可能です。

[*2] ウェブ用の画像形式 `https://developers.google.com/speed/webp`

1.3.5 従来フォーマットとの比較

▼表 1.2 WebP と従来フォーマットの機能比較

機能	JPEG	PNG	GIF	WebP
非可逆圧縮	○	×	×	○
可逆圧縮	×	○	○	○
アルファチャンネル	×	○	△[*3]	○
アニメーション	×	×	○	○
フルカラー（24bit 以上）	○	○	×	○
メタデータ（EXIF/XMP/ICC）	○	○	×	○

1.4 ブラウザー対応と普及

WebP の普及は、ブラウザーの対応と連動しています。

Chrome は Google が開発しているため、もっとも早く 2011 年にサポートを開始しました。Firefox は当初採用に消極的でしたが、WebP の圧縮効率が実証され Web での需要が高まったことから、2019 年に方針転換してサポートしました。Safari はもっとも遅く、2020 年にサポートしました。これにより、主要ブラウザーで WebP が使えるようになり、本格的に採用するきっかけとなりました。

2026 年現在、約 97% のブラウザーが WebP に対応[*4]しています。

[*3] GIF は特定の 1 色を透明に指定する方式。

[*4] WebP image format `https://caniuse.com/webp`

第 2 章

WebP ファイルフォーマット

WebP は RIFF コンテナをベースとした、柔軟なファイル構造を持ちます。12 バイトのヘッダーを起点に、VP8、VP8L、アニメーション、メタデータといったチャンクが連なるその構造を、バイト単位で読み解いていきます。

2.1 RIFF フォーマット

WebP は Resource Interchange File Format（RIFF）という、Microsoft と IBM が 1991 年に策定した汎用コンテナフォーマットをベースにしています。WAV 音声ファイルや AVI 動画ファイルなど、マルチメディアデータの格納に広く使われてきました。

RIFF はファイル全体を構造化されたブロックの連なりを、チャンク（chunk）として表現します。各チャンクは識別子、サイズ、データの 3 要素で構成されます。パーサーは未知のチャンクをスキップできるため、後方互換性を保ちながらフォーマットを拡張できます。当初は非可逆圧縮だけだった WebP が、可逆圧縮やアニメーションを追加できたのは、RIFF コンテナの柔軟な構造があったからです。

2.2 ファイルヘッダー

WebP ファイルの先頭は、常に次の 12 バイトで始まります。

▼表 2.1　RIFF ヘッダー

フィールド名	オフセット	サイズ	説明
FourCC	1〜4 バイト目	4 バイト	固定値「`RIFF`」（ASCII）
FileSize	5〜8 バイト目	4 バイト	残りのバイト数（LE、uint32）
FormType	9〜12 バイト目	4 バイト	固定値「`WEBP`」（ASCII）

FourCC は、このファイルが RIFF コンテナであることを示す固定の識別子です。

FileSize には、FourCC と FileSize の 8 バイトを除いた、残りのバイト数をリトルエンディアンの uint32 で記録します。

FormType は、RIFF コンテナに格納されているデータの種類を示します。WebP ファイルでは常に「`WEBP`」です。

13 バイト目以降に、画像データやメタデータを含むチャンクが続きます。

2.3 チャンクの基本構造

チャンクは次のような構造を持ちます。

▼表 2.2　チャンクの基本構造

フィールド名	オフセット	サイズ	説明
FourCC	1〜4 バイト目	4 バイト	チャンクの種類（4 文字の ASCII）
ChunkSize	5〜8 バイト目	4 バイト	Payload のバイト数（LE、uint32）
Payload	9 バイト目〜	ChunkSize	データ本体

FourCC は、チャンクの種類を表します。チャンクの種類は「`VP8 `」、「`VP8L`」、「`VP8X`」などです。

ChunkSize は、Payload のバイト数をリトルエンディアンの uint32 で記録します。サイズが奇数の場合は、次のチャンクとの境界を合わせるため、1 バイトの 0x00 が追加されます。この 1 バイトは ChunkSize に含まれません。

Payload は、チャンクが保持する実際のデータです。内容は FourCC で示されるチャンクの種類によって異なります。

2.4 ファイルレイアウト

WebP ファイルには、用途に応じて 3 種類のレイアウトがあります。

2.4.1 Simple Lossy

純粋な非可逆圧縮の画像に使用します。透過も、アニメーションも、メタデータも含められません。

▼リスト 2.1　Simple Lossy レイアウト

```
RIFF/WEBP
└──── 「VP8 」 チャンク（VP8ビットストリーム）
```

RIFF ヘッダーの直後に、VP8 チャンクが一つだけ配置されます。

2.4.2 Simple Lossless

可逆圧縮の画像に使用します。VP8L ビットストリームは元から ARGB フォーマットで動作するため、アルファチャンネルも扱えます。アニメーションやメタデータは含められません。

▼リスト 2.2　Simple Lossless レイアウト

```
RIFF/WEBP
└──── 「VP8L」 チャンク（VP8Lビットストリーム）
```

2.4.3 Extended

アルファチャンネル付きの非可逆圧縮の画像、アニメーション、ICC プロファイル、メタデータなど追加機能を必要とするときに使用します。

▼リスト 2.3　Extended レイアウト

```
RIFF/WEBP
├──── 「VP8X」 チャンク（拡張ヘッダー：機能フラグとキャンバスサイズ）
├──── 「ICCP」 チャンク（ICCカラープロファイル、任意）
├──── 「ANIM」 チャンク（アニメーションパラメーター、任意）
├──── 画像データ（静止画またはANMFフレームの並び）
│    ├──── 静止画の場合:
│    │    ├──── 「ALPH」 チャンク（アルファチャンネル、任意）
│    │    └──── 「VP8 」 or 「VP8L」 チャンク
│    └──── アニメーションの場合:
│         ├──── 「ANMF」 チャンク（フレーム1）
│         ├──── 「ANMF」 チャンク（フレーム2）
│         └──── ...
├──── 「EXIF」 チャンク（Exifメタデータ、任意）
└──── 「XMP 」 チャンク（XMPメタデータ、任意）
```

2.5 チャンク構造

WebP で使用される各チャンクの構造と役割を詳しく見ます。

2.5.1 VP8

FourCC は「`VP8 `」（末尾にスペースを含む 4 バイト）です。

VP8 チャンクのペイロードには、RFC 6386 で規定された VP8 キーフレームのビットストリームが格納されます。

▼表 2.3　VP8 チャンクのペイロード

フィールド名	オフセット	サイズ	説明
FrameTag	1〜3 バイト目	3 バイト	フレーム情報
StartCode	4〜6 バイト目	3 バイト	固定値 0x9D 0x01 0x2A
Width	7〜8 バイト目	2 バイト	画像幅とスケール（LE）
Height	9〜10 バイト目	2 バイト	画像高さとスケール（LE）
Data	11 バイト目〜	可変	VP8 圧縮データ

StartCode はキーフレームであることを示す、固定の 3 バイトです。WebP では、常に VP8 のキーフレームだけが使用されます。

Width と Height はそれぞれ 16 ビットのリトルエンディアン値です。下位 14 ビットが画像のピクセル数、上位 2 ビットがスケーリング係数を表します。このことから、VP8 の最大画像サイズは $16,383 \times 16,383$ ピクセルになります。

FrameTag は 3 バイト（24 ビット）をリトルエンディアンの整数として読み取り、次のビットフィールドに分解します。

▼表 2.4　FrameTag のビットフィールド

フィールド名	ビット位置	ビット幅	説明
frame_type	bit 0	1 ビット	フレーム種別
version	bit 1〜3	3 ビット	バージョン番号
show_frame	bit 4	1 ビット	表示フレームフラグ
first_part_size	bit 5〜23	19 ビット	最初のパーティションのサイズ

frame_type はキーフレームかどうかを示します。値は常に 0（キーフレーム）です。

show_frame は常に 1 です。

version はループフィルターの設定やデコーダーの再構成処理に影響する値です。0〜3 の範囲が定義されており、値が大きいほど再構成処理が簡略化され、デコード速度が向上する代わりに画質が低下します。

first_part_size は、VP8 ビットストリームの最初のパーティション（ヘッダー情報を含む部分）のバイト数です。

2.5.2 VP8L

FourCC は「`VP8L`」です。

VP8L チャンクのペイロードは、1 バイトのシグネチャーで始まり、続く 4 バイトに画像の基本情報がビットフィールドとして記録されます。

▼表 2.5　VP8L チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Signature	1 バイト目	1 バイト	固定値 0x2F
BitFields	2〜5 バイト目	4 バイト	画像情報
Data	6 バイト目〜	可変	VP8L 圧縮データ

BitFields は 4 バイト（32 ビット）を LSB ファーストで読み取り、次のビットフィールドに分解します。

▼表 2.6　BitFields のビットフィールド

フィールド名	ビット位置	ビット幅	説明
width_minus_one	bit 0〜13	14 ビット	画像幅 - 1
height_minus_one	bit 14〜27	14 ビット	画像高さ - 1
alpha_is_used	bit 28	1 ビット	アルファ使用ヒント
version_number	bit 29〜31	3 ビット	バージョン番号

width_minus_one と height_minus_one は、実際のピクセル数から 1 を引いた値です。これにより幅、高さが 0 になることを防ぎつつ、14 ビットをムダなく活用できます。このことから、VP8L の最大画像サイズは $16,384 \times 16,384$ ピクセルになります。

alpha_is_used はアルファチャンネルが使われているかどうかのヒントで、デコード結果には影響しません。

version_number は常に 0 です。

2.5.3 VP8X

FourCC は「VP8X」です。

VP8X チャンクは、Extended レイアウトで使用される拡張ヘッダーです。ファイルに含まれる機能のフラグとキャンバスサイズを保持します。ペイロードは 10 バイト固定です。

▼表 2.7　VP8X チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Flags	1 バイト目	1 バイト	機能フラグ
Reserved	2〜4 バイト目	3 バイト	予約（0 固定）
Canvas Width Minus One	5〜7 バイト目	3 バイト	キャンバス幅 - 1（LE）
Canvas Height Minus One	8〜10 バイト目	3 バイト	キャンバス高さ - 1（LE）

Canvas Width Minus One と Canvas Height Minus One は、実際のピクセル数から 1 を引いた値です。このことから、VP8X の最大キャンバスサイズは $16,777,216 \times 16,777,216$ ピクセルになります。ただし、キャンバス全体の画素数は $4,294,967,295$ 以下という制約があります。

Flags はビットフィールドで、次のフィールドを持ちます。ビット番号は MSB 0 方式で表記します。

▼表 2.8　Flags のビットフィールド

フィールド名	ビット位置	ビット幅	説明
Reserved (Rsv)	bit 0〜1	2 ビット	予約（0 固定）
ICC profile (I)	bit 2	1 ビット	ICCP チャンクが存在する
Alpha (L)	bit 3	1 ビット	透過情報を含む
Exif metadata (E)	bit 4	1 ビット	EXIF チャンクが存在する
XMP metadata (X)	bit 5	1 ビット	XMP チャンクが存在する
Animation (A)	bit 6	1 ビット	アニメーション画像である
Reserved (R)	bit 7	1 ビット	予約（0 固定）

ICC profile (I)、Exif metadata (E)、XMP metadata (X)、Animation (A) の各フラグは、対応するチャンクがファイル内に存在するかを示します。1 の場合は存在し、0 の場合は存在しないことを意味します。

Alpha (L) は、画像が透過情報を含むかを示します。非可逆圧縮の場合は ALPH チャンク、可逆圧縮の場合は VP8L ビットストリーム内にアルファチャンネルが格

納されます。

2.5.4 ICCP

FourCC は「`ICCP`」です。

ICCP チャンクには、ICC カラープロファイルのバイナリーデータがそのまま格納されます。WebP 固有のヘッダーや追加構造はありません。

▼表 2.9　ICCP チャンクのペイロード

フィールド名	オフセット	サイズ	説明
ICC Profile	1 バイト目〜	可変	ICC カラープロファイルデータ

ICCP チャンクはファイル内に最大一つで、画像データよりも前に配置しなければなりません。ICCP チャンクが存在しない場合、sRGB が仮定されます。

2.5.5 ALPH

FourCC は「`ALPH`」です。

ALPH チャンクは、非可逆圧縮の画像にアルファチャンネルを付加するために使用します。

▼表 2.10　ALPH チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Flags	1 バイト目	1 バイト	圧縮、フィルター設定
Data	2 バイト目〜	可変	アルファビットストリーム

Flags はビットフィールドで、次のフィールドを持ちます。ビット番号は MSB 0 方式で表記します。

▼表 2.11　Flags のビットフィールド

フィールド名	ビット位置	ビット幅	説明
Reserved (Rsv)	bit 0〜1	2 ビット	予約（0 固定）
Preprocessing (P)	bit 2〜3	2 ビット	前処理
Filtering method (F)	bit 4〜5	2 ビット	フィルター方式
Compression method (C)	bit 6〜7	2 ビット	圧縮方式

Preprocessing (P) は前処理の有無を示します。0 はなし、1 はレベル削減です。

Filtering method (F) はフィルター方式を指定します。0 はなし、1 は水平、2 は垂直、3 はグラディエントです。

Compression method (C) は圧縮方式を指定します。0 は非圧縮、1 は WebP Lossless 形式で圧縮です。非圧縮の場合、Data には画素数分の生データが格納され、各バイトが 8 ビットの透過値（0=透明、255=不透明）を表します。圧縮の場合、Data は WebP Lossless 形式で圧縮されたストリームです。

2.5.6 ANIM

FourCC は「`ANIM`」です。

ANIM チャンクは、アニメーション画像のグローバルパラメーターを保持します。ペイロードは 6 バイト固定です。

▼表 2.12　ANIM チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Background Color	1〜4 バイト目	4 バイト	背景色（BGRA 順）
Loop Count	5〜6 バイト目	2 バイト	ループ回数（LE）

Background Color はキャンバスの背景色を青、緑、赤、透明度の順で格納します。フレームの Disposal method が「背景色でクリア」の場合に使用されます。

Loop Count に設定した回数だけアニメーションを繰り返します。0 の場合は無限ループになります。

2.5.7 ANMF

FourCC は「`ANMF`」です。

ANMF チャンクは、アニメーションの各フレームを表します。16 バイトのヘッダーの後にフレームの画像データが続きます。

▼表 2.13　ANMF チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Frame X	1〜3 バイト目	3 バイト	X 座標 / 2（LE）
Frame Y	4〜6 バイト目	3 バイト	Y 座標 / 2（LE）
Frame Width Minus One	7〜9 バイト目	3 バイト	幅 - 1（LE）
Frame Height Minus One	10〜12 バイト目	3 バイト	高さ - 1（LE）
Frame Duration	13〜15 バイト目	3 バイト	表示時間（LE、ミリ秒）
Flags	16 バイト目	1 バイト	合成、破棄方法
Frame Data	17 バイト目〜	可変	フレーム画像データ

Frame X と Frame Y には実際の座標を 2 で割った値が格納されます。フレームの配置が常に偶数ピクセル境界にそろうことを意味します。

Frame Width Minus One と Frame Height Minus One は、実際のピクセル数から 1 を引いた値です。

Frame Duration は、フレームを表示してから次のフレームに遷移するまでの待ち時間をミリ秒単位で指定します。

Frame Data には、任意の ALPH チャンクと、VP8 または VP8L チャンクが格納されます。それぞれ独立したチャンクヘッダーを持つサブチャンクです。

Flags はビットフィールドで、次のフィールドを持ちます。ビット番号は MSB 0 方式で表記します。

▼表 2.14　Flags のビットフィールド

フィールド名	ビット位置	ビット幅	説明
Reserved	bit 0〜5	6 ビット	予約（0 固定）
Blending method (B)	bit 6	1 ビット	合成方法
Disposal method (D)	bit 7	1 ビット	破棄方法

Disposal method (D) はフレーム表示後の処理を指定します。0 はフレームを維持、1 は背景色でクリアです。

Blending method (B) はフレームの合成方法を指定します。0 はアルファブレンド（前フレームに合成）、1 は上書きです。

2.5.8　EXIF

FourCC は「`EXIF`」です。

EXIF チャンクには、Exif メタデータがそのまま格納されます。ペイロードは TIFF ヘッダー（「0x49 0x49 0x2A 0x00」または「0x4D 0x4D 0x00 0x2A」）から

始まります。

▼表 2.15　EXIF チャンクのペイロード

フィールド名	オフセット	サイズ	説明
Exif Data	1 バイト目〜	可変	Exif データ

EXIF チャンクはファイル内に最大一つで、画像データの後に配置されます。

2.5.9　XMP

FourCC は「`XMP `」（末尾にスペースを含む 4 バイト）です。

XMP チャンクには、XMP メタデータの XML テキストがそのまま格納されます。ラッパーやヘッダーは付きません。

▼表 2.16　XMP チャンクのペイロード

フィールド名	オフセット	サイズ	説明
XMP Data	1 バイト目〜	可変	XMP メタデータ

XMP チャンクはファイル内に最大一つで、画像データの後に配置されます。

第 3 章

非可逆圧縮（VP8）

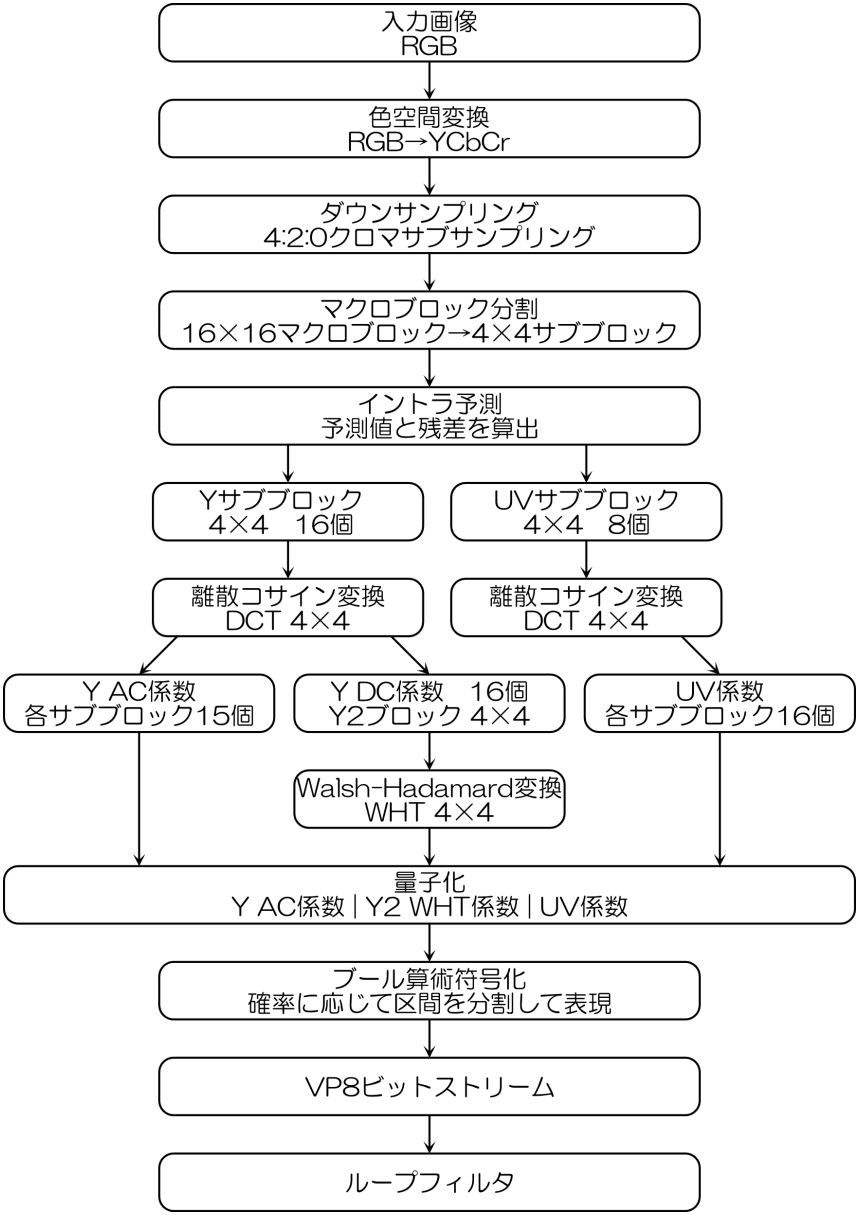
WebP の非可逆圧縮は、VP8 ビデオコーデックのキーフレームエンコードを静止画に応用したものです。画像をマクロブロックに分割し、予測、変換、量子化、エントロピー符号化というパイプラインで圧縮する、その各ステップの原理を追いかけます。

3.1 VP8 の概要

WebP の非可逆圧縮は VP8 と呼ばれるコーデックを使用します。VP8 はもともと On2 Technologies が開発したビデオコーデックでした。2010 年に Google が買収し、オープンソース化しました。

動画は連続するフレーム（静止画の系列）で構成されており、隣接フレーム間や同一フレーム内の類似性を利用してデータを削減します。動画圧縮にはイントラフレーム予測（フレーム内予測）と、インターフレーム予測（フレーム間予測）の 2 種類があります。WebP の非可逆圧縮は、イントラフレーム予測だけを使用する 1 フレームの動画を処理対象とします。

この 1 フレームの動画を圧縮するために、WebP は次の処理フローを実行します。



▲図 3.1　VP8 非可逆圧縮パイプライン

3.1.1 JPEG との処理フロー比較

JPEG のベースライン圧縮と比較すると、VP8 にはイントラ予測とループフィルターというビデオコーデック特有のステップがあります。また、エントロピー符号化も JPEG のハフマン符号化とは異なり、ブール算術符号化が使用されます。これらの違いにより、同じ画質なら JPEG より小さなファイルサイズを実現できます。

なかでも、イントラ予測の効果はとくに顕著です。JPEG が各ブロックを独立に量子化するのに対し、VP8 は隣接ピクセルから予測した残差だけを符号化します。予測が正確な領域では残差がほぼゼロになるため、JPEG では不可能なレベルのビット削減が実現できます。

▼表 3.1　JPEG と WebP の比較

処理ステップ	JPEG	WebP
色空間変換	YCbCr（BT.601）	YCbCr（整数近似式）
サブサンプリング	4:4:4、4:2:2、4:2:0	4:2:0 固定
ブロック分割	8 × 8 固定	16 × 16 マクロブロック ＋ 4 × 4 サブブロック
空間予測	DC 係数の DPCM のみ	イントラ予測
周波数変換	8 × 8 DCT-II	4 × 4 DCT-II の整数近似 ＋ WHT
DCT の入力	ピクセル値（レベルシフト後）	イントラ予測の残差
デコーダー間の再現性	実装依存（丸め誤差が異なる）	完全（整数演算のみ）
量子化	成分単位で固定の量子化テーブル	セグメント別の適応量子化
エントロピー符号化	ハフマン符号化	ブール算術符号化
デブロッキング	なし	ループフィルター

3.2　色空間変換

人間の視覚は輝度の変化には敏感ですが、色の変化には比較的鈍感です。色を輝度と色差に分離すれば、人間の視覚には鈍感な色差成分だけを粗く記録でき、知覚品質をほとんど損なわずにデータ量を削減できます。しかし、RGB 色空間では三つのチャンネルすべてに色と明るさの情報が混在しています。つまり、どのチャンネルも同じ精度で保持しなければなりません。

そこで、RGB 色空間から輝度と色差に分離する色空間変換を行います。

YCbCr 色空間であれば、色を輝度成分（Y）と二つの色差成分（Cb：青色差成分、Cr：赤色差成分）に分離できます。JPEG では、ITU-R BT.601 の浮動小数点変換

式をそのまま使用していました。

式 3.1: RGB 表色系から YCbCr 表色系へ変換（ITU-R BT.601）

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= -0.168736R - 0.331264G + 0.5B + 128 \\ C_r &= 0.5R - 0.418688G - 0.081312B + 128 \end{aligned}$$

VP8 ではエンコード、デコード処理を高速化するために、整数演算に最適化された近似式を使用します。近似式には固定小数点演算を用います。固定小数点演算とは、小数を含む値をあらかじめ一定の倍率でスケーリングして整数として扱う手法です。すべての計算を整数演算で行った後、最後にスケールを元に戻して本来の値に変換します。

YCbCr 色空間への変換は、256 倍したあとに計算し、最後に 256 で割ります。256 は 2^8 なので、割る代わりに 8 ビット右シフトすることで、浮動小数点演算を一切使わずに変換を完了できます。

VP8 はリミテッドレンジ[*1]を採用しているため、Y にはオフセットとして 16 を 256 倍した値を加算しています。Cb、Cr のオフセット 128 も 256 倍されます。右シフトによる切り捨てを四捨五入に補正するため、最後に $0.5 \times 256 = 128$ を加算します。

式 3.2: RGB 表色系から YCbCr 表色系へ変換（VP8 整数近似式）

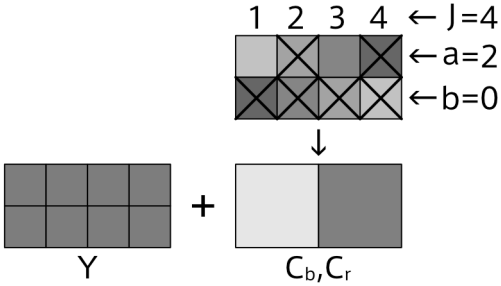
$$\begin{aligned} Y &= (\ 66R + 129G + 25B + 16 \times 256 + 128\ ) \gg 8 \\ C_b &= (\ -38R - 74G + 112B + 128 \times 256 + 128\ ) \gg 8 \\ C_r &= (\ 112R - 94G - 18B + 128 \times 256 + 128\ ) \gg 8 \end{aligned}$$

3.3 ダウンサンプリング

人間の視覚が色の違いに鈍感であるという特性を利用して、色差成分（Cb と Cr）を輝度成分（Y）よりも低い解像度で記録できます。これをダウンサンプリングと呼びます。WebP の非可逆圧縮でも、JPEG と同様にクロマサブサンプリング方式を採用しています。

JPEG では 4:4:4、4:2:2、4:2:0 のいずれかをエンコーダーが選択できますが、VP8 では 4:2:0 に固定されています。4:2:0 サブサンプリングでは、色差成分は輝度成分の半分の水平解像度と半分の垂直解像度で記録されます。

[*1] フルレンジでは Y が 0〜255 の全範囲を使うのに対し、リミテッドレンジでは Y が 16〜235、Cb、Cr が 16〜240 の範囲に制限されます。放送やビデオの規格で広く使われています。



▲図 3.2　4:2:0 サブサンプリング

色差情報の解像度低下による画質劣化は、通常の写真ではほとんど知覚できません。ただし、テキストの縁や色の境界が鮮明な画像では、色にじみが目立つことがあります。

3.4　マクロブロックへの分割

JPEG は画像を 8 × 8 ピクセルのブロックに分割して処理していました。

VP8 はより大きな 16 × 16 ピクセルのマクロブロックを基本単位とします。大きな単位を採用することで、イントラ予測の効果が広い範囲で発揮されます。画像サイズが 16 の倍数でない場合は、右端と下端がパディングされます。

各マクロブロックはさらに 4 × 4 ピクセルのサブブロックに分割されます。一つのマクロブロックに含まれるサブブロックの構成は、次のとおりです。

- 輝度成分（Y）
  - 4 × 4 ピクセルのサブブロックが 16 個
- 色差成分（Cb と Cr）
  - 4 × 4 ピクセルのサブブロックが 4 個

色差成分は 4:2:0 サブサンプリングにより水平、垂直ともに解像度が半分になっているため、そのマクロブロックのサイズは 8 × 8 ピクセルになります。

この階層構造は、JPEG の 8 × 8 固定ブロックと比較して圧縮効率に大きく寄与します。

JPEG は画像の内容にかかわらず常に 8 × 8 ブロック単位で処理するため、広い平坦領域でも 64 個の DCT 係数を計算、符号化する必要があります。また、ブロック境界をまたぐピクセル間の相関は DC 係数の DPCM 以外では利用できず、量子化誤差によってブロック境界で不連続が生じます。

VP8 は、平坦な領域ではマクロブロック全体に同一の予測を行います。テクスチャーが複雑な領域では、サブブロック単位で予測を行います。局所的なエッジやテクスチャーの傾きに沿った予測を選ぶことで、効果的に圧縮できます。

3.5 イントラ予測

自然画像ではあるピクセルの色は、ほとんどの場合、隣接するピクセルと似ています。青空の中の 1 ピクセルは、その隣のピクセルもほぼ同じ青色になります。このような空間的な相関を利用するのがイントラ予測です。

イントラ予測は、JPEG にはない技術です。

VP8 では隣接ピクセルから現在のブロックのピクセル値を予測し、実際の値との差分（残差）だけを符号化します。予測が正確であるほど残差は小さくなり、少ないビットで表現できます。このイントラ予測によって、WebP は JPEG よりも効率的に圧縮できます。

JPEG はピクセル値をそのまま DCT に入力します。ピクセル値は広い範囲に分布するため、多くの周波数成分にエネルギーが散らばります。

一方、VP8 が入力する予測残差はゼロ付近に集中しています。DCT 後のエネルギーが DC 成分と低周波成分に強く集中し、それ以外の係数は小さくなります。量子化後にゼロになる係数が増えるため、JPEG よりも少ないビット数で同等の画質が実現できます。

VP8 には、予測に二つのレベルがあります。マクロブロック全体を対象とする予測と、サブブロックごとに行う予測です。

3.5.1 輝度予測モード

輝度成分の予測では、マクロブロックごとに 5 種類の予測モードから最適なものを選択します。

▼表 3.2　マクロブロック輝度予測モード

モード名	説明
DC_PRED	上の行と左の列の平均をブロック全体の予測値とする
V_PRED	上の行（16 ピクセル）を垂直方向にそのまま引き延ばす
H_PRED	左の列（16 ピクセル）を水平方向にそのまま引き延ばす
TM_PRED	TrueMotion 予測。上の行と左の列から 2 次差分を伝播
B_PRED	サブブロックごとに個別の予測モードを選択

DC_PRED は上の行と左の列のピクセル値から平均値を計算し、ブロック全体を

均一に予測します。平坦な領域で効果的です。

V_PRED は上の行の各ピクセル値を下方向にそのままコピーします。水平方向に一様で、垂直方向の変化が少ない領域に効果的です。

H_PRED は V_PRED の水平版です。左の列の各ピクセル値を右方向にそのままコピーします。

TM_PRED（TrueMotion）は VP8 独自の予測モードです。連続的に変化するグラデーションや斜め方向のパターンに対してとくに効果的です。この予測は、左端の列を上から下へたどることで得られる変化量を、上端の各ピクセルの値に加算するものです。

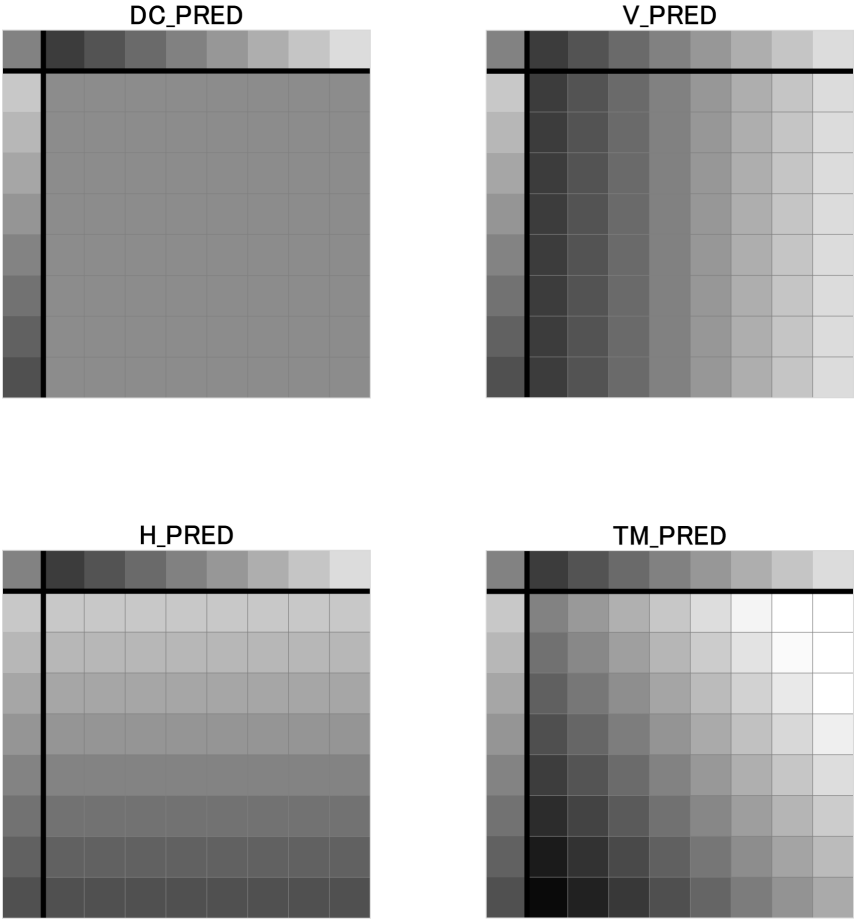
L_i を左の列の i 行目のピクセル値、A_j を上の行の j 列目のピクセル値、P を左上隅のピクセル値としたとき、次のような式になります。

式 3.3: TrueMotion 予測

$$X_{ij} = L_i + A_j - P$$

$L_i - P$ は左端が左上隅からどれだけ変化したかを表し、この変化量を上端の各ピクセルに適用することで、2 次元的なグラデーションを再現します。

各モードの予測は次のようになります。各図の上端 1 行と左端 1 列は、すでに復号済みの隣接ピクセル（レファレンス）を表し、黒い太線で囲まれた内側が予測されるブロックです。



▲図 3.3　マクロブロック輝度予測モードの可視化

エンコーダーは、各マクロブロックに対して複数の予測モード候補を試行し、最適なものを選択します。

サブブロック輝度予測モード

B_PRED が選択された場合、輝度成分の 4 × 4 サブブロックで個別に予測モードを選択します。サブブロック単位の予測は、テクスチャーが複雑な領域やエッジが多い領域でとくに効果的です。サブブロックには、10 種類の予測モードが使用でき

ます。

▼表 3.3　4 × 4 輝度予測モード

モード名	角度	説明
B_DC_PRED	-	上と左の参照ピクセルの平均値
B_TM_PRED	-	TrueMotion（2 次元勾配推定）
B_VE_PRED	0°	上辺を下方向に延長（平滑化あり）
B_LD_PRED	45°	右上から左下方向
B_VL_PRED	63°	B_LD_PRED よりやや垂直寄り
B_HE_PRED	90°	左辺を右方向に延長（平滑化あり）
B_VR_PRED	117°	B_RD_PRED よりやや垂直寄り
B_RD_PRED	135°	左上から右下方向
B_HD_PRED	153°	B_RD_PRED よりやや水平寄り
B_HU_PRED	207°	左下から右上方向

B_DC_PRED と B_TM_PRED は、マクロブロックの DC_PRED、TM_PRED と同じアルゴリズムを、サブブロックに適用したものです。

残りの八つのモードは方向予測モードで、隣接する参照ピクセルを指定の方向に沿って補間、伝播させることで予測します。各モードが異なる方向のエッジやテクスチャーパターンに対応します。たとえば、B_LD_PRED は右上から左下への斜めパターンに、B_RD_PRED は左上から右下への斜めパターンなどにそれぞれ適しています。

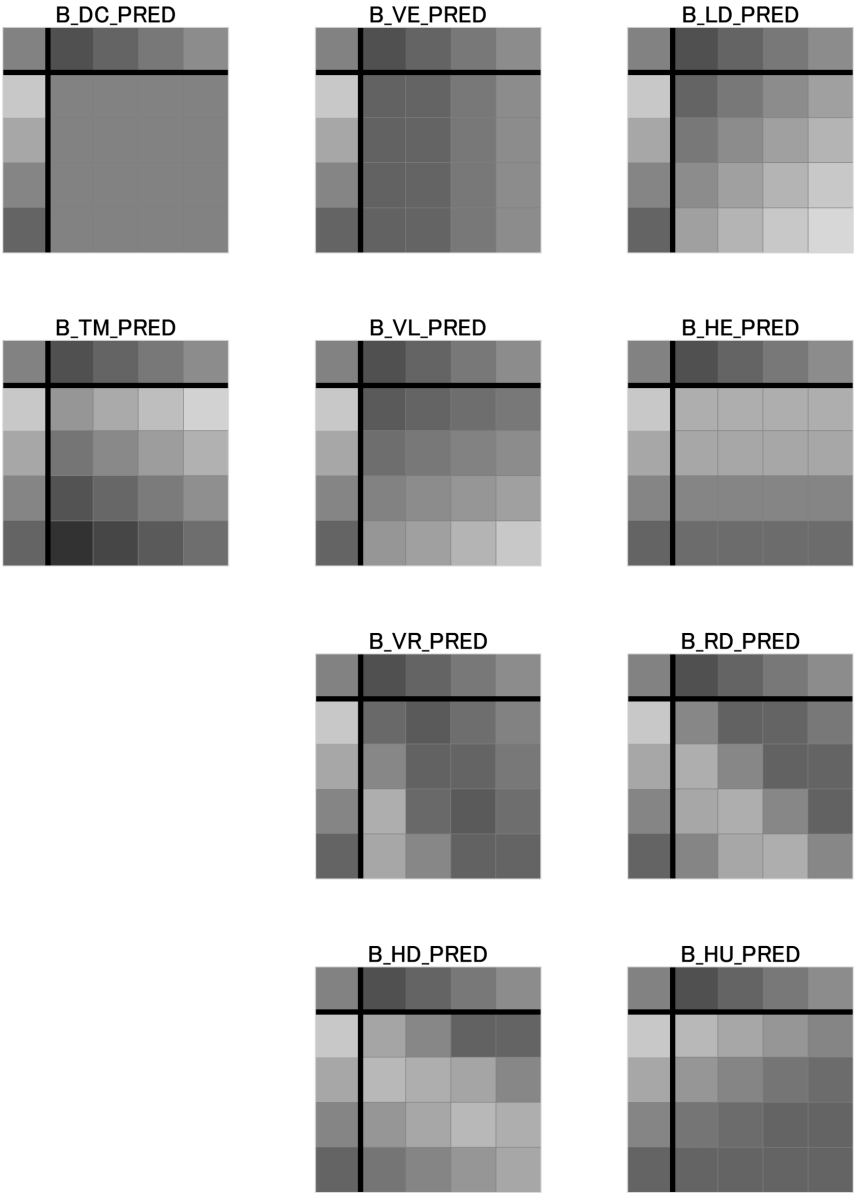
B_VE_PRED と B_HE_PRED は、マクロブロックレベルの V_PRED や H_PRED とは異なり、単純なコピーではなく 3 ピクセルの加重平均による平滑化フィルターを使用します。たとえば、B_VE_PRED では上辺の参照ピクセルを次の式で平滑化してから下方向にコピーします。

式 3.4: B_VE_PRED の平滑化

$$\mathrm{pred}[i] = (A_{i-1} + 2A_i + A_{i+1} + 2) \gg 2$$

B_HE_PRED でも同様に、左辺の参照ピクセルに対して垂直方向の加重平均を適用します。この平滑化により、サブブロックの小さな予測領域でのノイズ感度を抑え、より安定した予測を実現しています。

各モードの予測は次のようになります。各図の上端 1 行と左端 1 列は、すでに復号済みの隣接ピクセル（レファレンス）を表し、黒い太線で囲まれた内側が予測されるブロックです。



▲図 3.4　サブブロック輝度予測モードの可視化

3.14 エンコードパラメーターと内部処理の関係

libwebp の cwebp コマンドには、圧縮の挙動を制御する多数のパラメーターがあります。ここでは、各パラメーターがどのステップに影響するかを整理します。

3.14.1 quality パラメーター

quality（-q、0〜100）はもっとも基本的なパラメーターで、処理フローの複数ステップに影響します。

量子化への影響

quality の値は、量子化インデックス yac_qi（0〜127）に変換されます。

quality が高いほど yac_qi は小さくなり、量子化ステップが小さくなります。より多くの DCT 係数が保持され、画質が向上します。

逆に quality が低いほど yac_qi は大きくなり、量子化が粗くなるため、ファイルサイズは小さくなります。

yac_qi はベースラインインデックスとして、六つの量子化カテゴリーすべてに波及します。各カテゴリーの量子化ステップは、デルタ値が同じでも、ベースラインが変われば連動して変化します。

ループフィルターへの影響

quality はループフィルターの強度にも影響します。

低品質設定ではブロックノイズが目立ちやすくなります。ブロックノイズを打ち消すために loop_filter_level が強くなり、その効果も高まります。

レート歪み最適化への影響

レート歪みコスト関数のラグランジュ乗数は、量子化パラメーターから導出されます。quality が低い（量子化が粗い）ほどラグランジュ乗数は大きくなり、ビット数の削減がより重視されます。

3.14.2 method パラメーター

method（-m、0〜6）は、レート歪み最適化の探索精度と処理速度のトレードオフを制御します。quality が最終的な画質レベルを決めるのに対し、method はその画質レベルでの圧縮効率を高めます。method が大きくなるほど、圧縮効率は向上しますがエンコード時間も増加します。

3.14.9 cwebp 主要オプション一覧

▼表 3.16 cwebp の主要オプション一覧

オプション	デフォルト	説明
`-q`	75	圧縮品質（0〜100）
`-m`	4	圧縮方式（0〜6）
`-preset`	default	素材種別のプリセット default / photo / picture / drawing / icon / text
`-sharp_yuv`	off	高精度な YUV 変換を使用
`-pre`	0	前処理のビットフラグ（0〜3） 1=境界補正 / 2=バンディング軽減
`-sns`	50	空間ノイズシェーピング（0〜100） 0 で無効
`-segments`	4	セグメント数の上限（1〜4）
`-f`	60	ループフィルター強度（0〜100） 0 で無効
`-sharpness`	0	フィルターの抑制度（0〜7）
`-strong`	on	Normal フィルターを使用
`-nostrong`	off	Simple フィルターを使用
`-af`	off	フィルター強度の自動調整
`-size`	-	目標ファイルサイズ（バイト）
`-pass`	1	最大パス数（1〜10） `-size` 指定時は 6
`-alpha_q`	100	アルファチャンネル圧縮品質（0〜100） 100 でロスレス
`-alpha_filter`	fast	アルファフィルター選択戦略 none / fast / best
`-alpha_method`	1	アルファチャンネル圧縮方式 0=非圧縮 / 1=VP8L 圧縮
`-exact`	off	透明領域の RGB 値を保持
`-noalpha`	off	アルファチャンネルを破棄
`-mt`	off	マルチスレッドエンコードを使用
`-low_memory`	off	メモリ使用量を削減
`-metadata`	none	保持するメタデータ exif / icc / xmp / all / none

第 4 章

可逆圧縮（VP8L）

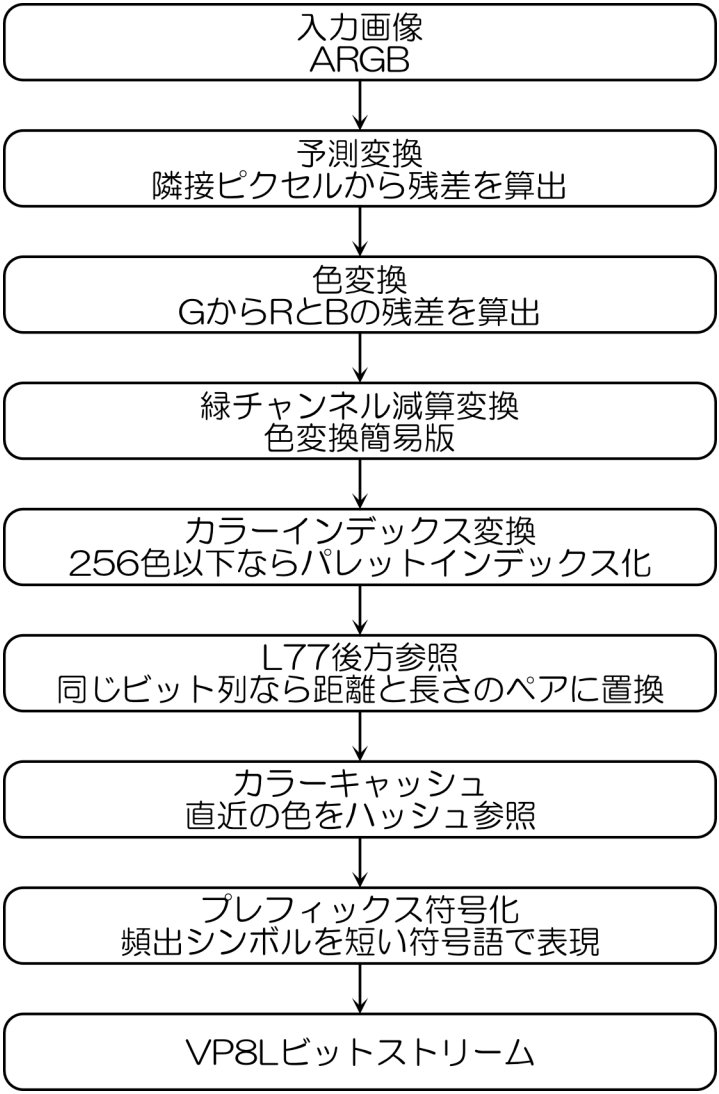
WebP の可逆圧縮は、VP8L と呼ばれる独自のコーデックを使用します。VP8 がビデオコーデック由来であるのに対し、VP8L は画像の特性に特化してゼロから設計されました。色空間変換や予測フィルターで冗長性を削り、LZ77 とハフマン符号化で一気に圧縮する、その仕組みを解き明かします。

4.1 VP8L の概要

VP8L は静止画の可逆圧縮のためにゼロから設計されたコーデックです。非可逆圧縮の VP8 がビデオコーデック由来であるのに対し、VP8L は画像の特性に特化した設計になっています。

可逆圧縮では、PNG が広く使われています。PNG は Deflate を用いた LZ77 とハフマン符号化で圧縮します。しかし、Deflate はバイト列の汎用圧縮アルゴリズムであり、画像の 2 次元的な構造やチャンネル間の相関を十分に活用できません。

VP8L はこれらの課題に対して、画像の特性を活かした独自の改良を導入し、より高い圧縮率を実現しています。VP8 と比べると、VP8L はシンプルなフローになります。



▲図 4.1　VP8L の処理フロー

4.1.1 PNG との処理フロー比較

PNG と比較すると、VP8L には 14 種類の予測モード、チャンネル間の相関を除去する色変換、2 次元空間を考慮した距離マッピングがあります。また、エントロピー符号化も PNG の Deflate とは異なり、チャンネル別の独立した符号表と 2 次元領域適応が使用されます。これらの違いにより、PNG より小さなファイルサイズを実現できます。

なかでも、メタプレフィックスコードの導入が圧縮率に大きく寄与しています。PNG の Deflate は、バイトストリーム上の 1 次元ブロック単位でハフマンテーブルを割り当てます。一方、VP8L は、2 次元矩形領域単位でチャンネルごとに独立した符号表を割り当てます。

▼表 4.1　PNG と VP8L の比較

処理ステップ	PNG	VP8L
空間予測	5 種類のフィルター（行単位）	14 種類の予測モード（ブロック単位）
色の非相関化	なし	色変換 ＋ 緑チャンネル減算変換
パレット	インデックスカラーモード	カラーインデックス変換 ＋ ピクセルバンドリング
LZ77 距離	1 次元バイトオフセット	2 次元距離マッピング
エントロピー符号化	1 次元ハフマンテーブル	チャンネル別の独立符号表 ＋ メタプレフィックスコード
色の再利用	なし	カラーキャッシュ

4.2　画像変換

VP8L では、エントロピー符号化の前に最大 4 種類の可逆変換を画像データに適用できます。各変換はデータの統計的な冗長性を減らし、エントロピー符号化の効率を高めます。

変換の種類は次のとおりです。

1. 予測変換
2. 色変換
3. 緑チャンネル減算変換
4. カラーインデックス変換

変換はエンコード時に順に適用され、デコード時には逆順に逆変換が適用されます。各変換は最大 1 回ずつ使用でき、エンコーダーが画像の特性に応じて最適な組み合わせを選択します。

4.3 予測変換

予測変換は、空間的な相関を利用してデータの冗長性を減らす変換です。考え方は、VP8 の非可逆圧縮におけるイントラ予測と同様です。

PNG のフィルタリングも予測変換の一種ですが、VP8L は画像の特性に応じて最適な予測を選択できます。

画像はまず、4×4〜512×512 ピクセル四方のブロックに分割されます。各ブロック内では、ピクセルの値を隣接ピクセルから予測し、実際の値と予測値の差分（残差）を記録します。自然画像では隣接ピクセル同士が類似するため、残差は小さくなる傾向があり、エントロピー符号化の効率が向上します。

現在のピクセルを P とし、左隣を L、上を T、右上を TR、左上を TL とします。

▼リスト 4.1　ピクセルの位置

```
TL   T   TR
 L   P
```

VP8L の予測モードには次の 14 種類があります。

▼表 4.2　予測モード

モード	予測値	説明
0	0xff000000	不透明の黒
1	L	左のピクセル
2	T	上のピクセル
3	TR	右上のピクセル
4	TL	左上のピクセル
5	$\mathrm{avg}(\mathrm{avg}(L, TR), T)$	L、TR、T の 2 段階平均
6	$\mathrm{avg}(L, TL)$	L と TL の平均
7	$\mathrm{avg}(L, T)$	L と T の平均
8	$\mathrm{avg}(TL, T)$	TL と T の平均
9	$\mathrm{avg}(T, TR)$	T と TR の平均
10	$\mathrm{avg}(\mathrm{avg}(L, TL), \mathrm{avg}(T, TR))$	4 ピクセルの平均
11	$\mathrm{Select}(L, T, TL)$	勾配に基づく選択
12	$\mathrm{ClampAddSubtractFull}(L, T, TL)$	$L + T - TL$ の線形予測
13	$\mathrm{ClampAddSubtractHalf}(\mathrm{avg}(L, T), TL)$	L と T の平均と TL の中間

各チャンネル（透明度（A）、赤（R）、緑（G）、青（B））に対して、独立に予測を行います。

画像の境界にはピクセルが存在しないため、割り当てられたモードにかかわらず、予測値は次のように固定されます。

▼表 4.3　境界ピクセルの予測値

位置	予測値
左上隅	0xff000000（不透明の黒）
左上隅を除く最上行	左のピクセル
左上隅を除く最左列	上のピクセル
最右列の右上	同じ行の最左ピクセル

モード 1 は PNG の Sub フィルター、モード 2 は Up フィルター、モード 7 は Average フィルターにそれぞれ相当します。モード 11 は Paeth フィルターに近い位置づけですが、候補ピクセルの範囲や判定方法が異なります。

モード 1〜4 は、隣接する一つのピクセルをそのまま予測値として使用する単純なモードです。水平方向に色が一定な領域ではモード 1（左）、垂直方向ではモード 2（上）が効果的です。

モード 5〜10 は、複数の隣接ピクセルの平均を予測値とするモードです。平均を取ることで、単一ピクセルによる予測よりもノイズの影響を受けにくくなります。モー

ド 6（L と TL の平均）やモード 7（L と T の平均）は、水平方向、垂直方向の変化が穏やかな領域で有効です。モード 10 は四つの隣接ピクセルの平均を使うため、もっとも平滑化の効果が高く、ノイズの多い平坦な領域に適しています。モード 5 は L、TR、T の 2 段階平均で、右上方向への変化を考慮した非対称な予測です。

エンコーダーは複数のサイズと予測モードを試して、もっとも効率的な組み合わせを選択します。どのモードが有効かは画像の領域特性によって異なります。

4.3.1 モード 11（Select 予測）

勾配推定値 $p = L + T - TL$ を各チャンネルで計算し、L と T のうち p に近い方を予測値として選択します。エッジ付近で効果的です。勾配の向きに応じて水平、垂直の予測を切り替えることで、エッジに沿った予測をします。

判定には、4 チャンネル間の差分の絶対値を合計した、マンハッタン距離を使用します。ユークリッド距離のように、二乗や平方根を使わないため整数演算だけで高速に計算できます。

式 4.1: Select 予測

$$
\begin{aligned}
p_c &= L_c + T_c - TL_c \quad (c \in \{A, R, G, B\}) \\
d_L &= \sum_c |p_c - L_c| \\
d_T &= \sum_c |p_c - T_c| \\
\text{pred} &= \begin{cases} L & (d_L < d_T) \\ T & (\text{otherwise}) \end{cases}
\end{aligned}
$$

PNG の Paeth フィルターは、各チャンネルで独立に L、T、TL のうち、p にもっとも近いものを選択します。VP8L の Select は、4 チャンネルの距離を合算して L か T のどちらかを選択する点が異なります。

4.3.2 モード 12（ClampAddSubtractFull 予測）

勾配推定値 $L + T - TL$ を計算し、`clamp` 関数で結果が $[0, 255]$ の範囲に収まるよう切り詰めます。なめらかなグラデーション領域に有効です。

式 4.2: ClampAddSubtractFull 予測

$$\text{pred}_c = \text{clamp}(L_c + T_c - TL_c)$$

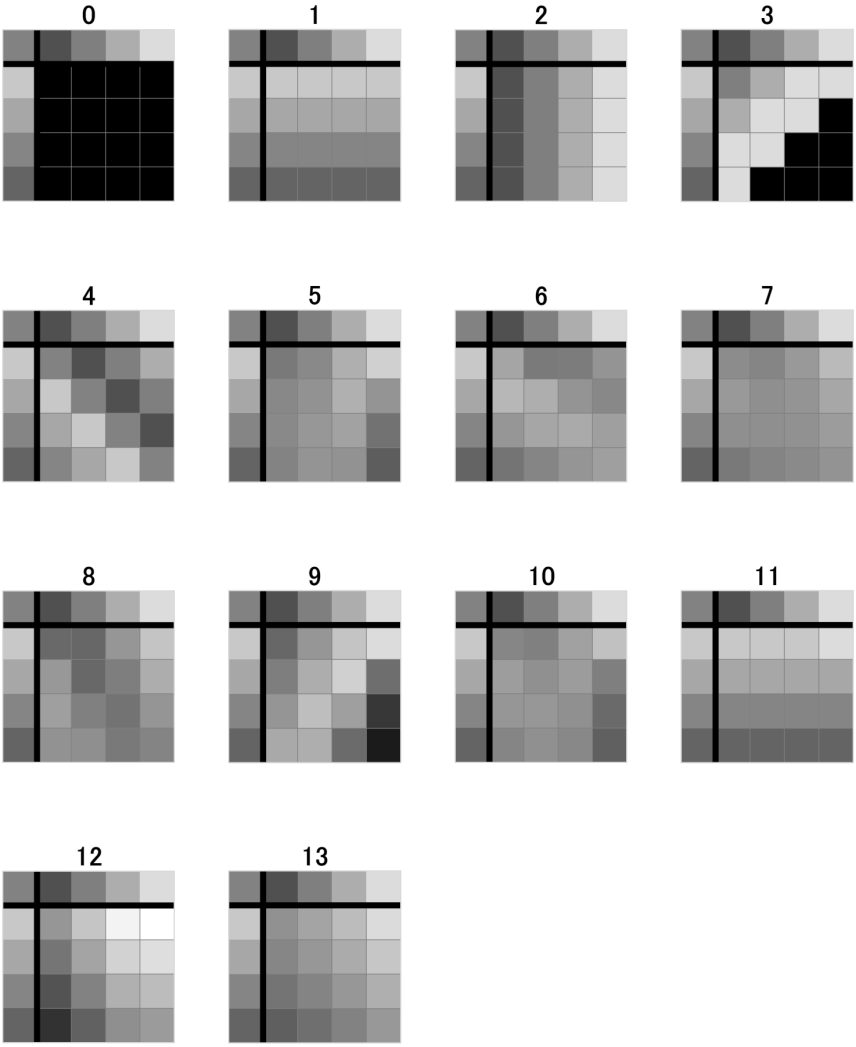
4.3.3 モード 13（ClampAddSubtractHalf 予測）

L と T の平均値に、TL との差分の半分を加えた、補正値を予測とします。モード 12 よりも保守的な予測で、急激な変化が少ない領域で有効です。

式 4.3: ClampAddSubtractHalf 予測

$$
\begin{aligned}
a_c &= \mathrm{avg}(L_c, T_c) \\
\mathrm{pred}_c &= \mathrm{clamp}\Big(a_c + \mathrm{trunc}\Big(\frac{a_c - TL_c}{2}\Big)\Big)
\end{aligned}
$$

各モードの予測は次のようになります。各図の上端 1 行と左端 1 列は、復号済みの隣接ピクセル（レファレンス）を表し、黒い太線で囲まれた内側が予測されるブロックです。



▲図 4.2　予測モードごとの予測値の可視化

同じ隣接ピクセルに対してもモードごとに予測値は大きく異なります。画像の特徴に合ったモードを選択することで、残差を最小化できます。

残差の計算

残差の計算と復元にはモジュラ算術を使用します。

8 ビットのモジュラ算術では、減算と加算が互いの逆操作になります。オーバーフローが起きても下位 8 ビットだけを取ることで、常に元の値を正確に復元できます。

たとえば $30 - 200 = -170$ の下位 8 ビットを取ると、$(-170)\ \&\ \mathrm{0xff} = 86$ になります。復元時に $86 + 200 = 286$ となりますが、下位 8 ビットは $286\ \&\ \mathrm{0xff} = 30$ で、元の値が正しく復元されます。

4.4 色変換

色変換は、RGB 各チャンネル間の相関を除去することで、エントロピーを低下させる変換です。自然画像では R、G、B の値は似ていることが多いため、チャンネル間の相関を利用して冗長性を減らせます。

G はそのまま保持し、R を G から予測した残差に、B を G と R から予測した残差に置き換えます。人間の視覚は緑にもっとも敏感であるため、G を基準にします。R で得られた残差情報を利用することで、B はより効率的に予測できます。

画像は予測変換と同様にブロックに分割され、各ブロックに三つの変換パラメーターが割り当てられます。

- green_to_red
  - G から R を予測する係数
- green_to_blue
  - G から B を予測する係数
- red_to_blue
  - R から B を予測する係数

各パラメーターは符号なし 8 ビット整数として格納され、計算時に符号付き 8 ビット整数に変換されます。符号付きなのは、チャンネル間の相関が正にも負にもなりうるためです。

t を色変換パラメーター、c をチャンネル値とすると、固定小数点乗算による色変換の予測計算は次のようになります。

式 4.4: 色変換の予測計算

$$\mathrm{ColorTransformDelta}(t, c) = (t \times c) \gg 5$$

$\gg 5$ は、32 で除算することに相当します。そのため、変換パラメーターは実質的に 1/32 刻みの係数として機能します。

エンコード時の変換は次のとおりです。

式 4.5: 色変換（エンコード）

$$\begin{aligned} R' &= (R - \text{ColorTransformDelta}(\text{green\_to\_red},\ G))\ \&\ \text{0xff} \\ B' &= (B - \text{ColorTransformDelta}(\text{green\_to\_blue},\ G) \\ &\qquad - \text{ColorTransformDelta}(\text{red\_to\_blue},\ R))\ \&\ \text{0xff} \end{aligned}$$

デコード時は、減算を加算に置き換えます。

4.5 緑チャンネル減算変換

緑チャンネル減算変換は、色変換の簡略版です。

式 4.6: 緑チャンネル減算変換

$$\begin{aligned} R' &= (R - G)\ \&\ \text{0xff} \\ B' &= (B - G)\ \&\ \text{0xff} \end{aligned}$$

これは色変換で green_to_red = 32、green_to_blue = 32、red_to_blue = 0 とした場合と同等です。パラメーターが不要であることから、単純な画像では色変換よりも効果的な場合があります。

自然画像では RGB の相関が強く、とくに緑チャンネルは支配的な相関源である傾向があります。そのため、この単純な変換だけでも効果的にエントロピーを低下できます。

4.6 カラーインデックス変換

画像内のユニークな色数が 256 色以下の場合、カラーインデックス変換を適用できます。PNG のパレットモード（カラータイプ 3）に相当する処理です。

4.6.1 カラーテーブルの構築

画像に含まれるユニークな色を抽出し、カラーテーブル（パレット）を構築します。カラーテーブルは ARGB 値のリストで格納されますが、隣接するエントリー間の差分を取ることで圧縮効率を高めます。パレット内では類似した色が隣接しやすいため、差分値は元の色値よりも小さくなる傾向があります。

4.10.4 near_lossless パラメーター

near_lossless（-near_lossless、0〜100）は、VP8L の変換パイプラインの前に適用される前処理です。可逆ではなくなりますが、知覚的にはほぼ区別できない品質でファイルサイズを大幅に削減できます。

値が 100 の場合は前処理を行いません。100 未満の場合、値に応じてピクセルの量子化ビット数（limit_bits）が 1〜5 に設定されます。

画像端のピクセルと平坦な領域は保持され、エッジ付近だけが量子化されます。

4.10.5 cwebp 可逆圧縮オプション一覧

▼表 4.16　cwebp の可逆圧縮関連オプション一覧

オプション	デフォルト	説明
-lossless	off	VP8L 可逆圧縮を有効化
-q	75	圧縮努力度（0〜100）
-m	4	圧縮方式（0〜6）
-z	off	可逆プリセット（0〜9）
-near_lossless	100	準可逆の前処理強度（0〜100） 100=完全可逆
-exact	off	透明領域の RGB 値を保持
-noalpha	off	アルファチャンネルを破棄
-mt	off	マルチスレッドエンコードを使用
-low_memory	off	メモリ使用量を削減
-metadata	none	保持するメタデータ exif / icc / xmp / all / none

第 5 章

WebP 活用法

WebP の仕組みを理解したところで、実際の活用方法に目を向けます。HTML での記述、サーバー設定による自動配信、変換ツールの使い方、そして既存フォーマットとの使い分けまで、WebP を導入するために必要な知識をまとめます。

5.1 HTML での記述方法

WebP を HTML で使う方法は、大きく二つあります。`img` 要素で直接指定する方法と、`picture` 要素でフォールバックを用意する方法です。

5.1.1 img 要素による直接指定

もっともシンプルな方法は、ふだんどおりに `img` 要素を使って WebP 画像を指定する方法です。

▼リスト 5.1　img 要素

```
<img src="photo.webp" alt="写真の説明" width="800" height="600">
```

2026 年現在、約 97% のブラウザーが WebP に対応しているため、問題なく表示されるケースがほとんどです。

5.1.2 picture 要素によるフォールバック

WebP 非対応の環境を考慮する場合は、`picture` 要素を使ってフォールバックを用意します。

▼リスト 5.2　picture 要素

```
<picture>
  <source srcset="photo.webp" type="image/webp">
  <img src="photo.jpg" alt="写真の説明" width="800" height="600">
</picture>
```

ブラウザーは source 要素を上から順に評価し、type 属性の MIME タイプに対応していれば、その画像を使用します。どれにも対応していない場合は、img 要素の画像がフォールバックとして表示されます。

5.1.3　レスポンシブ画像との組み合わせ

picture 要素は、フォーマットの切り替えとレスポンシブ対応を同時に実現できます。

▼リスト 5.3　レスポンシブ画像との組み合わせ

```
<picture>
  <source srcset="photo-small.webp 480w,
                  photo-medium.webp 800w,
                  photo-large.webp 1200w"
          sizes="(max-width: 600px) 480px,
                 (max-width: 1000px) 800px,
                 1200px"
          type="image/webp">
  <source srcset="photo-small.jpg 480w,
                  photo-medium.jpg 800w,
                  photo-large.jpg 1200w"
          sizes="(max-width: 600px) 480px,
                 (max-width: 1000px) 800px,
                 1200px"
          type="image/jpeg">
  <img src="photo-medium.jpg" alt="写真の説明"
       width="800" height="600">
</picture>
```

srcset 属性の w 記述子で画像の幅を指定し、sizes 属性でビューポートに応じた表示サイズを指定します。ブラウザーはフォーマットの対応状況とビューポートのサイズを考慮して、最適な画像を自動選択します。

5.1.4　画像の読み込み最適化

画像の読み込みを最適化するための、ベストプラクティスもあわせて紹介します。

▼リスト 5.4　画像の読み込み最適化

```
<img src="photo.webp" alt="写真の説明"
     width="800" height="600"
     loading="lazy" decoding="async">
```

width、height 属性は、画像の幅と高さを指定します。未指定でもブラウザーは画像のサイズを取得してレンダリングを行います。しかし、画像を読み込むまでレイアウトが確定しないため、ページのレイアウトが途中で変わることがあります。この現象を Cumulative Layout Shift（CLS）と呼び、ユーザー体験を損なう要因の一つです。幅と高さを指定することで、ブラウザーが画像の読み込み前でもレイアウト領域を確保できるため、CLS を防止できます。

loading 属性は、画像の読み込みタイミングを制御します。lazy を指定すると、画像に近づくまで読み込みが遅延します。ファーストビュー以外の画像に指定すると、初期のページ表示が高速化され、ユーザー体験が向上します。

decoding 属性は、画像のデコードタイミングを制御します。async を指定すると、画像のデコードを待たずに、他の内容を先にレンダリングします。loading 属性と同様に、ファーストビュー以外の画像に指定するとよいでしょう。

5.2　配信と変換

HTML での記述に加え、サーバーやツールを活用して WebP を効率的に配信、変換する方法もあります。ここではサーバーサイドでの自動配信、外部サービスの活用、変換ツールを紹介します。

5.2.1 コマンドラインツールによる変換

libwebp に付属する `cwebp` コマンドで個別に変換できます。

▼リスト 5.5　cwebp による変換

```
# 非可逆圧縮（品質80）
cwebp -q 80 input.jpg -o output.webp

# 可逆圧縮
cwebp -lossless input.png -o output.webp

# 非可逆圧縮 + Sharp YUV（色の精度を重視）
cwebp -q 80 -sharp_yuv input.jpg -o output.webp

# アルファチャンネル付き非可逆圧縮
cwebp -q 80 -alpha_q 90 input.png -o output.webp
```

5.2.2 Content Negotiation

WebP 対応ブラウザーは、画像リクエストの HTTP Accept ヘッダーに `image/webp` を含めます。

▼リスト 5.6　Accept ヘッダーの例

```
Accept: image/avif,image/webp,image/apng,image/*,*/*;q=0.8
```

このヘッダーを確認することで、適切な画像を配信できます。