3.1 HTTP/3のはじめに

1章で説明してきたとおり、HTTP/3はQUICを利用しており、QUICはパケットロスがあったとしても後続のパケットの処理が可能なようになっています。そのメリットを活かすために、パケットがロスしたり順番が入れ替わっても、回復をまたずに処理が進められるようにHTTP/3レイヤでも処理の依存関係が生まれないように工夫がされています。その点を踏まえつつ、HTTP/3について詳しく見ていきます。

まず、クライアントとサーバがどのようにHTTP/3通信を開始するのか。その次に、QUICの提供する単方向ストリーム・双方向ストリームをどのように使用し、どのようにHTTPメッセージをやりとりしているのか見ていきます。その後、HTTP/2の頃にもあった、サーバプッシュやヘッダ圧縮がHTTP/3どのように実現されているかを確認していきます。

HTTP/2の頃にあったストリームの制御やフロー制御といった機能はQUICレイヤで行われるようになっているためHTTP/3の仕様には含まれていません。その他にHTTP/3の仕様に含まれてない機能として、HTTP/2の頃にはあった優先度制御の仕組みは拡張として別の仕様で定義されるようになりました。拡張仕様ではありますが、その優先度制御の仕様については本章で扱います。

この章では、フレームという用語が出てきますが、これはHTTP/3レイヤのフレームです。単にフレームといった場合、HTTP/3のフレームを指します。明示的に呼び分ける際は、QUICレイヤのフレームをQUIC フレーム、HTTP/3レイヤのフレームをHTTP/3 フレームと呼びます。

3.2 HTTP/3対応の通知とコネクションの開始

HTTP/3では、サーバがHTTP/3に対応していることをクライアントに通知し、クライアントがHTTP/3で通信を開始しています。まずは、その流れを見ていきます。

「RFC7838 HTTP Alternative Services」という仕様を用いて、サーバが自身がHTTP/3をサポートしていることを通知します。この仕様は、通称alt-svcと呼ばれ、サーバが自身のサービスを別のドメイン・ポート・プロトコルで提供できることをクライアントに通知するための仕様です。

たとえば、HTTPレスポンスヘッダを用いて以下のように送信することで、サーバが50781ポートでHTTP/3を提供できることを示しています。HTTPレスポンスヘッダを用いて通知するため、クライアントは一旦HTTP/1.1もしくはHTTP/2で通信を行ってから、HTTP/3の通信を開始することになります。

```
Alt-Svc: h3=":50781" ; ma=3600
```

このレスポンスヘッダを受けとったクライアントは、クライアント自信がHTTP/3に対応している場合は指定されたエンドポイントに対して通信を試みます。もちろん、通信経路上のファイアウォールなどのネットワーク機器によってブロックされる可能性もあるため、通信がうまくできない場合はHTTP/2などにフォールバックされます。

また、Alt-Svcヘッダは複数指定できます。またdraft版 HTTP/3はh3-xxで指定することで、HTTP/3のdraftバージョンの対応を通知することができます。

```
Alt-Svc: h3=":443", h3-09=":8009"
```

alt-svcには、HTTPレスポンスヘッダで通知する他に以下の方法で通知できます

- HTTP/2のALTSVCフレーム (RFC7838)
- DNSで通知する(HTTPVSSVCレコード draft-ietf-dnsop-svcb-httpssvc)

この際使用した、h3やh3-09といった識別子は、QUICのハンドシェイク中でもALPNの値と指定されます。

HTTP/2のときは、極力1つのTCPコネクションを活用してHTTPリクエストを送信していました。HTTP/3でも同様です。サーバ証明書が有効であれば既存コネクションを利用できるため、コネクション確立時に*.example.com のワイルドカード証明書を提供された場合は、www.example.comやtest.example.comへのHTTPリクエストをそのコネクション上で送信できます。

3.3 QUICストリームの利用

HTTP/3では、QUICの単方向ストリームおよび双方向ストリームを使い分けます。
単方向ストリームはさらに、HTTP/3及びQPACKで使い分けられます。

双方向ストリーム

- Request Stream: HTTPメッセージをやり取りするのに使用されます。

単方向ストリーム

Stream Type	Stream Type 値	説明
Control Streams	0x00	コネクションの制御情報を送信する。パラメータの送信やエラー通知などに使用される
Push Stream	0x01	サーバプッシュでリソースをプッシュするのに使用される。Push ID
QPACK Encoder Stream	0x02	QPACKで使用される
QPACK Decoder Stream	0x03	QPACKで使用される
Reserved Stream	(0x1f * N) + 0x21	相手が、不明なストリームタイプを無視するという要件を正しく無視するか確認するのに使用していい予約ストリームタイプ。

単方向ストリームは複数の用途で使用されます。単方向ストリームをオープンしたときに、どの用途で使用するのか、Unidirectional Stream HeaderとしてStream Type 値を送ります（Push Stream の場合は各サーバプッシュを識別するためのPush IDも続けて送信されます）。それ以後はHTTP/3フレームを送信します。

```
Unidirectional Stream Header {
  Stream Type (i),
}
```

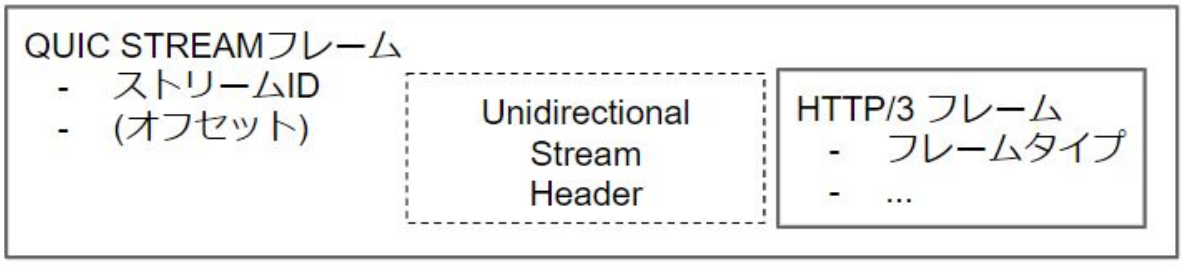
各単方向ストリームの使われ方については、以降用途の中で説明します。

双方向ストリームは、HTTPメッセージの送受信に使われるRequest Streamのみが定義されており、常にクライアントからオープンされます。Request StreamはHTTPリクエストとHTTPレスポンスを1つずつ送受信したらクローズされ、再利用はされません。

なお、それ以外の単方向ストリーム及びサーバからの双方向ストリームはHTTP/3の仕様では定義されていませんが、拡張仕様によって使用される可能性があります。

3.4 HTTP/3 フレーム

HTTP/3レイヤでも各メッセージはフレームというメッセージ単位で送信されます。



各フレームは、Type値を持っておりこの値によって各フレームは識別されます。

HTTP/3のフレームはHTTPメッセージを送信するものや通信を制御するのに使用され、用途ごとにタイプが別れています。各フレームはそれぞれストリーム上で送受信されますが、使用できるストリームタイプは決められています。

フレームは下記の7種類および予約されたものがあります。

Frame	Type Value	Control Stream	Request Stream	Push Stream	説明
DATA	0x0	No	Yes	Yes	HTTPボディを送信するのに使用される
HEADERS	0x1	No	Yes	Yes	HTTPヘッダを送信するのにしようされる
CANCEL_PUSH	0x3	Yes	No	No	サーバプッシュをキャンセルするよう要求するのに使用される。
SETTINGS	0x4	Yes	No	No	コネクションに関わるパラメータを送信するのに使用される
PUSH_PROMISE	0x5	No	Yes	No	サーバプッシュのPush IDやリクエストヘッダを送信する
GOAWAY	0x7	Yes	No	No	エラーを通知し、コネクションを切断するのに使

					用される
MAX_PUSH_ID	0xD	Yes	No	No	プッシュ可能な上限IDを通知するのに使用される
Reserved	0x2, 0x6 0x8, 0x9	Yes	Yes	Yes	予約されたフレーム

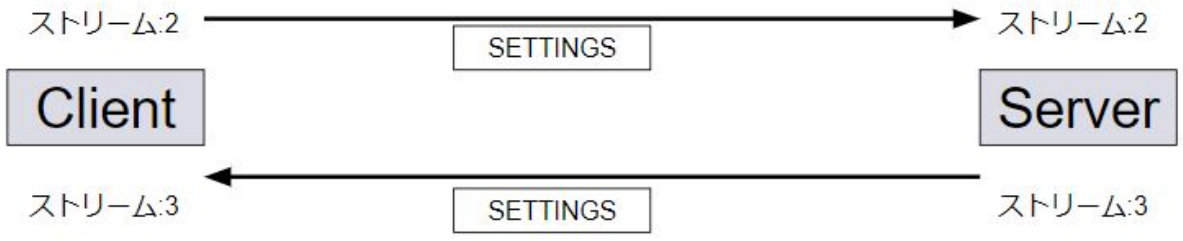
3.5 HTTPメッセージの送受信

ここでは、QUICのコネクションが確立したあとに、HTTPメッセージの送受信をするまでの準備から、HTTPメッセージをストリーム上でどのように通信するか見ていきます。また、補足としてCONNECTメソッドの扱いに触れます。

3.5.1 HTTPメッセージを送るまで

QUICのコネクションが確立すると、まず最初にクライアントとサーバは相互にControl Streamsをオープンします。Control Streamsは相互に必ずただ1つオープンしている必要があり、クローズしてはいけません。

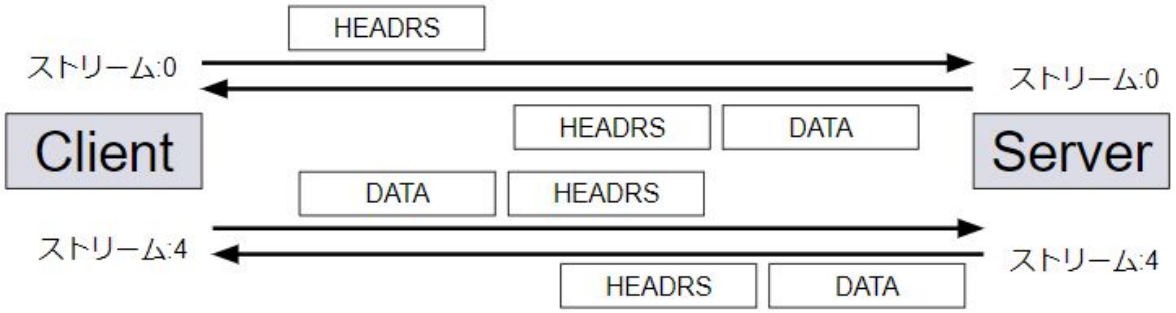
さらに、Control Streams上でまずSETTINGSフレームを送信する必要があります。このSETTINGSフレームにはMAX_FIELD_SECTION_SIZEといったHTTPヘッダサイズ上限のパラメータや、QPACKで利用されるパラメータが含まれます。なお、各エンドポイントは相手からのSETTINGSフレームの受信を待たずに後続のデータ送信ができます。



ここまでが、HTTPメッセージをやりとりするまでの前準備になります。このあとHTTPメッセージのやり取りが開始されます。

3.5.2 HTTPメッセージの送受信

クライアントは双方向ストリームを用いてHTTPメッセージの送信をします。HTTPメッセージは、HTTPヘッダを格納するHEADERSフレームと、HTTPボディを格納するDATAフレームを用いて送受信されます。



HTTPメッセージは、QUICコネクション上で複数の双方向ストリームを使用して並列的に行われます。QUICの章で説明したとおり、QUICパケットの欠損や順番の入れ替わりは回復されますが、その影響を受けなかったストリームは回復を待たず後続のパケットを処理することができます。

HEADERSフレームはHTTPヘッダを格納しますが、その形式はヘッダ圧縮の仕組みであるQPACKで定義されます。また、HTTPメソッドやHTTPステータスコードは、HTTP/1.1ではヘッダという扱いではありませんが、HTTP/2よりHPACKで効率よく表現する都合、疑似ヘッダという形で扱われるようになっています。通常のヘッダと異なり、ヘッダ名が “:” (コロン)で始まっています。

HTTP/3でも同様にこれらの疑似ヘッダを扱います。

下記の疑似ヘッダを使用します

- HTTPリクエスト
  - ":method" HTTPリクエストメソッド
  - ":scheme" HTTPリクエストURLのスキーム
  - ":authority" HTTPリクエスト先のドメイン、hostヘッダ相当。
  - ":path" HTTPリクエストURLのPATH
- HTTPレスポンス
  - “:status” レスポンスのステータスコード

HTTP/1.1の下記のリクエストは

```
GET /resource HTTP/1.1
Host: example.org
Accept: image/jpeg
```

疑似ヘッダを用いて下記のように表現されます。なお、HTTP/2およびHTTP/3ではすべてのHTTPヘッダ名は小文字を使用しなければなりません。

```
:method = GET
:scheme = https
:path = /resource
```

```
host = example.org
accept = image/jpeg
```

3.5.3 CONNECTメソッド

HTTPにはCONNECTメソッドというメソッドが定義されています。その扱いは特殊ですので、簡単に補足します。CONNECTメソッドはProxyサーバに接続し、最終的な通信相手であるサーバに対して通信データをトンネリングするようになります。HTTP/1.1ではコネクション全部のデータをトンネリングしますが、HTTP/2とHTTP/3ではCONNECTメソッドのリクエストを送ったストリームのデータがトンネリングされるようになります。

HTTP/3のCONNECTメソッドではリクエストヘッダは下記のように指定されます

- :method 疑似ヘッダは“CONNECT”を指定する
- :schemeと:path 疑似ヘッダは省略される
- :authority 疑似ヘッダは、トンネリング先のホスト異名とポートが指定されます。

このCONNECTメソッドを受け取ったProxyは、指定されたホストに対してTCPコネクションを確立します。プロキシはクライアントからそのストリーム上で送られてくるDATAフレームを、サーバとのTCPコネクションに中継します。逆も同じように中継されます。プロキシはクライアントとのストリームが終了されたら、中継先とのTCPコネクションを切断します。逆にサーバとのTCPコネクションが切断された場合も、クライアントとのストリームも終了します。

3.6 ヘッダ圧縮 QPACK

user-agentヘッダや、accept-encodingヘッダといったヘッダは、ほとんど変わることがないのに、HTTPリクエストの都度送信されていました。HTTPレスポンスに関しても、同一のサーバからであればほとんど変わらないヘッダがあります。このようなヘッダ領域を圧縮したいという議論がHTTP/2のころからありました。

HTTP/2ではHPACKというHTTPヘッダ圧縮の仕組みを使用していました。詳細については、RFC 7541 「HPACK: Header Compression for HTTP/2」を参照してください。HPACKはHTTP/2のためのものであり、TCPが使われる前提でした。TCPでは、クライアントがデータを送信した順番で、受信側のアプリケーションが処理をする前提がありましたが、HTTP/3ではそうではありません。ストリーム0, ストリーム4上の順番でHEADERSフレームを送信したとしても、ストリーム4, ストリーム0という順番で処理される可能性もあります。

そのため、HTTP/3でもそのままHPACKを使ってしまうと、ストリームをまたがって処理順番に依存関係ができてしまい、パケットロス時にそのデータが再送されるまでの待ち時間が発生されます。

そこで新しくHTTP/3用でQPACKという仕様が登場しました。

QPACK: Header Compression for HTTP/3
- https://tools.ietf.org/html/draft-ietf-quic-qpack-14

QPACKはHPACK同様、以下の2つを組み合わせてHTTPヘッダを表現します。
- ハフマン符号
- 静的テーブル、動的テーブル

なお、これらのハフマン符号やテーブルを使う使わないは送信者が選択します。使用しない場合は通常通り文字列でヘッダ名、ヘッダ値は表現されます。

3.6.1 ハフマン符号

ハフマン符号を用いて各HTTPヘッダ名と値の文字列を表現します。

ハフマン符号は各文字の出現頻度に合わせてbit表現を変更します。例えば出現頻度の多い'a'は0b00011を割り当て 5bitで表現できますが、出現頻度の低い'q'は 0b1110110 が割り当てられており 7bit での表現となります。このハフマン符号を用いてヘッダ名やヘッダ値の文字列(リテラル)を短く表現できます。

英数字や記号を含む各文字とハフマン符号の対応表はRFC7541 HPACKの仕様中に定義されており、通信中に変更されるものではありません。

ハフマン符号を使用しない場合は文字列として送信されます。

3.6.2 静的テーブル、動的テーブル

テーブルとは、辞書データのようものです。テーブルにはindex番号毎に、ヘッダ名だけ、もしくはヘッダ名とヘッダ値の両方が格納されています。各index番号, ヘッダ名, ヘッダ値のペアをエントリといいます(表の各行に相当します)。indexは0番からはじまります。

Index	Name	Value
0	:authority	
1	:path	/
2	age	0

...	...	..
31	accept-encoding	gzip, deflate, br
...	...	...

例えば、下記のHTTPリクエストヘッダを送信する場合、テーブルの31番に同様のデータがあるため、HTTPヘッダをそのまま送るのではなく”index:31”という情報をおくるだけで下記のHTTPヘッダを表現できます。

```
accept-encoding: gzip, deflate, br
```

なお、ヘッダ名のみテーブルindexを参照し、ヘッダ値は文字列(ハフマン符号する)のような使い方もできます。

このテーブルには2種類あり、静的テーブルと動的テーブルです。

- 静的テーブル: QPACKの仕様中で事前に定義されているテーブル
- 動的テーブル: 1つのコネクション中に更新されていくテーブル

HPACKとは異なり、静的テーブルと動的テーブルともにindexは0番からはじまります。

静的テーブルは事前に定義されており、よく使うであろうヘッダが格納されてます。なお、HPACKも静的テーブルを持ちますが当時とはよく使うヘッダが異なっており、刷新されています。静的テーブルは汎用的なものしか入っておらず、例えばuser-agentヘッダの値といったクライアントによって値が異なるものには使用できません。

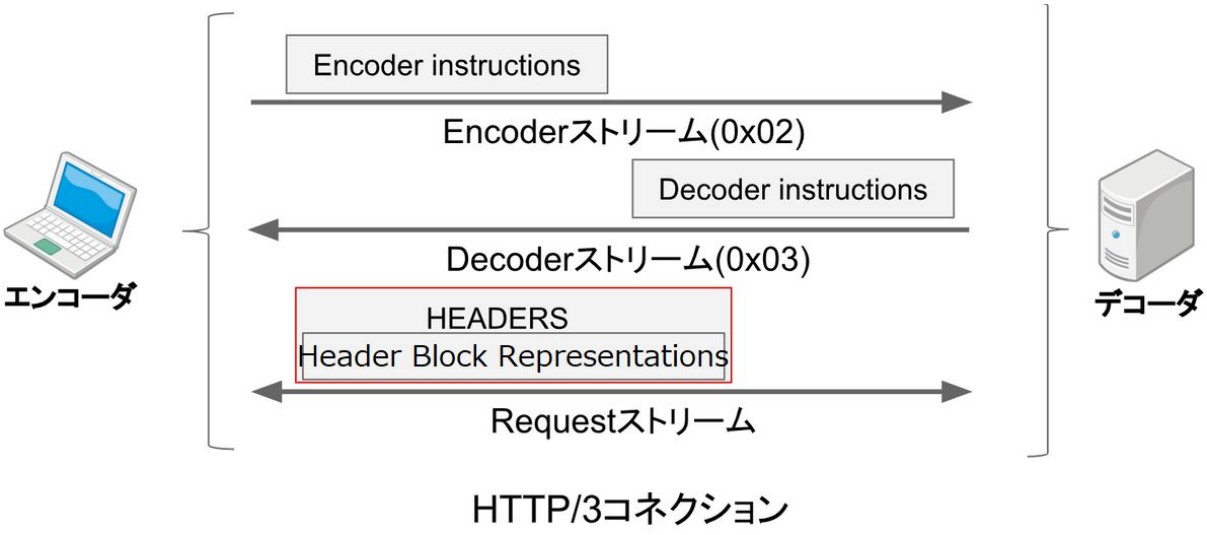
動的テーブルは、1つのコネクション中に更新されていくテーブルであり、クライアントやサーバごとに異なるヘッダも個々に追加していき、indexでHTTPヘッダを表現できるようにします。

この動的テーブルはヘッダを送信する側と受信する側で状態が一緒である必要があります。HEADERSフレームで参照するエントリがずれてしまうと、HTTPヘッダを正しく解釈できなくなってしまいます。

クライアントとサーバそれぞれ両方の立場があり、HTTPリクエストを送る場合は、クライアントがエンコーダ、サーバがデコーダです。HTTPレスポンスを送る場合は、サーバがエンコーダ、クライアントがデコーダです。

エンコーダとデコーダで齟齬なく動的テーブルを更新し参照するために、単方向ストリームのEncoderストリーム・Decoderストリームをそれぞれ1つだけ使用しま

す。1つの単方向ストリーム上で動的テーブルの操作を指示するため、その命令の順番は入れ替わることはなく、エンコーダが送った順番通りデコーダ側で処理されます。

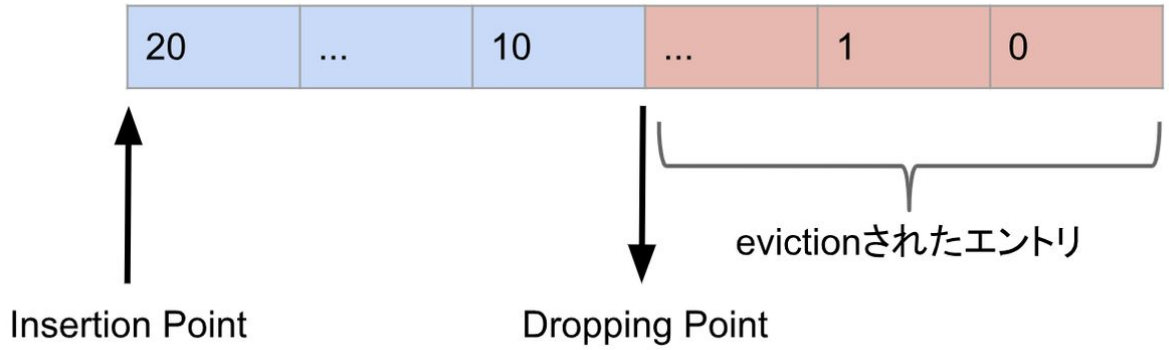


Encoderストリーム上で、Encoder instructionsを送信して動的テーブルにエントリを追加していきます。Encoder側が送信するHEADERSフレームのHeader Block Representationsから動的テーブル上のエントリを参照して使うことができます。

Encoder instructionsには下記の4つがあります

- Set Dynamic Table Capacity: 動的テーブルのキャパシティを設定する
- Insert With Name Reference: エントリを追加する。ヘッダ名は既存のテーブルにあるものをindexで指定し、ヘッダ値は文字列で指定する
- Insert Without Name Reference: エントリを追加する。ヘッダ名とヘッダ値を文字列で指定する
- Duplicate: 動的テーブル内にあるエントリを追加し直す

動的テーブルへのエントリ追加する際のindexは1つずつ単調増加で増えていきます。indexが飛ばされることはありません。そして、first-in, first-outであり古いものから順番に削除されていきます。



EncoderストリームとRequestストリームではストリームが異なるため、HEADERSフレームとEncoder instructionsでは、送信した順番でデコーダに受信されるとは限りません。仮に、Encoder instructionsを運ぶパケットがロスし、デコーダ側が受け取ったHEADERSフレームの処理をする際に参照先のエントリが存在しないということがあります。この際デコーダは、HEADERSフレームを処理するのに必要なEncoder instructionsが届くまで待ちます

また、動的テーブルはキャパシティに達していくと古いエントリから削除(eviction)されていきます。

エンコーダ側が特定のエントリを参照するHEADERSフレームを送ったのに、デコーダ側でそのHEADERSフレームを処理しようとしたらそのエントリがすでに削除されていては処理を継続することができません。

このような不整合が起きないようにDecoderストリームでフィードバックを送信します。

- Header Acknowledgement: 動的テーブルを参照するストリームを処理した際に、そのストリームIDを示す。
- Insert Count Increment: 以前に送信してから、動的テーブルに追加されたエントリの増分を示す。
- Stream Cancellation: ストリームをキャンセルした(動的テーブルを参照せず、不使用となる)

エンコーダはこのフィードバックを受け取ることで以下のことが行なえます
- 特定エントリを参照するHEADERSフレームがデコーダに処理されたことがわかるので、動的テーブルのエントリをevictionすることができる (もともと各エントリを参照するHEADERSを送った数をカウントしておく)
- Encoder instructionsがデコード側で処理されて、エントリが追加された。これによって、HEADERSフレームから参照してもデコーダ側で待ち時間が発

生することはない。(待ち時間が発生しうるHEADERSフレームを送信しても良い)

3.6.3 ヘッダ表現

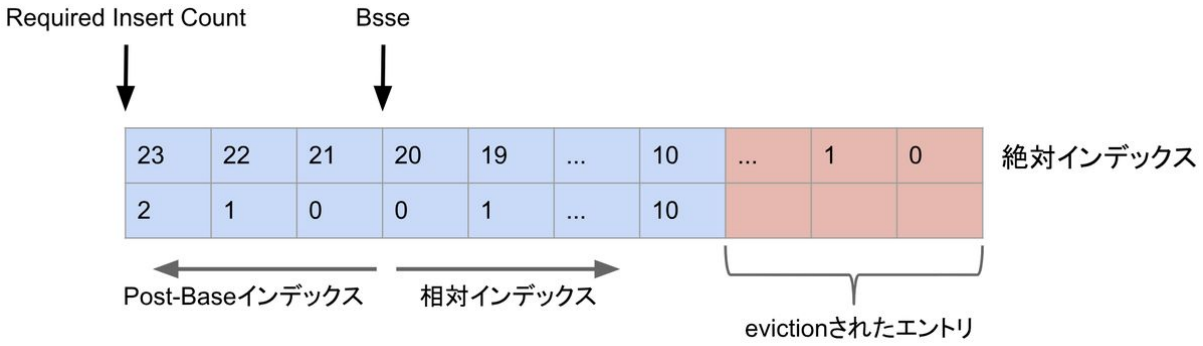
HTTPヘッダ群は上記で説明したような、ハフマン符号化やテーブルへの参照を使って「Header Block Representations」という形式で表され、HEADERSフレームに格納されます。

Header Block Representationsには、このHeader Block Representationsを処理するのに必要な動的テーブルエントリの追加回数が記載されています。デコーダは予め動的テーブルへのエントリの追加数をカウントしておくことで、Header Block Representationsを解釈できるかが予め分かる仕組みになっています。追加数のカウントが足りない場合は、パケットの順番が入れ替わっており、デコーダは処理可能となる追加のEncoder instructionsが届くのを待ちます。

Header Block Representationsでは、1つのHTTPヘッダは以下の5つのいずれかの方法で表現されます。通常HTTPメッセージを送る際は複数のヘッダを送りますが、それぞれが違う表現方法を使って表現しても良いです。テーブルを参照する際は別途静的テーブルを参照するか、動的テーブルを参照するか指定されます。なお、文字列で表現する場合はハフマン符号化するかしないか選択できます。

- Indexed Header Field: テーブルのindex番号のみでHTTPヘッダが表現される
- Indexed Header Field With Post-Base Index: テーブルのPost-Base Indexのみでヘッダが表現される
- Literal Header Field With Name Reference: ヘッダ名のみテーブルのindex番号で表現され、ヘッダ値は文字列で表現される。
- Literal Header Field With Post-Base Name Reference: ヘッダ名のみテーブルPost-Base Index番号で表現され、ヘッダ値は文字列で表現される。
- Literal Header Field Without Name Reference: ヘッダ名とヘッダ値ともに文字列で表現される。

indexの参照方法には、2種類のIndex指定方法があります。Post-Baseインデックス、相対インデックスです。HTTPヘッダをエンコードする際に、絶対インデックスでBaseを決めます。そのBaseからどちらの方向の番地を参照するのかによってPost-Baseインデックス、相対インデックスを使うのかが異なってきます。



このPost-Baseインテックスにより、エンコーダがヘッダ郡をエンコードする処理を1回のループ(ワンパス)で処理できるよになります。HTTPメッセージのヘッダ群をエンコードする際に途中でEncoder instructionsを送ることでエントリーを追加し、エンコードしているHeader Block Representationsから追加したエントリを参照できるようになります。

3.7 サーバプッシュ

HTTP/2ではサーバプッシュという機能があり、HTTP/3でもこの機能はサポートされています。

HTTP/1.1ではHTTPリクエストがないと、サーバからHTTPレスポンスを返すことはできませんでした。サーバプッシュはHTTPリクエストがなくてもHTTPレスポンスを送る機能です。

例えば、以下のようなHTMLファイルへリクエストがあった場合のことを考えます。

```
<html>
  <body>
    <img src=”./1.jpg” />
    <img src=”./2.jpg” />
    <img src=”./3.jpg” />
    <img src=”./4.jpg” />
  </body>
</html>
```

サーバはこのHTMLファイルへのリクエストを受け取ると、クライアントは次に画像類を要求してくることが予想できます。そういった場合に、サーバ側から画像類のHTTPレスポンスを先んじて送信することで、クライアントはより早くHTMLページを表示できるようになります。

HTTP/3におけるサーバプッシュは、コンセプトはHTTP/2と似ていますがPush Streamを使う点などいくつかの違いがあります。サーバプッシュの具体的な流れを見ていきましょう。

HTTP/3においては、HEADERSフレームを受け取ったサーバが、そのRequestストリーム上でPUSH_PROMISEフレームを送信し、サーバプッシュに使用するPush IDを通知します。

PUSH_PROMISEフレームには以下の内容が含まれています

- Push ID: HTTPレスポンスを送るPush Streamと紐付けるためのID
- Encoded Field Section: QPACKでエンコードされたHTTPリクエストヘッダ

PUSH_PROMISEフレームにはHTTPリクエストヘッダが含まれています。このリクエストヘッダには、:authorityや:pathといった疑似ヘッダが含まれており、サーバプッシュするレスポンスが、そもそもどのURLへのリクエストに対するものなのかが記述されます。

サーバはPUSH_PROMISEフレームを送信したあと、Push Streamを開き Push IDを送信し、このPush StreamがどのPUSH_PROMISEフレームと紐づくのか示します。そのあと、そのPush Stream上でHEADERSフレームとDATAフレームでHTTPレスポンスを送信します。

クライアントはPUSH_PROMISEフレームを受け取ると、このサーバプッシュを受けるか判断します。すでにそのURLのキャッシュを持っている場合などは、サーバプッシュを受け付ける必要はありません。そこでCANCEL_PUSHフレームを送信し、サーバプッシュのキャンセルを要求します。CANCEL_PUSHにはキャンセルするPush IDが含まれています。

クライアントがサーバプッシュを受け付けるという判断をし、無事HTTPレスポンスを受信した場合は、そのリソースはキャッシュ領域に格納され利用されます。

HTTP/2の頃はサーバサイドのアプリケーションからミドルウェアにプッシュを指示するのにLinkヘッダを使用していました。下記のヘッダを解釈できるミドルウェアはこのHTTPレスポンスヘッダを処理する際に、指定されたリソースをサーバプッシュします。

```
Link: </app/style.css>; rel=preload; as=style; nopush
Link: </app/script.js>; rel=preload; as=script
```

おそらくHTTP/3でも同様のことができるでしょう。

3.8 優先度制御

HTTP/2からは、1つのコネクション上で複数のHTTPリクエストを並列的に送信することができるようになりました。それらHTTPリクエストのうち、Webページのレンダリングを開始するのにCSSは早くほしい、画像は遅れても良いといったように優先度が異なっている場合があります。このような優先度をクライアントからサーバに指示できるような仕組みがHTTP/2にはありました。

HTTP/2の優先度制御は、HEADERSフレームやPRIORITYフレームで各ストリームの優先度を指示していました。優先度は、dependencyとweightという形で表現され、ストリームAの処理が終わったらストリームBといった指示ができました。このDependency Treeと呼ばれる方式は複雑であり、よりWebに適した方式が模索されました。

そのような議論の中で、HTTP/3の仕様からは優先度制御の定義は外され、拡張仕様として定義されました。それが下記仕様です。

「Extensible Prioritization Scheme for HTTP」
- https://tools.ietf.org/html/draft-ietf-httpbis-priority-00

この仕様では、2つのパラメータで優先度を指定します。
- Urgency: 0~7 までの優先度 (値が小さい方が優先度が高い)。uと表現される
- Incremental: このリソースがIncrementalに処理可能かどうか。iと表現される

これらの優先度はHTTPヘッダで送信され、例えば、CSSなど優先度の高いものはUrgency=0と指定されます。

```
priority = u=0
```

画像といった、Incrementalに処理し活用(表示)できるものはIncrementalのフラグが示されます

```
priority = u=5, i
```

このようにHTTPリクエストの優先度はリクエストヘッダで示されますが、サーバがレスポンスを返すのに選択した優先度は同じようにレスポンスヘッダで示されます。

また、HTTPレスポンスのボディを受信中に、クライアントから優先度を上げたい/下げたいというケースもあります。例えば、モダンなブラウザではタブ機能を有しているものがありますが、表示しているタブが変われば優先度を変えるということ

もできます。その場合は、この仕様で定義されている、PRIORITY_UPDATEフレームで該当のストリーム優先度を変えるよう指示することができます。

3.9 コネクションの終了

コネクションを正常終了する際はHTTP/3レイヤでまずGOAWAYフレームを送信します。GOAWAYフレームはこのコネクションでこれ以上の通信を受け付けないことを相手に通知します。もちろん受信中のHTTPメッセージはやりとりが完了するのを待ちます。

正常終了する際や、コネクションを維持できないエラーがあった際はQUICレイヤでの切断を行います。QUICレイヤでCONNECTION_CLOSEフレームを送信することでコネクションを終了します。アプリケーションレイヤによるクローズを示す0x1dタイプのQUICのCONNECTION_CLOSEフレームが使用されます。この時、エラーコードを付加して相手に送ります。ここでは、HTTP/3で定義されているエラーコードの一部を紹介します。

- H3_NO_ERROR: 正常終了を示す
- H3_GENERAL_PROTOCOL_ERROR: 具体的には示さないが、仕様違反により切断する
- H3_INTERNAL_ERROR: HTTPスタック内のエラー
- H3_MISSING_SETTINGS: 必要なSETTINGSフレームが送られてこなかった
- H3_VERSION_FALLBACK : HTTP/3でリクエストに答えられない(HTTP/2以下をしようするように)

3.10 拡張性

HTTP/2同様、HTTP/3では拡張性を有しています。フレームやストリームタイプ、SETTINGSパラメータ、エラーコードは拡張仕様で追加していいことになっています。そのため、各エンドポイントは予期せぬタイプ値のフレームやストリームタイプを受け付ける可能性がありますが、単純に破棄するように仕様で決まっています。

これにより、特定の拡張仕様をサポートしていないエンドポイントに対してそれらを使用しても単純に無視されるだけでコネクションを破壊しません。

なお、明示的に相手が特定の拡張仕様をサポートしていることが知りたい場合は、SETTINGSパラメータで明示的にその仕様をサポートしていることを相手とネゴシエーションすることもできます。