インターネット 10min 分 講座

おさえておきたい基本や、最新動向を解説するコーナーです。

HTTP/2とは
～Webの高速化が期待できるプロトコル～

HTTPはバージョン1.1が1997年に制定されてから、長きにわたって使われ続けているプロトコルです。シンプルで使いやすいプロトコルですが、より効率を重視し、速度向上を見込める新しいバージョンとして、HTTP/2が普及しつつあります。本稿ではHTTP/2で使われている技術を中心に、高速化の手法などを解説します。

1 HTTP/2の目標

■HTTP/2は何を解決するのか

HTTP/2はRFC7540「Hypertext Transfer Protocol Version 2（HTTP/2）」として標準化されています。すでに多くのブラウザやHTTPサーバでサポートされており、実際に広く使われる技術となっています。

HTTP/2の標準化が開始されたのは2012年頃で、Webアプリケーションがよりリッチになり、1ページあたりのリソース数も各リソースのサイズも大きくなってきていました。それに伴い、Webのパフォーマンスについて重要度が増している時期でした。そのためWebアプリケーションレイヤでさまざまなノウハウが生み出され、Webの技術はより複雑になってきていました。

そこでGoogle社は、HTTPのメッセージをより効率よく転送する「SPDY」というプロトコルを開発し、自社のプロダクト群へディプロイすることで、プロトコルレイヤでの改善の効果を確かめていました。

SPDYのインパクトもあり、IETFのhttpbis WGではHTTP/2の標準化を開始します。HTTP/2の目標が「HTTPメッセージのセマンティクスを維持し、パフォーマンスとセキュリティを改善する」であった通り、HTTP/2ではパフォーマンスを改善するための多くの機能が盛り込まれています。

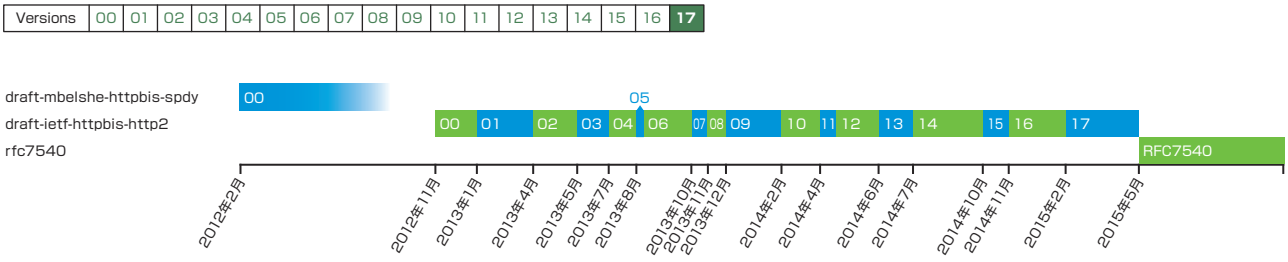
2 歴史

■SPDYからHTTP/2へ

2012年ごろ、HTTP/2の標準化を開始するにあたって、標準化の開始点となるような仕様の提案がいくつか提出されました。その一つであったSPDY（version 3）は、すでにGoogle社によってサービスへディプロイされており、実績がある点などの理由からHTTP/2の開始点として選ばれました。2012年11月に提出されたHTTP/2（当時はHTTP/2.0）の最初のInternet-Draft（https://tools.ietf.org/html/draft-ietf-httpbis-http2-00）がSPDY Protocolとなっていることからも、SPDYを受け継いでいることがわかります。

HTTP/2はSPDYの特徴を残しつつ、多くの改善が行われています。この変更の多くはSPDY/4へと取り込まれ、Google社側でもSPDYの開発は続けられていきました。17回にわたる草案改版の後、2015年11月にHTTP/2の標準化完了をもってSPDYはその役目を終え、Google社のプロダクト群もHTTP/2を使うようになりました。図1

図1 HTTP/2 Internet-Draftの歴史



3 技術概要

HTTP/2はHTTPメッセージを効率よくやり取りするために、多くの機能を持っています。例えば、HTTP/1.1ではHTTPリクエスト・HTTPレスポンスをするたびにTCPコネクションを切断していました。これでは、リクエストのたびにTCPの3ウェイハンドシェイクを行うことになりますし、HTTPSを使っていればそこにTLSハンドシェイクのオーバーヘッドも加わります。また、一度にHTTPリクエストを送る時は六つ（実装依存）のTCPコネクションを利用していました。生存時間の短いTCPコネクションをいくつも使うというのは、スロースタートであるTCPとの相性が悪く帯域を効率よく使えません。

HTTP/2では、一つのTCPコネクションを使い回し、その中でHTTPリクエストとHTTPレスポンスを多重化することで、複数のHTTPメッセージを上限なく並列的にやりとりすることができます。多重化を実現するために、HTTP/2では仮想的な通信単位であるストリームという概念を導入しており、ストリームごとにフレームというメッセージをやりとりします。

その他にも、効率を上げるためにヘッダ圧縮、優先順位、サーバプッシュと行った機能を持っています。ここでは、そういった以下の機能を紹介していきます。

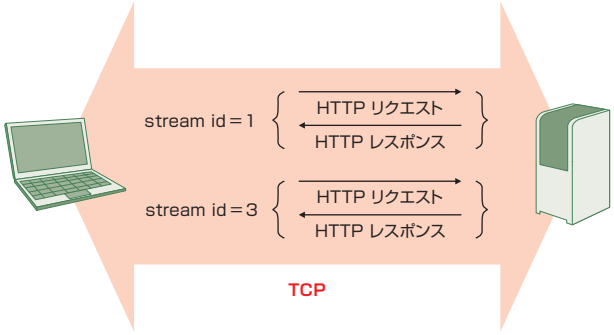
- フレームとストリーム（多重化）
- ネゴシエーション
- コネクションの再利用
- ヘッダ圧縮
- 優先順位
- サーバプッシュ
- セキュリティの向上

■フレームとストリーム（多重化）

HTTP/2では一つのTCPコネクション上で、複数のHTTPリクエストとHTTPレスポンスをやりとりします。それらのやりとりを管理するための概念がストリームです。わかりやすく説明すると、一対のHTTPリクエストとHTTPレスポンスが一つのストリームに所属します。図2 一度使ったストリームは再利用されることはなく、HTTPレスポンスが返ってくるとストリームは使われなくなります。

ストリームには一意のIDがあり、ストリームIDと呼ばれます。クライアントから開始したストリームは奇数のストリームIDを、後述するサーバプッシュによってサーバから開始されるストリームは偶数のストリームIDを使用します。これによって、ストリームIDは常にユニークです。また、ストリームID 0は、コネクション自体を意味し特別に使用されるため、HTTPメッセージのやりとりには使用されません。

図2 HTTPリクエスト・HTTPレスポンスとストリーム



優先度処理などHTTP/2の機能のいくつかはストリームごとに行われるため、このストリーム単位で制御されます。そのため、このストリームというのはHTTP/2の大事な概念の一つです。

次にフレームの説明をします。HTTP/1.1ではテキスト形式のメッセージでしたが、HTTP/2ではフレームというバイナリ形式のメッセージをやりとりします。フレームは使用用途ごとに10種類のフレームタイプが定義されています。表1 伝達するデータの種類ごとに使用するフレームが異なります。例えば今までのHTTPリクエストやHTTPレスポンスは、HEADERSフレームとDATAフレームを組み合わせて表現されます。

その他にもHTTP/2では通信の制御や優先度処理・サーバプッシュのためにさまざまなフレームを使用します。

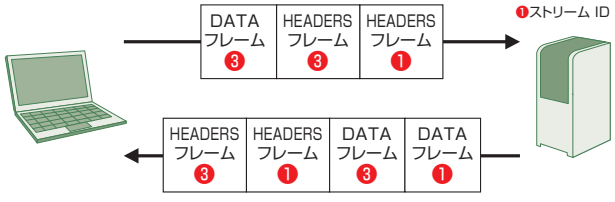
表1 HTTP/2で定義されているフレームタイプ

フレーム名	フレームタイプ番号	説明
DATA	0	POSTリクエストのデータや、HTTPレスポンスのデータといった、HTTPボディを転送するのに使用される。
HEADERS	1	HTTPリクエストヘッダや、HTTPレスポンスヘッダといったヘッダを送信するのに使用される。
PRIORITY	2	ストリームの優先度を変更するのに使用される。
RST_STREAM	3	エラーが有った場合など、エラーでストリームを終了するのに使用される。
SETTINGS	4	並列ストリーム上限など、コネクションに関するパラメータをやりとりするのに使用される。
PUSH_PROMISE	5	サーバプッシュに用いるストリームを予約するのに使用される。
PING	6	コネクションが維持できることを確認するために使用される。
GOAWAY	7	コネクションを切断するのに使用される。
WINDOW_UPDATE	8	フロー制御で使用される。
CONTINUATION	9	続けてHTTPヘッダデータを送信するのに使用される。

すべてのフレームは、どのストリームのデータなのかを示すストリームIDを持っています。HEADERSフレームやDATAフレームは所属しているストリームIDを含みます。

HTTPリクエストとHTTPレスポンスはストリームとして管理され多重化され送信されますが、実際に一度に並べて送信することはできません。各HTTPメッセージはフレームとして直列化した上で送信します。図3 それぞれのフレームはストリームIDを持っているため、どのストリームに関するやりとりなのかわかる仕組みになっています。

図3 実際の通信では、各フレームがバラバラにやりとりされます。ストリームIDを手がかりに、それぞれのストリームが復元されます。



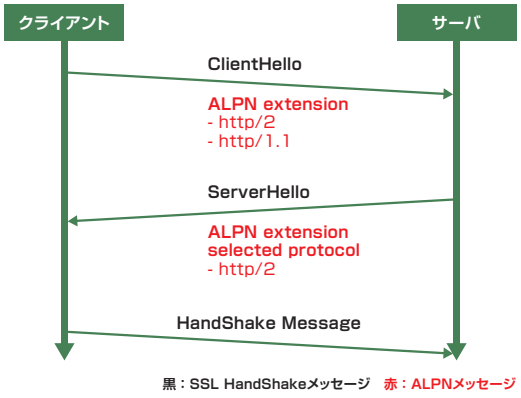
■ネゴシエーション

HTTP/2はフレームというメッセージ形式でメッセージを送信するため、HTTP/1.1と互換性はありません。HTTP/2に対応していないサーバにHTTP/2のメッセージを送信しても正しく解釈できないでしょう。しかし、HTTP/2でもHTTP/1.1と同様に80番ポートと443番ポートを使用します。そのためHTTP/2通信を開始するためには相手とHTTP/2の使用を合意する必要があります。仕様ではいくつかの通信開始方法が定義されています。

1. ALPNを使用する
2. HTTP/1.1からアップグレードする
3. ダイレクトで開始する

一つめのALPNとは、RFC7301 Transport Layer Security（TLS）Application-Layer Protocol Negotiation Extensionで定義されているTLS拡張です。ALPNでは、図4のようにTLSハンドシェイク中に使用するアプリケーションレイヤプロトコル、ここではHTTP/2を使用するかどうかのネゴシエーションを行います。TLSの通信が前提ですので、httpsの場合に使用できます。

図4 ALPNの流れ



二つめのアップグレードする手順は、最初にHTTP/1.1で通信を開始した後に、そのコネクションをHTTP/2の通信にアップグレードする手順です。httpsの場合はALPNを使用するので、アップグレードはhttpの場合のみ使用できます。

三つめのダイレクトで開始する手順とは、サーバがHTTP/2に対応していることがわかっている場合のみHTTP/2で通信を開始できます。具体的には定義されていませんが、一度接続したことがある等のケースがあります。

これらの方法でHTTP/2利用のネゴシエーションを行います。

■コネクションの再利用

前述の通りHTTP/2では、極力既存のTCPコネクションを使い回してHTTPリクエストを送信します。仕様で既存のTCPコネクションを再利用できる条件が定義されており、httpの場合と、httpsの場合でその条件が分かれています。

httpで接続を行っている場合は、ドメイン名のIPアドレスが同じであればTCPコネクションを再利用できます。例えば、a.example.comとb.example.comが同じIPアドレスであれば、両者との接続はコネクションを再利用できます。

httpsで接続を行っている場合は、http同様にドメイン名のIPアドレスが同じことに加え、証明書が有効なことが条件になります。例えば、a.example.comとb.example.comの証明書が＊.example.comといったワイルドカード証明書であれば、両方のドメインで有効な証明書ですので、両者との接続はコネクションを再利用できます。

■ヘッダ圧縮

HTTPリクエストはHTTPリクエストヘッダ、HTTPレスポンスはHTTPレスポンスヘッダを持ちます。HTTPリクエストヘッダには、User-AgentヘッダやAccept-Encodingといったクライアント側の情報などが格納されています。以下はHTTP/1.1でのサンプルです。

```
GET / HTTP/1.1
Host: example.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/63.0.3239.84 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: ja,en-US;q=0.9,en;q=0.8
```

いくつかの部分は変わることはありませんが、HTTP/1.1ではリクエストの都度すべてのHTTPリクエストヘッダを送信していました。これは冗長なため、ヘッダを圧縮することが考えられていました。

SPDY/3では、ヘッダ領域をDeflate圧縮していましたが、その方法には脆弱性があったため、実際にはヘッダ圧縮機能は無効化されていました。この脆弱性を回避しつつヘッダを圧縮するために、新しくHPACKというヘッダ圧縮方式を策定し、HTTP/2ではHPACKを使用しています。HPACKはHTTP/2とは別にRFC7541で標準化されています。

HPACKでは主に二つの仕組みを併用してヘッダを圧縮します。

1. ハフマン符号を使用する
2. インデックステーブルを使用する

一つめはハフマン符号を用いる方法です。HTTPヘッダで使用される文字の出現頻度には偏りがあることがわかっています。それを利用し出現頻度の高い文字（a, c, e 等）は短いbit数で、出現頻度の低い文字は（\,] 等）は長いbit数で表現することで文字列トータルのデータ量を削減しています。各文字とbit列の対応表はRFC7541で定義されており、エンコードとデコードはその対応表を使います。

二つめはインデックステーブルを使用する方法です。インデックステーブルには、ヘッダ名とヘッダ値が格納されており、表現したいヘッダがすでにインデックステーブルにある場合は、インデックス番号を指定するだけでそのヘッダを表現できるようになります。例えば、「accept-encoding : gzip, deflate」というヘッダは、インデックステーブルにすでに存在しており、インデックス16番と指し

示すだけでこのヘッダを送ったことと同じことになります。

またインデックステーブルには2種類あり、RFC7541で事前に定義されている静的テーブルと、送信したヘッダを動的に追加していく動的テーブルが存在します。テーブルに無いヘッダは、ハフマン符号などを使用して表現し送信します。その時、同時に動的テーブルに追加しておけば、次から同じヘッダを送信する際はインデックス番号を指定するだけでそのヘッダを表現できるようになります。

このように、HTTP/2では二つの方式を組み合わせてヘッダを圧縮しますが、どれを使用するかは実装依存になっており、ハフマン符号すら使わずにヘッダを送信することも可能です。そうして作られたヘッダがHEADERSフレームに格納され、相手に送信されることになります。

■優先度

Webページを表示するのに優先度の高いリソースとそうでもないリソースがあります。例えばCSSはページのレンダリング開始に必要ですが、画像などはレンダリング開始時点ではそれほど優先度は高くありません。

HTTP/2ではHTTPリクエストを多重化して送信しますが、クライアントはHTTPリクエストに優先度を指定できます。HEADERSフレームで優先度を指定するか、もしくはあとからPRIORITYフレームで優先度を変更できます。

優先度はストリームごとに指定できます。指定する際はDependencyとWeightというパラメータを用いて優先度を伝えます。Dependencyは処理する順番を指定できます。例えば、DependencyでCSSのリクエストを処理してから、画像のリクエストを処理するようにサーバに指示できます。Weightは優先度の比を指定できます。CSSファイルAとCSSファイルBを1:2の割合で処理するようにサーバに指示できます。

このようにして、ブラウザは自身の処理して欲しい順番をサーバに伝えることができます。

■サーバプッシュ

HTTP/1.1は、HTTPリクエストが受信されてからHTTPレスポンスが送信されます。HTTP/2ではサーバプッシュという機能があり、HTTPリクエストがなくてもHTTPレスポンスを送信することができます。サーバ側は後々クライアントが必要になるであろうコンテンツを先んじて送信できるようになります。これによってクライアントは待ち時間が少なくなります。

サーバプッシュは、サーバがHTTPリクエストを受け取ってHTTPレスポンスを返す間にPUSH_PROMISEフレームを送信することで開始できます。PUSH_PROMISEはサーバプッシュに使用するストリームの予約と、プッシュするHTTPレスポンスがそもそもどういうHTTPリクエストを想定してのHTTPレスポンスなのかが記述されています。どのURLへのHTTPリクエストに対するHTTPレスポンスなのかというのが重要な情報です。その後、予約したストリームでHTTPレスポンスを送信します。

クライアントはプッシュされたリソースをキャッシュし、HTTPリクエストを送信する際に確認します。もしキャッシュに必要なデータがあれば、そこから使用するような動作をします。

■セキュリティの向上

HTTP/2ではセキュリティの向上も一つの目的になっていました。特にTLSの利用についてHTTP/2の仕様として制限をかけています。既知のセキュリティ上の問題を回避するためです。

HTTP/2においてTLSを使用する場合は以下を守る必要があります。

・TLS1.2以上を使用する
・SNI（Server Name Indication）をサポートする
・仕様で指定される暗号スイートを使用する
・TLSの圧縮機能を無効にする
・TLSの再ネゴシエーションを使用しない

これらが守られない場合は、HTTP/2レイヤでコネクションエラーとして接続は切断されます。

4 発展

■HTTP/2と拡張

HTTP/2は拡張性があり、フレームやSETTINGSフレームでやり取りするパラメータなどを追加できるようになっています。ここでは議論中のものも含め、いくつかの拡張仕様について紹介します。

「The ORIGIN HTTP/2 Frame」という仕様で、ORIGINフレームという新しいフレームを定義しています。HTTP/2はコネクションを再利用するため、オリジン（スキーム・ドメイン名・ポート番号の組）が異なっていても既存のコネクションを使います。しかし実際には、そのコネクションがつながっているサーバではそのHTTPリクエストを処理できない可能性があります。そこで、ORIGINフレームではサーバから明示的にコンテンツを提供できるオリジンを、クライアントに通知できるようにします。

「Secondary Certificate Authentication in HTTP/2」という仕様では、CERTIFICATE_NEEDEDフレーム、USE_CERTIFICATEフレームなどの拡張フレームが定義されています。HTTP/2ではTLSの再ネゴシエーションが禁止されているため、通信の途中で追加のクライアント証明書を要求することはできません。そこで、これらの拡張フレームを用いて通信の途中でも証明書を要求・提供できるようにします。この仕様では、サーバもしくはクライアントのどちらからでも証明書を追加要求できます。

今回はHTTP/2の拡張を紹介しましたが、HTTPステータスコードやHTTPヘッダの追加といったメンテナンスや、HTTP、HTTP/2に関連する仕様の議論も続いています。興味があれば、作業中のドキュメントがGitHub（https://github.com/httpwg/http-extensions）でも公開されているので見てみると面白いと思います。HTTP/2の標準化は大きなマイルストーンでしたが、これからもパフォーマンスとセキュリティの観点での改善は進められていくでしょう。

（グリー株式会社 後藤浩行）