HTTP/2.0の標準化動向と今後の展望

IIJ 大津 繁樹

2013年12月19日

インターネットアーキテクチャ研究会（IA）
情報ネットワーク研究会（IN）

内容

1. HTTP/2.0 標準化動向
   – これまでの歩み、動向
   – SPDYの解説
   – HTTP/2.0 仕様概要
2. HTTP/2.0 今後の展望
   – 相互接続試験から見えること
   – 導入後の課題、将来的な展望
   – セキュリティに関する議論

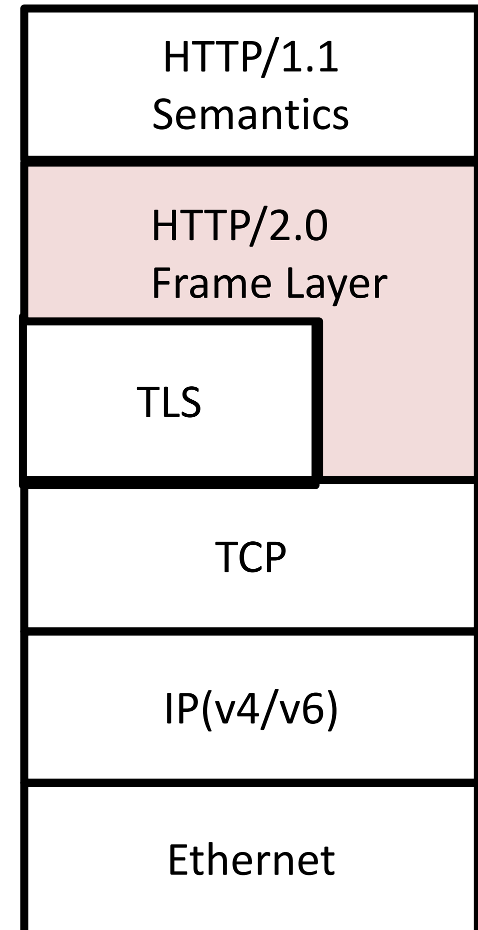
HTTP/2.0標準化動向

HTTP仕様化の歴史

- 1996年 HTTP/1.0 RFC1945 Informal
- 1997年 HTTP/1.1 RFC2067 Standards Track
- 1999年 HTTP/1.1 RFC2616 Standards Track
  - RFC2067を改訂
- 2007年 httpbis WG 発足
  - HTTP/1.1関連仕様の改訂を目指す
- 2012年 HTTP/2.0仕様化開始の議長提案
- 2013年 HTTP/1.1改訂版ドラフト IESGレビュー中
  - Proposed Standardへ

HTTP/2.0とは、

- HTTP/1.1 の策定(1999年)から14年。
- IETF httpbis WGで HTTP/1.1仕様改訂の見込みがたった。
- 新しい仕様を作る動きが開始
- 従来のHTTP/1.1のセマンティクス維持。互換性保持。
- HTTP/2.0でフレーム化、新しいシンタックスを導入。
- SPDYをアイデアにしているが、仕様提案を一般に公募する。



HTTP/2.0 これまでの歩み

年 月	トピック
2012年1月	IETF httpbis WGでHTTP/2.0の仕様検討開始することを決定
2012年11月	3つの候補案から**SPDY仕様をベースにすることを決定** **draft-00**(SPDY/3仕様をそのまま)リリース
2013年1月	**第1回中間会議(東京)** **draft-01**リリース(HTTPからのUpgrade方法を追加)
2013年4月	**draft-02**リリース(フレームフォーマット・タイプの大幅な変更)
2013年5月	**draft-03**リリース(中間会議に向けて修正点の整理・まとめ)
2013年6月	**第2回中間会議(サンフランシスコ)** 新ヘッダ圧縮仕様の採用を決定
2013年7月	**draft-04**リリース(最初の実装仕様)
2013年8月	**第3回中間会議(ハンブルグ)** 最初のHTTP/2.0相互接続試験を実施 **draft-05**リリース(接続試験結果を反映) **draft-06**リリース(次の相互接続試験向け実装仕様)
2013年10月	**第4回中間会議(シアトル)** 2回目の相互接続試験を実施 **draft-07**リリース(中間会議の議論を反映)
2013年11月	**draft-08**リリース(次回相互接続試験の実装ドラフト)
2014年1月	第5回中間会議(チューリッヒ) 開催予定

HTTP-draft-06/2.0 対応相互接続試験実装リスト (2013/10時点）

	名称	実装言語	Client,Server, Intermidate	ニゴシエーション
1	nghttp2	C	S, C, I	NPN, Upgrade, Direct
2	http2-katana	C#	S, C	ALPN, Upgrade
3	node-http2	Node.js	S, C	NPN, direct
4	Mozilla Firefox	C++	C	ALPN, NPN
5	**iij-http2**	**Node.js**	**S, C**	**ALPN, NPN, Upgrade, Direct**
6	Akamai Ghost	C++	I	NPN
7	Chromium	C++	C	ALPN, NPN
8	Twitter	Java	S, C	NPN
9	Wireshark	C	other	NPN, ALPN
10	Ericcson MSP	C	proxy	

(https://github.com/http2/http2-spec/wiki/Implementations より引用）

SPDYについて

ＳＰＤＹ(スピーディ）とは、

- ＨＴＴＰの次期バージョン(**HTTP/2.0**) のベース仕様
- SPDYは、Googleの社内プロジェクトから生まれ、Ｗｅｂページの表示速度を速くするためのプロトコルである。
- 既に２年以上に渡りGoogleの全サービスで利用され、TwitterやFacebook、LINEなど大規模なシステムへSPDYの導入が始まっている。

SPDYの歩み

2009/11	spdy/1 仕様公開
2010/01	TLS NPN拡張仕様のドラフトリリース
2010/02	spdy/2 仕様公開
2010/09	Chrome6安定版リリース。SPDYがデフォルトで有効になる。
2011/01	**Google サービスの90%がSPDY化完了のアナウンス**
2011/05	spdy/3 仕様公開
2011/12	FireFox11 開発版に SPDY実装される
2012/03	**Twitter** SPDY化開始
2012/08	**wordpress.com** が SPDY対応開始
2013/01	**LINE**がSPDYを利用していることを公表 **Facebook** が SPDY対応開始
2013/06	Win8.1 の IE11 (preview)で SPDY対応していることが判明
2013/09	SPDY/3.1 仕様公開

ブラウザのSPDY対応状況

ブラウザ	デスクトップ版	モバイル版
Chrome	**対応** (Ver. 6以上)	**対応** (Ver. 18以上)
FireFox	**対応** (Ver. 13以上)	**対応** (Ver.15以上)
Opera	**対応** (Ver. 12.10以上)	**対応** (Ver. 12.10以上)
Android標準ブラウザ	----	**対応** (Ver. 3以上)
Safari	未対応	未対応
Internet Explore	**対応** (Ver. 11 on WIn8.1以上)	?

(標準でSPDYが有効になったバージョンを記載)

サーバソフトのＳＰＤＹ対応状況

サーバソフト	言語
node-spdy	Node.js
spdylay	C
apache mod_spdy	C
Jetty	Java
nginx(Proxy用途)	C(現状 ver.2のみ)

（他に python, ruby 実装も)

SPDYによる性能向上効果の実測

表: Googleサービスにおけるページ読み込み時間の短縮率(対SSL)

	Google News	Google Sites	Google Drive	Google Maps
中央値	－43%	－27%	－23%	－24%

筆者の単純なベンチマーク結果からは、10%～20%程度の性能向上が観測されたが、SPDYで逆に遅くなる場合も見られた。

データ出典: Making the web faster with SPDY and HTTP/2
http://blog.chromium.org/2013/11/making-web-faster-with-spdy-and-http2.html

SPDYからHTTP/2.0へ

HTTP/2.0の主な技術的な特徴

- クライアント・サーバのTCP接続数を１つに限定
- 3種類・2段階の初期接続方法
- バイナリープロトコル
- 優先度付全２重多重化通信
- フロー制御(コネクション・ストリームレベル）
- サーバプッシュ機能
- HTTPヘッダに特化したデータの圧縮手法

（注：SPDYの特徴も含みます）

主なSPDYとHTTP/2.0の違い

- 明確にHTTP利用を宣言
  - ただし拡張する余地はあり
- TLS以外のハンドシェイク手法の導入（適応議論中）
- フレームヘッダの簡略化
  - 20byte(SPDY)→8byte(HTTP/2.0) 無駄なフィールドを削除
- フレームタイプ、設定情報の見直し
  - ストリーム確立のためのハンドシェイクを廃止、設定値の交換情報を削減
  - サーバプッシュ手法の変更（リクエスト同期から予約型に）
- HTTP/1.1セマンティクスとの整合性を厳密化
  - ヘッダ情報のマッピングを厳密に定義
  - Expect 100 Continue や chunk Encodingの廃止
- 新ヘッダ圧縮手法 HPACKの導入

HTTP/2.0仕様概要項目 (draft-08ベース)

1. 初期ハンドシェイク方法
2. フレーム
3. ストリーム、多重化、フロー制御
4. HTTPマッピング、サーバプッシュ
5. 新ヘッダ圧縮手法 (HPACK)

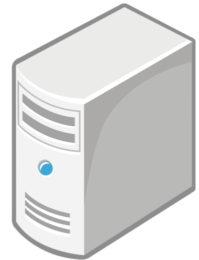
HTTP/2.0初期ニゴシエーション 3種類で2段階（その１）

(1) TLS + ALPN

詳細後述

TLS接続時にALPN拡張フィールドを利用してHTTP/2.0に接続を行う。

(2) HTTP Upgrade



HTTP/1.1の接続後 Upgradeヘッダを使って、HTTP/2.0 に接続をアップグレードする。

別仕様への分離や廃止も議論中

(3) Direct接続

あらかじめサーバがHTTP/2.0対応とわかっている場合、直接第２段階の接続方法を行う。(DNSレコードや HTTPヘッダによるリダイレクト）

ALPN
(Application Layer Protocol Negotiation)

TLSハンドシェイク

1. ClientHello + ALPN拡張

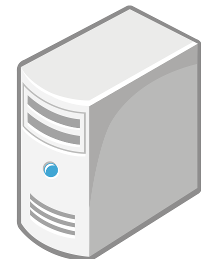
プロトコルリストをサーバに送信
http/2.0,spdy/3.1,http/1.1

2. ServerHello + ALPN拡張

サーバ側でプロトコルを決定し、通知する
http/2.0

3. TLS 証明書・暗号化情報交換

クライアント

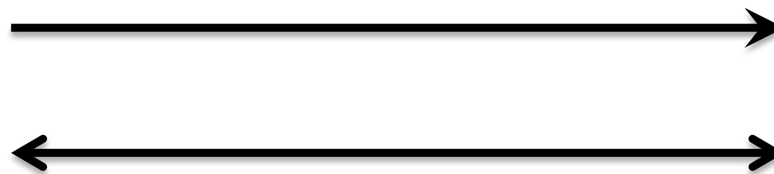


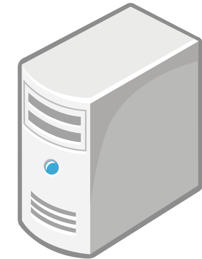
サーバー

HTTP/2.0初期ニゴシエーション
3種類で2段階（その２）

PRI * HTTP/2.0¥r¥n
¥r¥n
SM¥r¥n
¥r¥n

505249202a20485454502f322e300d0a0d0a534d0d0a0d0a





SETTINGS
(初期ウィンドウサイズ、ストリームの最大同時オープン数等の設定情報を含む）

クライアントから謎の24byteのマジックコードをサーバに送り、初期情報(SETTINGSフレームを交換する）

HTTP/2.0フレーム形式

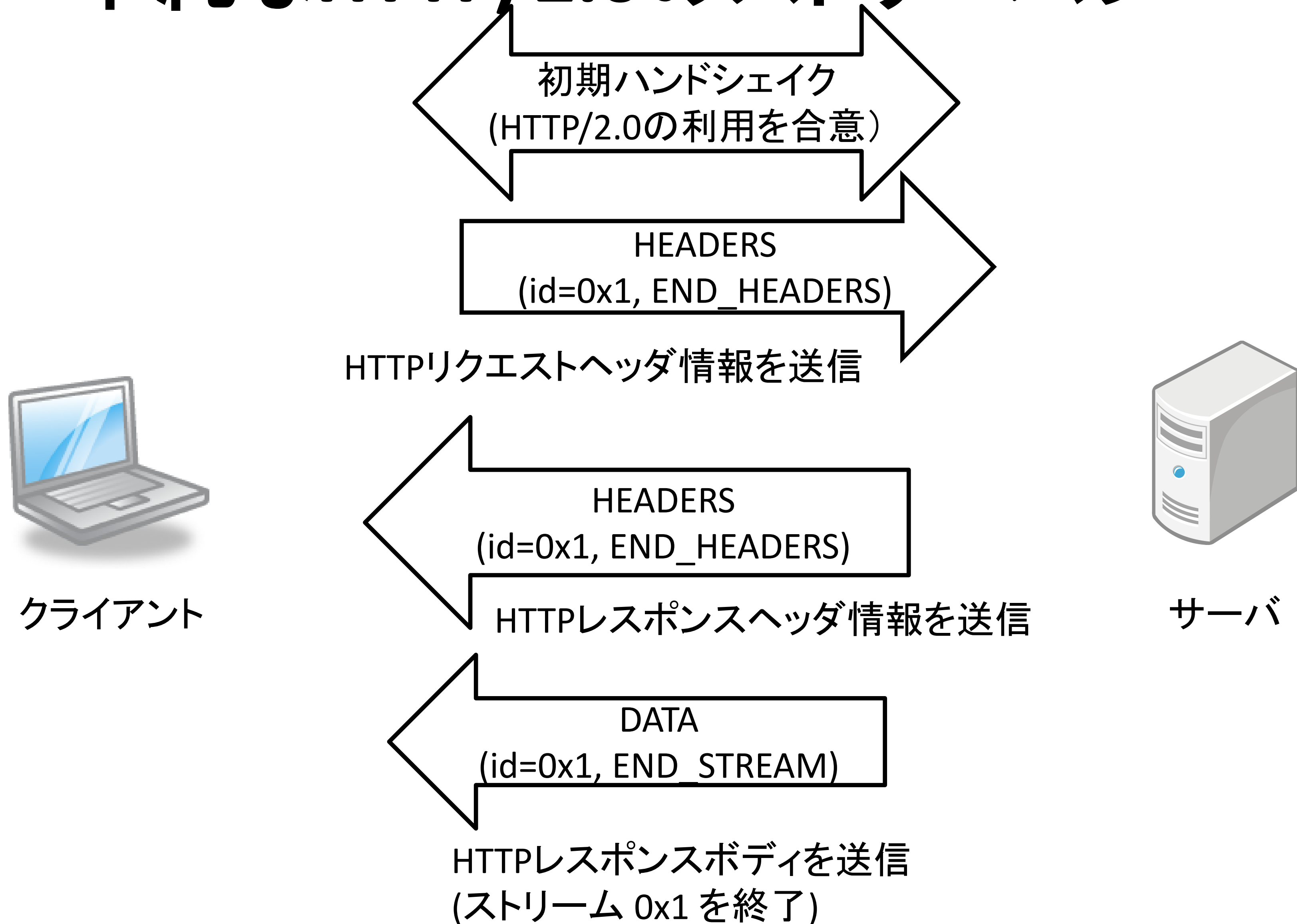
- フレームヘッダ: 8バイト
- フレームペイロード長: 0～最大16,383バイト
- フレームタイプ: 10種類
- ストリームID: 31ビット長
  - 0: コネクション全体
  - 奇数: クライアント発信のストリーム
  - 偶数: サーバ発信のストリーム

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="2">R</td><td colspan="14">Length(14)</td><td colspan="8">Type(8)</td><td colspan="8">Flags(8)</td></tr>
<tr><td>R</td><td colspan="31">Stream Identifier(31)</td></tr>
<tr><td colspan="32">Frame Data</td></tr>
<tr><td colspan="32"></td></tr>
</table>

HTTP/2.0フレームタイプ

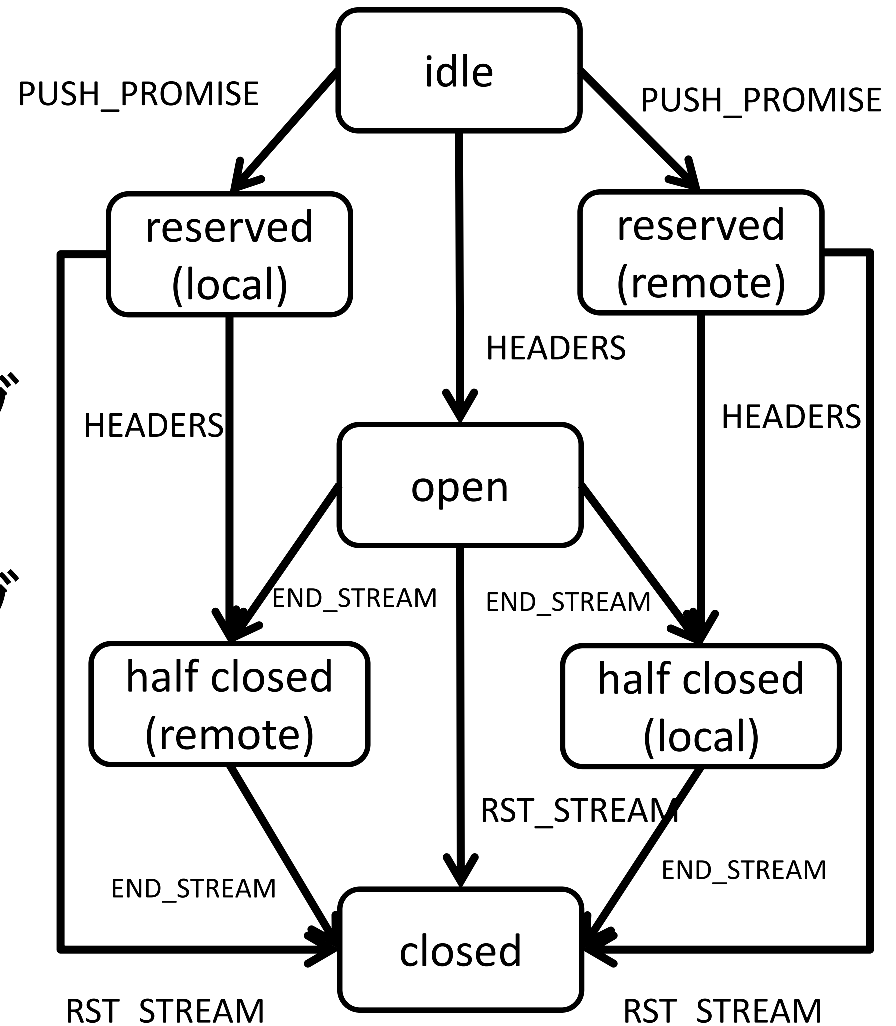
Type	フレーム名	役割
0x0	DATA	リクエスト･レスポンスボディのデータ送受信
0x1	HEADERS	ヘッダ･優先度の送受信、ストリームの開始
0x2	PRIORITY	優先度変更の通知
0x3	RST_STREAM	ストリームのリセット通知
0x4	SETTINGS	設定情報の交換
0x5	PUSH_PROMISE	サーバプッシュ情報の通知
0x6	PING	死活確認
0x7	GOAWAY	コネクションの切断
0x8	欠番	
0x9	WINDOW_UPDATE	フロー制御ウィンドウの通知
0xA	CONTINUATION	大きなヘッダ情報の分割データの送信

単純なHTTP/2.0のストリームフロー



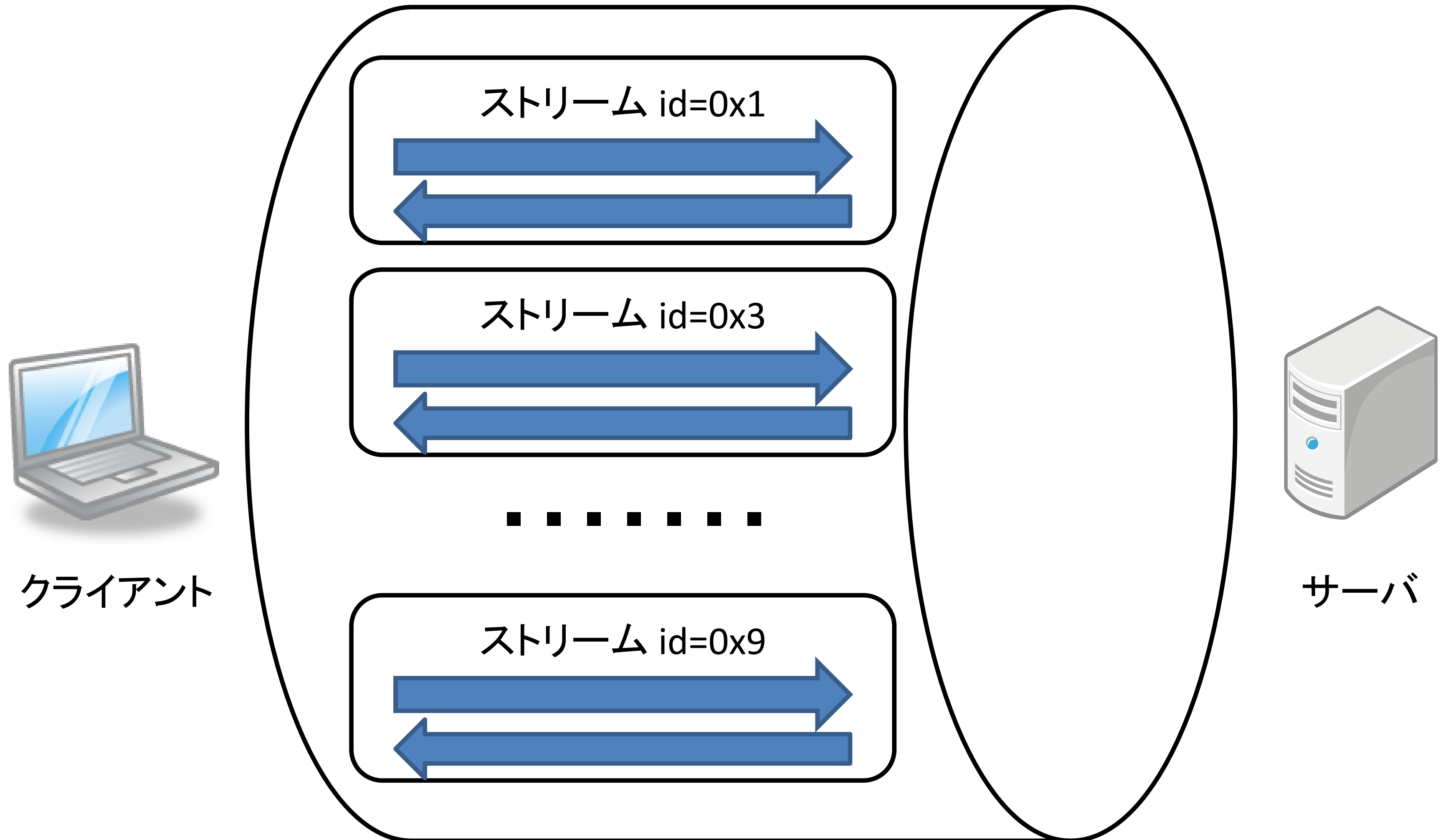
HTTP/2.0 ストリーム状態

- 最初は全て idle 状態
- HEADERを送受信したら open
- 片側が END_STREAMフラグを送ったら half closed
- 双方が END_STREAMフラグを受信したら closed
- PUSH_PROMISEで指定されたストリームは reserved で予約済状態に



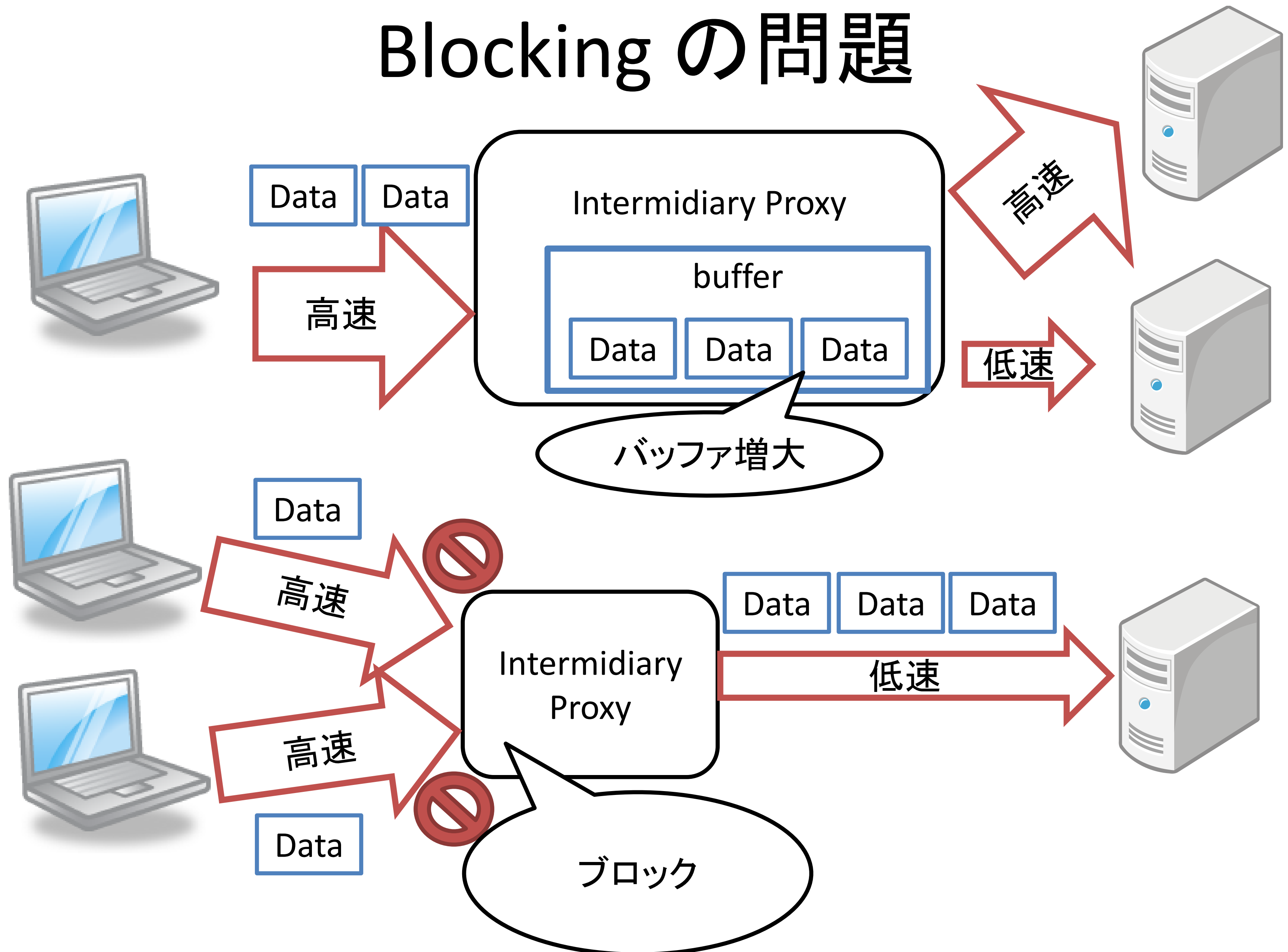
ストリーム状態遷移図

HTTP/2.0 Stream Multiplex



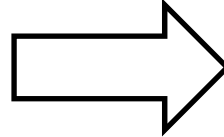
1つのTCP接続中でストリームの多重化を実現

Buffer BloatとHOL(Head of Line) Blockingの問題



HTTP/2.0 フロー制御

SETTINGフレーム
（**初期ウィンドウ値**）



コネクション・ストリーム毎のウィンドウサイズを規定
(default 64KB)

コネクション/ストリーム

HEADERS

Data フレーム

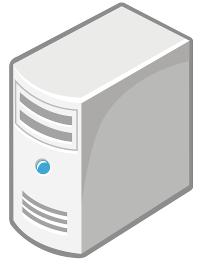
Data フレーム

Data フレーム

ウィンドウサイズ分だけ最大送れる

WINDOW_UPDATEフレーム
（Delta-Window値）

ウィンドウサイズ更新



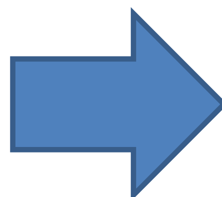
HTTPマッピング

リクエストヘッダ

GET / HTTP/1.1

Host: example.com

User-Agent: foo



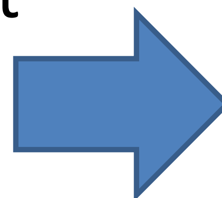
HTTP/2.0でのヘッダ情報

key	value
:method	GET
:scheme	https
:authority	example.org
:path	/
user-agent	foo

レスポンスヘッダ

HTTP/1.1 204 No Content

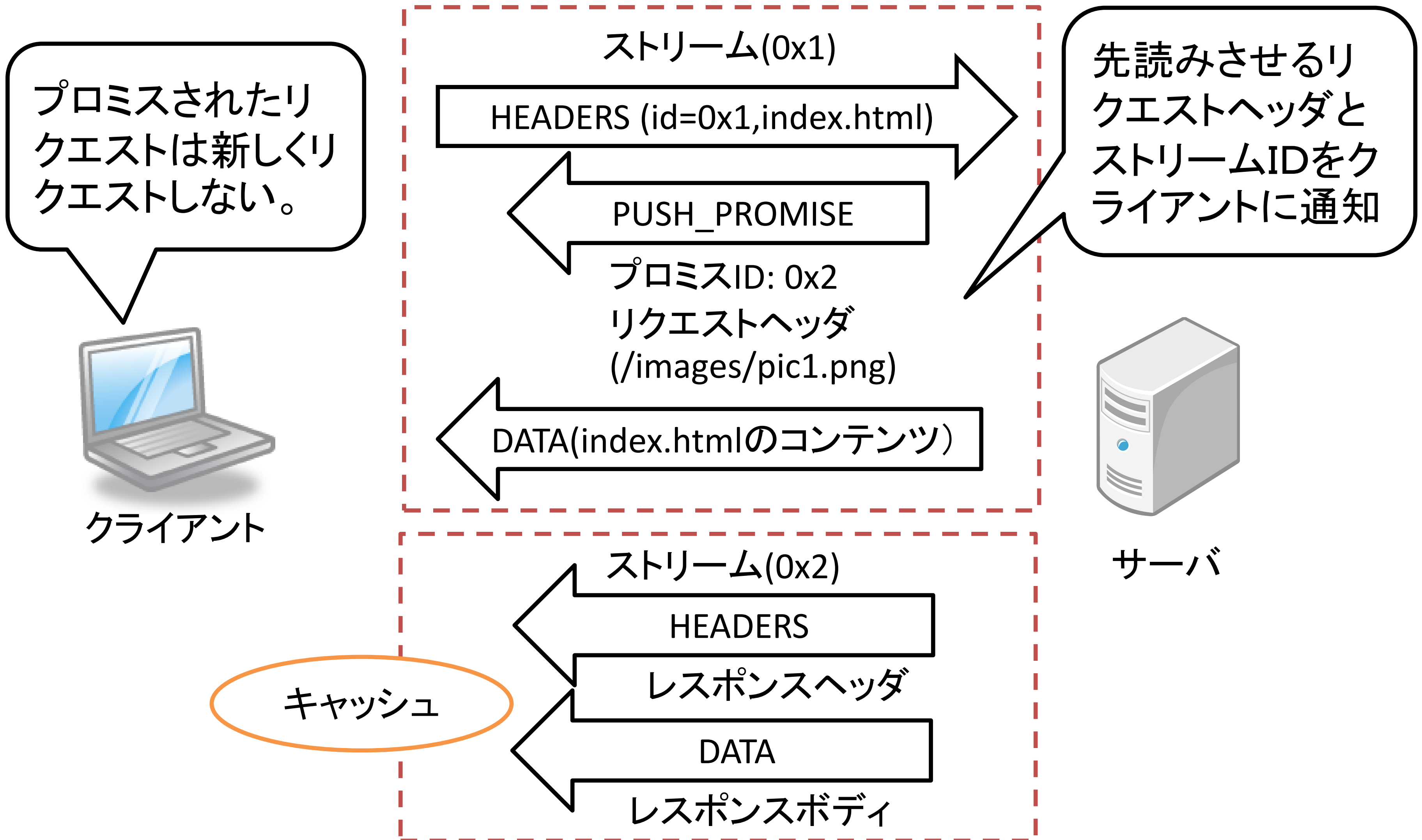
Content-Length: 0



key	value
:status	204
content-length	0

HTTP/1.1のヘッダをHTTP/2.0用にマッピング。
HPACK(後述)で符号化し、HEADERのペイロードで送受信。

サーバプッシュ機能



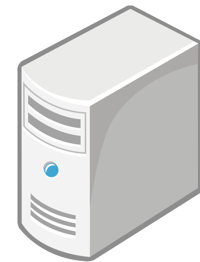
サーバがコンテンツをプッシュしてクライアントにキャッシュ

HPACK:新しいヘッダ圧縮仕様

GET / HTTP/1.1
host: www.example.com

1. 2番の:method GETを追加
2. 7番の:scheme http を追加
3. 6番の :path / を追加
4. 4番の : authority に www.example.com をハフマン符号化して追加

(ヘッダの差分情報を符号化してやり取りする)



平均20～30%データ量を削減
CRIME脆弱性対応

送信前ヘッダテーブル

1. :authority,
2. :method, GET
3. :method, POST
4. :path, /
5. :path, /index.html
6. :scheme, http

....

0x82
0x87
0x86
0x04 0x8b 0 xdb 0x6d 0x88
0x3e 0x68 0xd1 0xcb 0x12
0x25 0xba 0x7f
（実際に送信するヘッダ情報）

受信後ヘッダテーブル

1. :authority, www.example.com
2. :path, /
3. :scheme, http
4. :method, GET

....

HTTP/2.0 今後の展望

今後の展望
相互接続試験から見えたこと

- フレーム・ストリーム処理は、SPDY導入による技術的ノウハウがあるため各実装とも安定している。
- ヘッダ圧縮(HPACK)技術の実装はまだ手探り状態。クライアント、サーバ間で状態の不一致が発生した場合など問題の切り分け、特定が困難。
- 仕様準拠のためのテストフレームワークの議論はこれから。

今後の展望

- ユーザ視点では、一見なにも変わらない。
  - 2011年1月よりGoogleサービスは全面SPDY化
  - Chrome ユーザは、その違いに気づいただろうか？
- 確かに昔より表示が速く、スムーズになったような感じはある。いろんな最適化の複合要因であろう。
- 標準化で多種ブラウザーが対応し、広く利用できる環境が整う。

今後の展望

実は単純導入だけでは難しい

- ブラウザが決めるリソース取得の優先度にどう対応する？
- プロキシ構成などで異なるトラフィックをフロー制御して最適化を図れるのか？
- 細かいチューニング手法は未知数。運用してサイトにあった構成を。日々の測定が大事。

今後の展望

- 大規模システムでは、＊おそらく＊変わるのではないか？（手元に定量的なデータがない）
- SPDYでは、TLS利用によるオーバヘッドより性能向上効果が上回ったとの話も聞く。
- サーバリソース、ネットワークリソースを効率的に利用できるので大規模サービスになればなるほどその効果を享受できるのではないか？

今後の展望

- ブラウザー以外の利用への展開
  - {key,value} + data をやり取りする独自アプリ(LINE など）

- 原理的には接続後サーバ側からリクエストも可能（双方向化)

- いろんな付加情報をあらかじめ送りつける (DNS, Proxy, NTP 等々)

今後の展望

- Internet Giants を中心にHTTP/2.0の導入が進む。（先行者利益を享受）
- 小規模サイト、一般ユーザの移行メリットは少ないだろう。HTTP/1.1はなくならない。（二極化）

- ブラウザ側の対応も進み、ますますHTTP/2.0に最適化されるであろう。
- 特にモバイル環境での性能改善に期待がかかる。

今後の展望
セキュリティに対する取り組み

- スノーデン事件の余波
  - NSAが盗聴、改竄などを大規模・広範囲に実施していることが明るみに。

> - PRISM： 情報収集・通信監視
> - Quantum：バックボーンMITM
> - Fox Acid： 脆弱性攻撃
> - MUSCULAR：データセンター内通信傍受
>
> など・・・・

IETFで「インターネットをより堅牢に」と機運が高まる。

今後の展望
IETF httpbis WG議長から提案

HTTP/2.0を**https(暗号化必須)**に限定

今後の展望

HTTP/2.0を**https(暗号化必須)**に限定？

賛成

- 広範囲なネット盗聴・監視に対抗できる
- 既存のネット環境にすばやく導入可能
- TLSのALPN拡張と組み合わせて初期接続時間の短縮が図れる
- Chrome/Firefoxは、平文でのHTTP/2.0 通信は実装しないと明言

反対

- Proxy機能が損なわれる
- 人々にHTTP/2.0が絶対に安全であると勘違いさせる
- そもそもHTTP/2.0の設計と暗号化は独立した別問題
- ユーザの同意なしに全て暗号化するのは問題
- IoT(Internet of Things)に暗号化は不要な機能
- 平文通信が必要なら仕様に関係なく使い始める

技術的な優劣での判断がしづらく流動的

御清聴ありがとうございました。