Java EE 8 アプリケーション設計ガイド

Servlet 4.0 編

日本アイ・ビー・エム　システムズ・エンジニアリング株式会社

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2019年3月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- ＩＢＭ、ＩＢＭロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれＩＢＭまたは各社の商標である場合があります。現時点でのＩＢＭの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, Windows および Windowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

目次

本文書の位置づけ

- この文書は、Java EE 8 アプリケーション設計ガイド シリーズにおいて、「Servlet 4.0」を対象に主に仕様解説の観点で記述したものです。

- 今後、この文書の記載内容をベースに、WAS Libertyでの実装例や役立つTipsなどをトピック毎にまとめ、developerworksにて随時リリースする予定です。
  - http://www.ibm.com/developerworks/jp/websphere/category/was/

1. はじめに

1.1. Java EE 8について

1.1. Java EE 8について

- Java EE 8 新機能
  - Security API の追加
    - Java EEアプリケーション内で統一した方法でセキュリティの機能を実装可能
  - JSON-Bindingの追加
    - Java のクラスおよびインスタンスと、認められた表記法に従った JSON ドキュメントとの間でデフォルト・マッピングを提供
  - CDI(Contexts and Dependency Injection) 2.0
    - 非同期起動への対応
  - Bean Validation 2.0
    - データの整合性を保つための制約機能のバリエーションが豊富に
  - **Servlet 4.0**
    - **HTTP/2に対応**

- 本開発ガイドの目標

Servlet 3.1(JavaEE 7) ⇨ Servlet 4.0(Java EE 8)

・新機能は何か
・どのように実装していくのか

2. Servlet 4.0

2-1. Servlet 4.0 概要

- Servlet 4.0の新機能(ピックアップ)
  - **HTTP/2対応**
    - HTTP/1.xに比べ処理速度が向上
    - プログラムの変更なしに利用可能
  - **サーバープッシュ**
    - １リクエストに対して複数のレスポンスを返すこと が可能に
  - **マッピングの実行時検出**
    - どのようにServletがリクエストされたかを調査可能に
  - **HTTP Trailer**
    - ヘッダーと同じ構造でボディの末尾に情報を加えることが可能

- その他新機能
  - GenericFilterとHttpFilter
    - フィルター記述の単純化
  - ServletContextの新メソッド
    - addJspFile()
      - 指定したJSPファイルのサーブレットを サーブレット・コンテキストへ追加
    - getSessionTimeout()
    - setSessionTimeout()
      - セッション・タイムアウトのアクセスを提供可能
    - getRequestCharacterEncoding()
    - setRequestCharacterEncoding()
    - getResponseCharacterEncoding()
    - setResponseCharacterEncoding()
      - デフォルトのリクエスト、レスポンスの文字エンコーディングを 変更可能

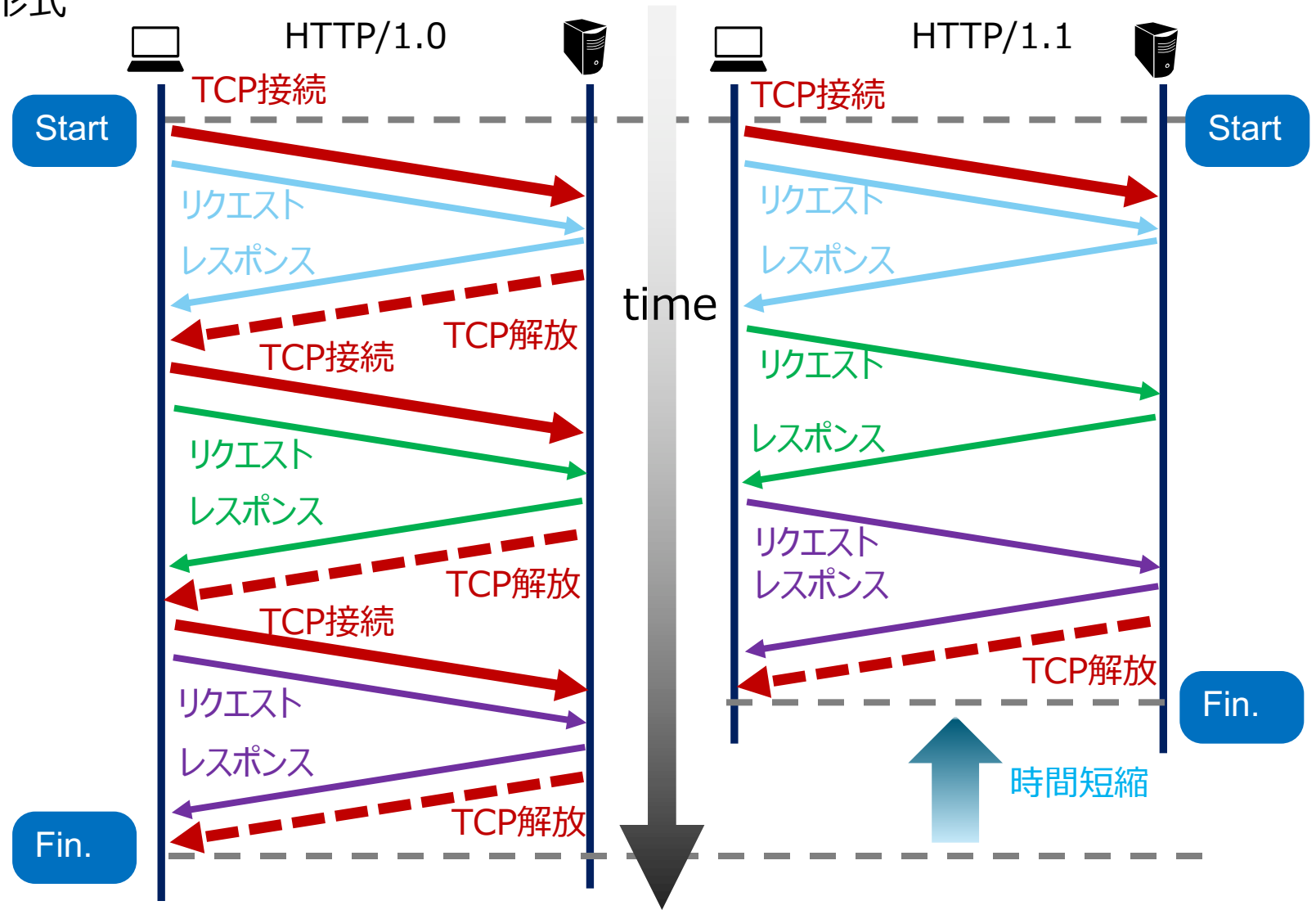
2-2. HTTP/2について

- HTTP/1.0とHTTP/1.1の差異と課題
  - –リクエスト処理形式

HTTP/1.0

- 1回のTCP接続につき1リクエストのみ処理可能
- TCPの再接続によるタイムロス

処理遅延の原因



HTTP/1.1

改善点

- 一回のTCP接続で複数のリクエストを処理可能
- TCPの再接続による時間ロスの解消

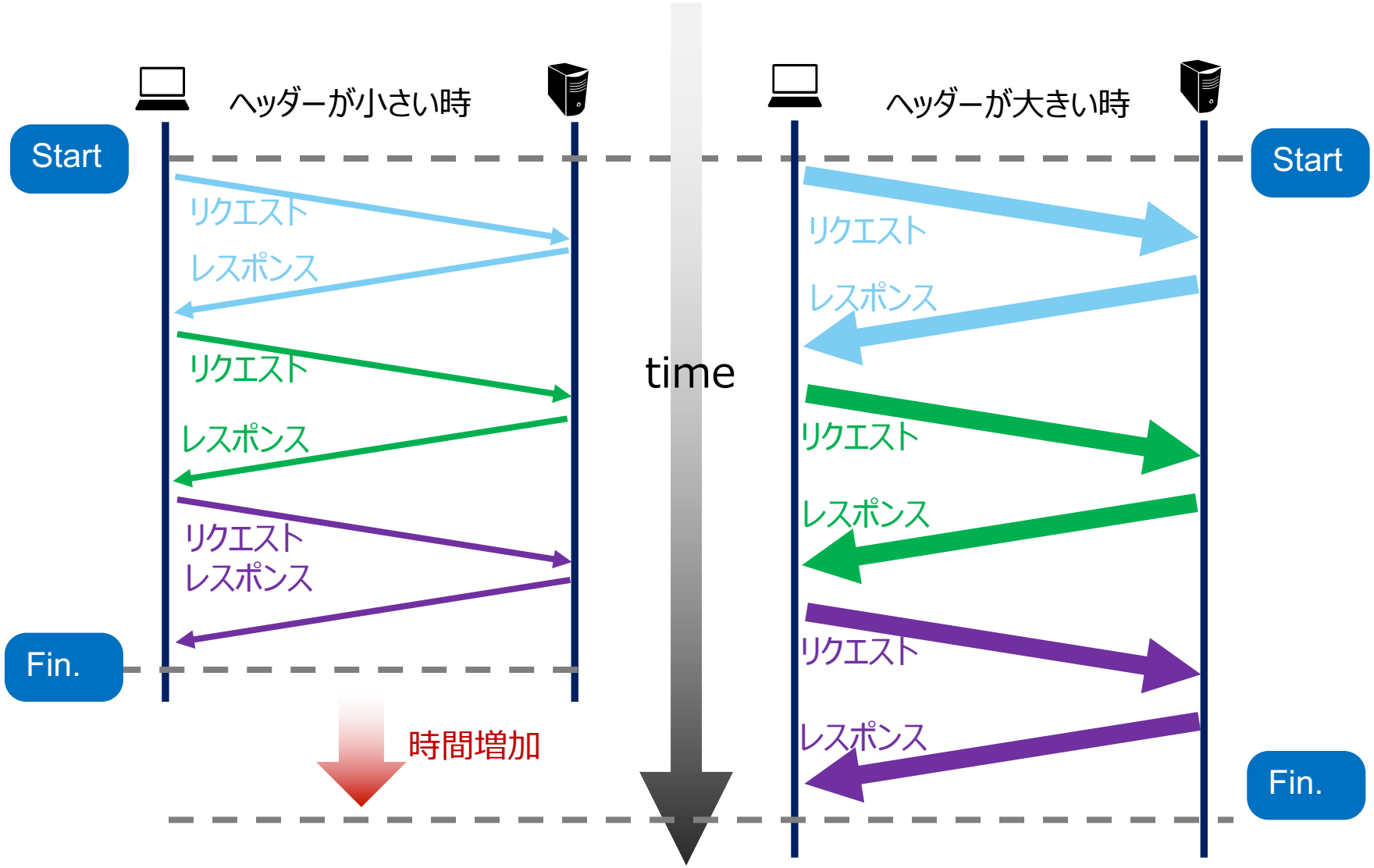
課題

- 前のリクエストが完了を待ってから次のリクエストを処理
- リクエストの優先順位の変更不可
- TCP接続を複数張ることができるが、サーバーに負担

改善点はあるが、課題も残る

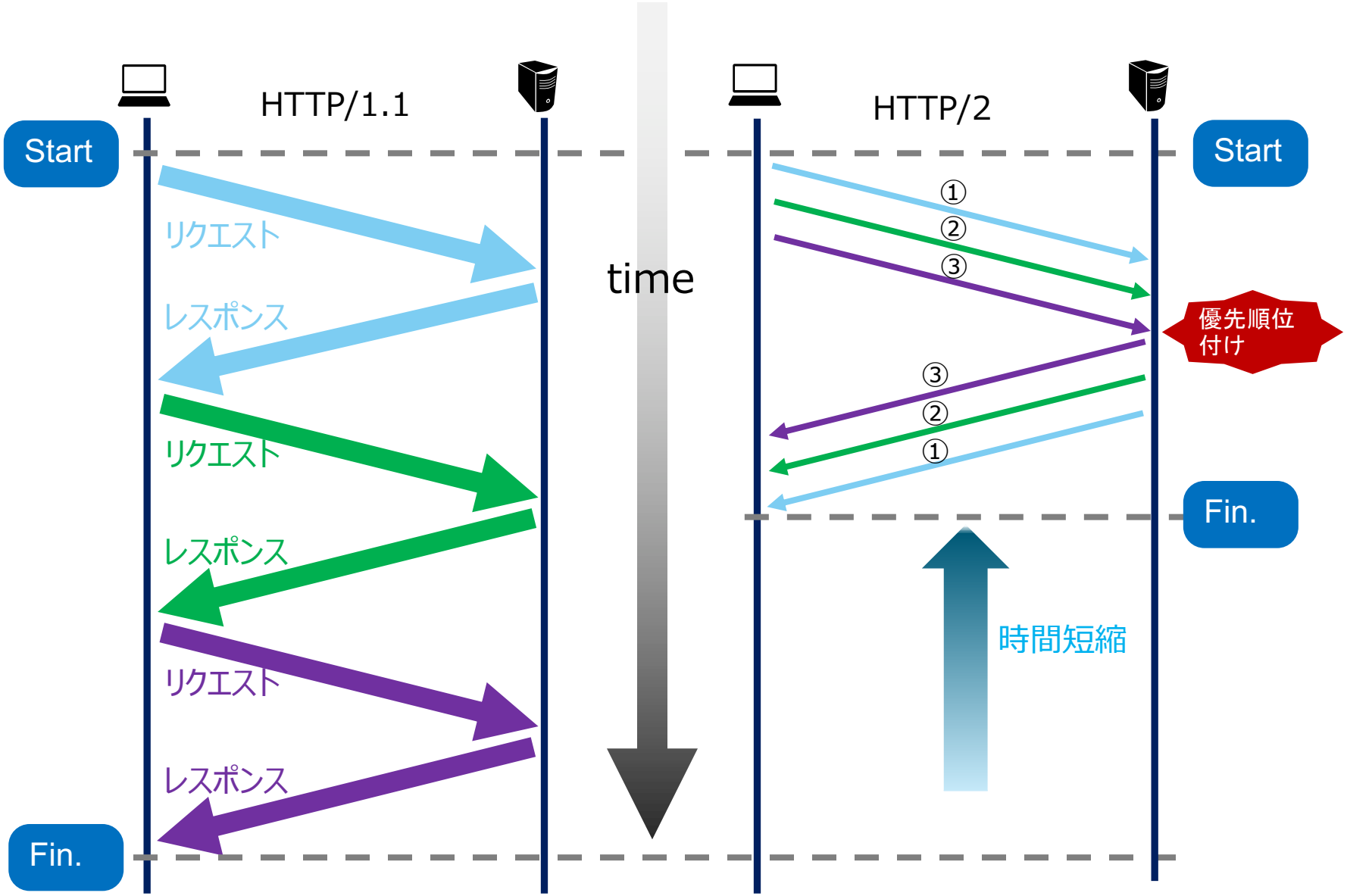
2-2. HTTP/2について

- HTTP/1.1の課題
  - –プロトコルオーバーヘッド
    - ヘッダーのデータ量が大きいと、毎回のリクエストの処理時間が増加
    - 昨今のWEBでは様々な内容がヘッダーに付与されることが多くなったため、ヘッダーのデータ量も増大傾向



2-2. HTTP/2について

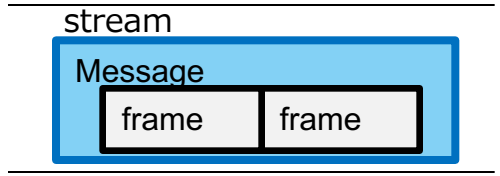
- HTTP/2の新機能
  - 通信方式の変更
    - 複数リクエストの並列処理が可能に
  - リクエストの優先順位付け
    - リクエストの優先順位づけにより、より効率的なリクエスト群の処理が可能に
  - ヘッダー圧縮機能
    - ヘッダーに圧縮機能がつき、各リクエストの軽量化が可能に



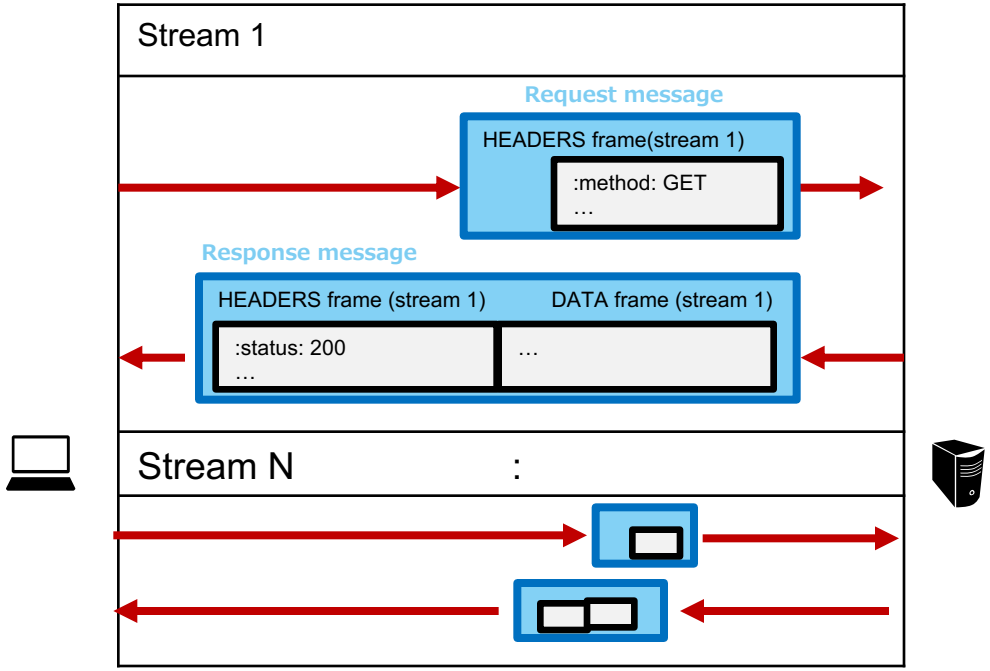
2-2. HTTP/2について

- HTTP/2の構成
  - HTTP/2 通信は3つの単位で構成される
    - フレーム(frame)
      - HTTP/2 での通信の最小単位。それぞれヘッダーフレームが含まれ、フレームが属するストリームを識別
      - HEADERSフレームや、DATAフレーム、後述するPUSH_PROMISEフレームなどタイプが複数存在
    - メッセージ(message)
      - 論理的なリクエストまたはレスポンスメッセージにマッピングする、フレームのシーケンス
    - ストリーム(stream)
      - 確立した接続におけるバイト単位のデータの双方向フロー。1つ以上のメッセージを伝送
      - 1つのTCP接続内に任意の数の双方向ストリームを伝送可能
      - メッセージ伝送に使用する識別子、優先順位情報を含む

- HTTP/2通信の単位
  - フレーム<メッセージ <ストリーム<TCP接続



- 1TCP接続中のHTTP/2通信のイメージ図



2-2. HTTP/2について

- ヘッダー圧縮、リクエスト優先順位づけについて
  - HPACK圧縮形式
    - HPACK というヘッダー圧縮フォーマットを使用することで、一度送信したヘッダーを再送信せずに、必要な差分ヘッダーのみ送信することが可能に

  - リクエスト優先順位づけ
    - 各ストリームに重みと依存関係を割り当てる
    - 依存関係と重みを組み合わせて優先順位ツリーを構築し、優先的に受信するレスポンスを指定できる。
    - この情報を使用して、CPU、メモリ、その他リソースの割り当て、帯域幅の割り当てを制御することでクライアントへの配信を最適化

2-3. サーバープッシュ

- HTTP/2の新機能
  - **サーバープッシュ**
    - サーバーはクライアントからのリクエストがあって初めてレスポンスを返すことが出来たが、 HTTP/2では、サーバーはクライアントからのリクエストが無くてもレスポンスを返すことが出来る。

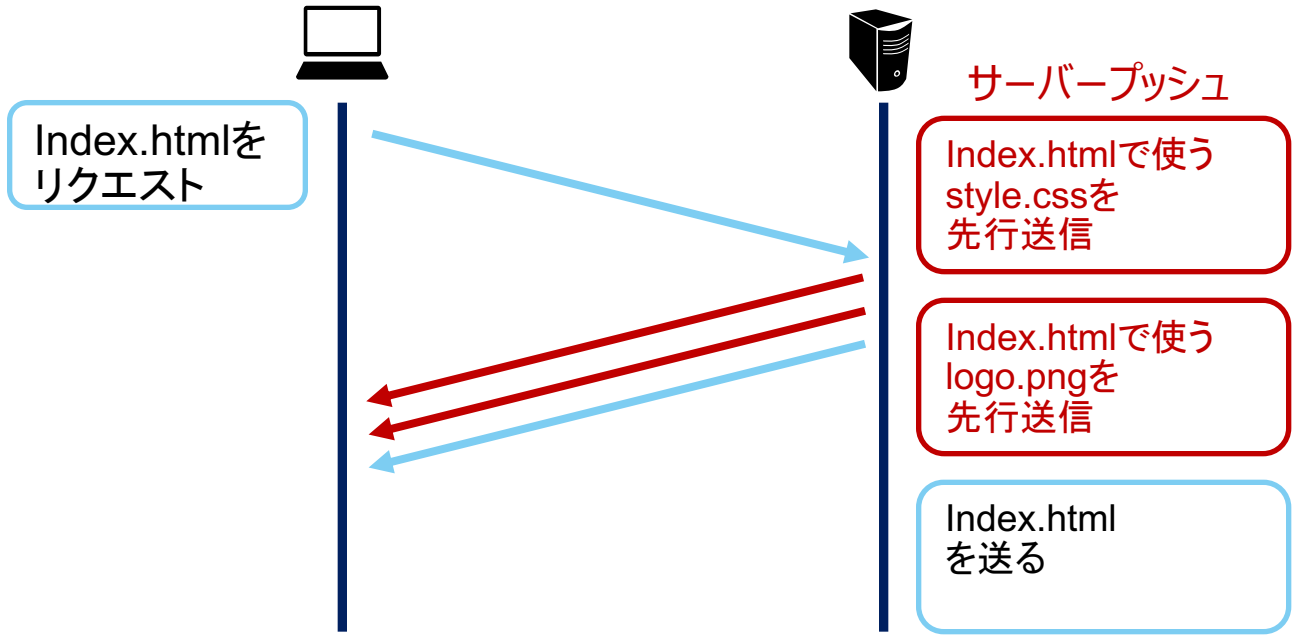
Index.html

```
<html>

<head>
・・・
<link rel="stylesheet" href="css/style.css">

</head>
・・・

<img class="logo" src="images/logo.jpg"  />
・・・
```



2-3. サーバープッシュ

- サーバープッシュ
  - サーバー側でプッシュするファイルを指定するよう記述
  - リクエストに対しPUSH_PROMISEフレームの送信

Index.html

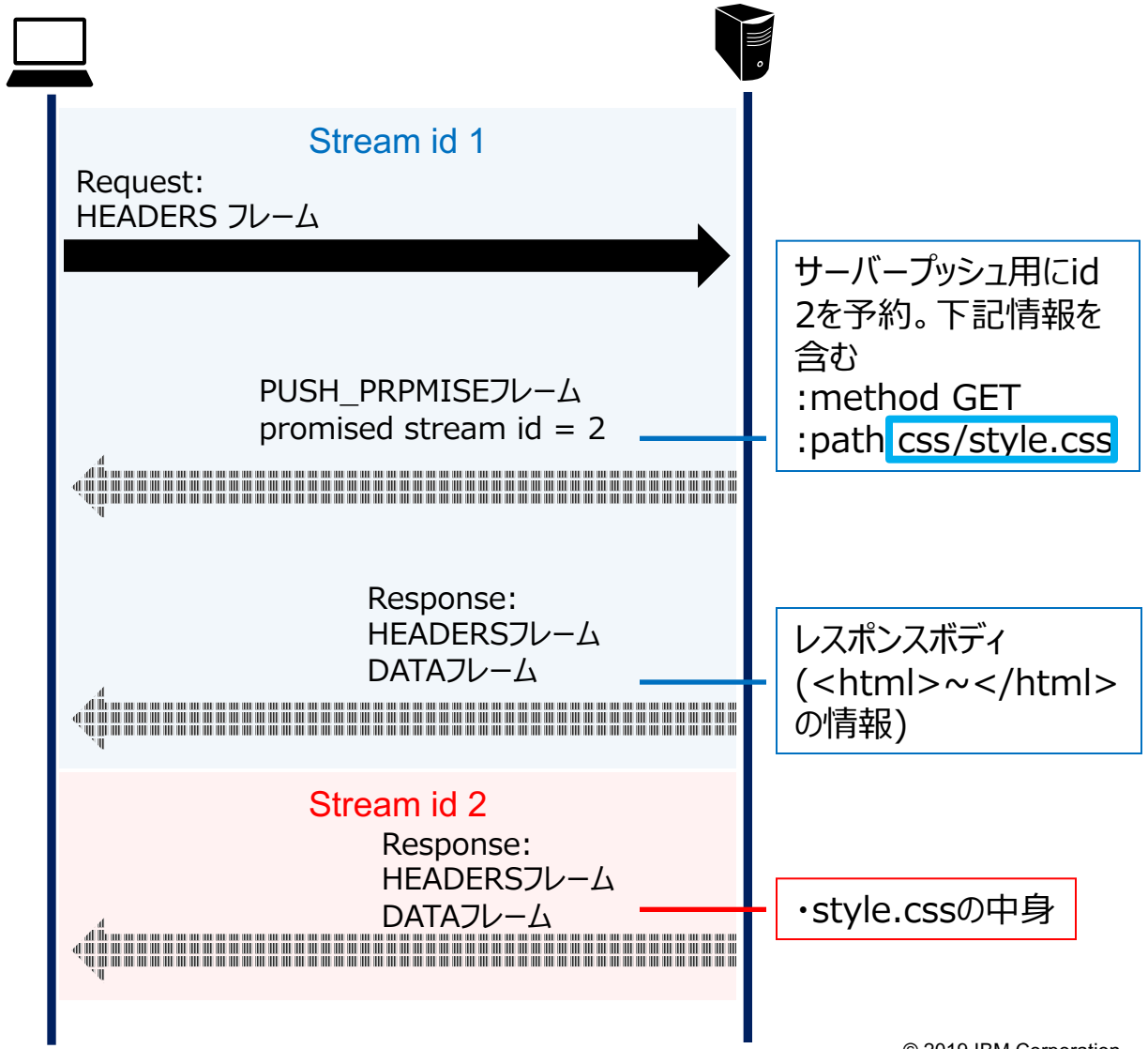
```
<html>

<head>
・・・
<link rel="stylesheet" href="css/style.css">

</head>
・・・
```

サーバー

Index.htmlからのリクエストに対し..
“css/style.css”をPush



2-4. マッピングの実行時検出

- 現在のServletがどのように呼び出されたかを実行時に検出できる
- 下記4つのパラメータを返すことが可能
  - MappingValue
    - リクエストの原因となったURLパスの一部分を返す
  - MappingPattern
    - このマッピングのurl-patternを返す
  - MappingMatch
    - 一致するタイプを返す
  - ServletName
    - Servlet名を返す

結果

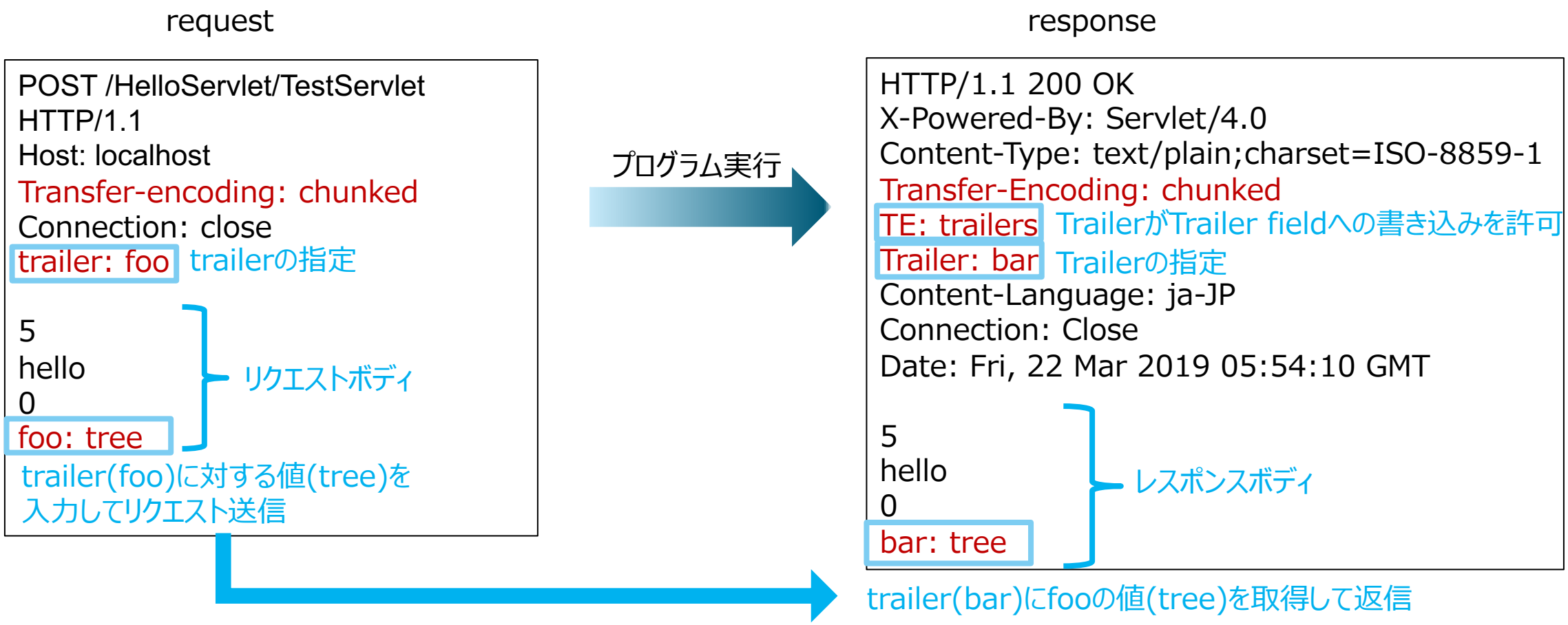
結果例)

入力　URI Path	MatchingValue	MappingPattern	MappingMatch	ServletName
“”	“”	“”	CONTEXT_ROOT	MyServlet
“/index.html”	“”	/	DEFAULT	MyServlet
“/MyServlet”	MyServlet	/MyServlet	EXACT	MyServlet
“/foo.extention”	foo	*.extention	EXTENTION	MyServlet
“/path/foo”	foo	/path/*	PATH	MyServlet

・Servletがどのように呼ばれたかが分かる

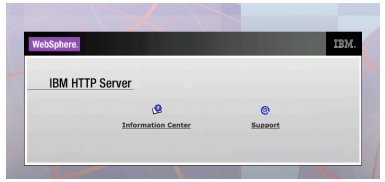
2-5. HTTP Trailer

- ヘッダーと同じ構造でボディの末尾に情報を加えたり、編集することが可能
- HTTP/1.1でも利用可能



2-6. Webサーバー(IHS)とLibertyの連携について

- クライアント、Liberty、IHS間でSSL通信が可能
  - 連携方法
    - Liberty, IHSにおけるSSL鍵データベースの作成と、証明書の鍵データベースへの追加
    - Httpd.confや、plugin-cfg.xmlを編集し、証明書のパスやリスニングポートなどを追記
  - アクセス
    - https://<guest IP>へアクセス



- HTTP/2 はSSL通信が必須で、Webサーバーとブラウザがともに HTTP/2 に対応している必要
- Libertyは18.0.0.2以降でHTTP/2 に対応しているが、IHSはHTTP/2 に2019年3月時点で未対応
  - IHS経由のリクエストで、サーバープッシュ機能を用いることはできない
- サーバープッシュ以外の新機能(P.8)は使用可能
  - マッピング実行時検出
  - HTTP Trailer
  - GenericFilterとHttpFilter
  - ServletContextの新メソッド

3. Servlet 4.0を用いたプログラム

3-1. プログラム前提

- プログラム実装環境
  - Eclipse Java EE IDE for Web Developers
  - Java version 1.8.0_201
  - Mac OS X
  - WebSphere Application Server 19.0.0.1
  - WAS Liberty with Java EE 8 Full Platform
  - ブラウザ：chrome
  - chromeの拡張機能としてHTTP/2 and SPDY indicator 1.0.0 導入
    - HTTP/2のインディケーター

- HTTP/2利用について
  - javaee-8.0, servlet-4.0 featureを有効にしていればSSL通信する際にデフォルトで利用可能
  - 利用にあたって従来からプログラムを変更する必要はなし

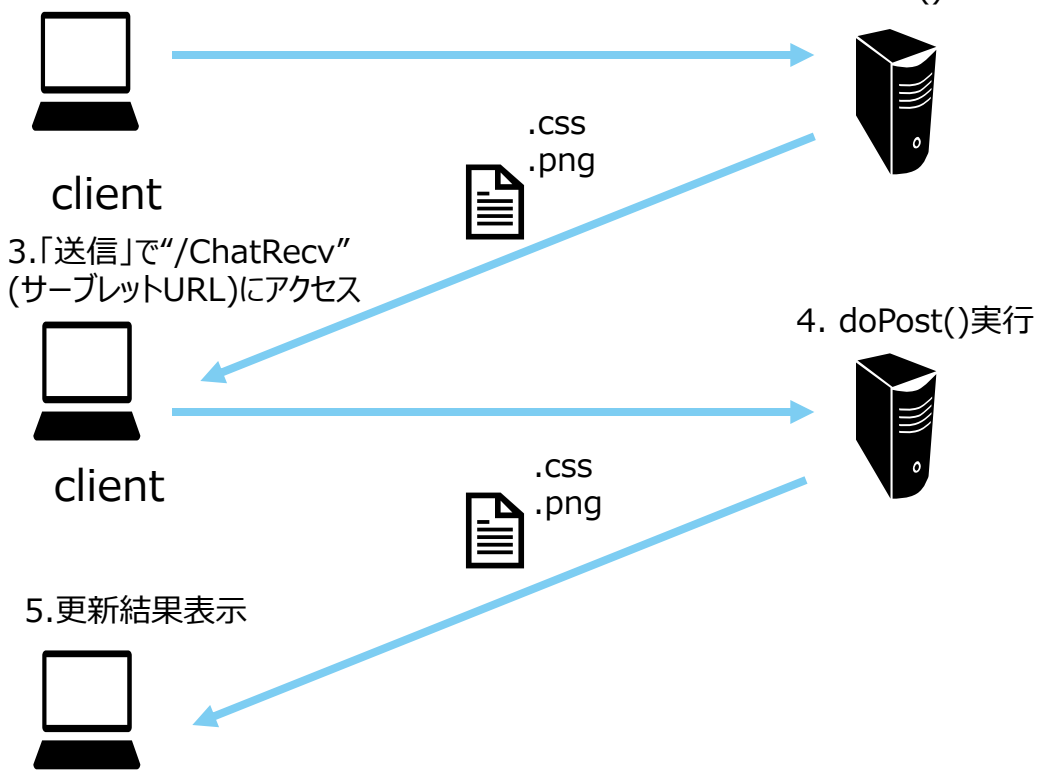
- IHSを介した構成において、Servlet4.0の新機能は未検証

server.xml
```
<featureManager>
        <feature>javaee-8.0</feature>
        <feature>servlet-4.0</feature>
</featureManager>
```

3-2. プログラム概要

- アプリケーション概要

プログラムシナリオ



アプリ挙動

任意の名前、文章の入力と、画像の選択

/ChatRecv

メッセージと選択した画像がグレーゾーンに表示

3-2. プログラム概要

- ディレクトリ構成(Mappingの実行時検出は別プロジェクトで作成)

HelloServlet(一部省略)

```
├── WebContent
│   ├── WEB-INF
│   │   ├── classes
│   │   │   ├── TestServlet.class
│   │   │   └── chat
│   │   │       ├── ChatRecv.class
│   │   │       └── ChatRecvPush.class
│   │   │                    .
│   │   ├── lib              .
│   │   └── view
│   │       ├── http2.jsp
│   │       └── recv.jsp
│   │                    .
│   ├── css              .
│   │   └── style.css
│   ├── favicon
│   │   └── TEST.png
│   └── images
│       ├── A.png
│       └── B.png
│                    .
└── src              .
    ├── chat
    │   ├── ChatRecv.java
    │   └── ChatRecvPush.java
    └── trailer
        └── TrailerServlet.java
                     .
                     .
```

クライアントに表示する画面を司るjspファイル

画像やcssなど画面側からリクエストされるファイル群

サーバーサイドの処理
サーブレットを用いて記述

ServletMapping(一部省略)

```
├── WebContent
│   └── WEB-INF
│       ├── classes
│       │   └── map
│       │       ├── Mapping.class
│       │       └── MappingApiList.class
│       ├── lib
│       └── view
│           ├── favicon
│           │   └── TEST.png
│           └── mapping.jsp
└── src
    └── map
        ├── Mapping.java
        └── MappingApiList.java
```

3-3. アクセス

- デフォルトのアクセス方法
  - HTTP/1.1を利用し、SSLを用いない場合
    - httpでアクセス
    - http://localhost:9080/HelloServlet/<Servlet URL>
      プロジェクト名(ServletMappingを用いる場合は"ServletMapping")
  - SSLによるアクセス
    - https://localhost:9443/HelloServlet/<Servlet URL>
      ポートの番号の変更
  - HTTP/2を利用
    - SSLアクセスが必須
    - SSLによるアクセスで、デフォルトで利用可能
    - WebサーバーとブラウザがHTTP/2に対応している必要があり、Libertyは18.0.0.2以降で対応
    - 下記比較のように、SSLかつHTTP/1.1を利用することも可能

server.xml 抜粋

```
<httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint">
```

→ HTTP/2を利用

```
<httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint" protocolVersion="http/1.1">
```

→ SSL、HTTP/1.1を利用

3-4. 新機能検証

- 新機能検証方法
  - HTTP/2の挙動
    - httpによるアクセスとSSLによるアクセスで挙動比較
  - サーバー・プッシュ利用時の挙動
    - サーバープッシュの記述を追加、扱うデータの変更はなし(画面,ロジックの変更はなし)
  - マッピングの実行時検出
    - マッピングの実行時検出を行う記述を追加、画面に表示されるテキストの変更
  - HTTP Trailer
    - HTTP Trailerを使用するプログラムを追加、HTTP/1.1でも利用可能

3-3-1. HTTP/2の挙動

HTTP/2インディケーターは作動しない

- HTTP/1.1でアクセス

100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms

TCP接続が複数生成

Name	Protocol	Remote.	Initiator	Size	Waterfall
style.css /HelloServlet/css	http/1.1	127.0.0....	Welcome... Parser	1.4 KB 1.2 KB	
A.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	2.6 KB 2.4 KB	
B.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	2.6 KB 2.4 KB	
C.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	3.1 KB 2.9 KB	
D.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	2.3 KB 2.1 KB	
E.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	1.5 KB 1.3 KB	
F.png /HelloServlet/images	http/1.1	127.0.0....	Welcome... Parser	1.5 KB 1.3 KB	
WelcomeChat /HelloServlet	http/1.1	127.0.0....	Other	1.8 KB 1.6 KB	

Initial connectionによるタイムロス

Queued at 286.52 ms
Started at 352.22 ms

Resource Scheduling	TIME
Queueing	65.69 ms
Connection Start	**TIME**
Stalled	5.75 ms
DNS Lookup	0
Initial connection	4.98 ms
SSL	4.97 ms
Request/Response	**TIME**
Request sent	74 µs
Waiting (TTFB)	36.04 ms
Content Download	1.53 ms
Explanation	109.09 ms

3-3-1. HTTP/2の挙動

HTTP/2インディケーター作動

- HTTP/2でアクセス

3-3-2. サーバー・プッシュ利用時の挙動

- サーバープッシュを用いる
  - サーバーサイドのプログラムに記述
  - HTTP/2を用いることが必須

WelcomeChatPush.java

```
package chat;

…
import javax.servlet.http.PushBuilder;

@WebServlet("/WelcomeChat")
…
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        request.setCharacterEncoding("utf-8");
        request.setAttribute("text0", "初めまして！");

        PushBuilder pushBuilder = request.newPushBuilder();
        if (pushBuilder != null) {
                pushBuilder.path("images/A.png").push();
                pushBuilder.path("images/B.png").push();
                pushBuilder.path("css/style.css").push();
                        …
        }
        …
}
```

PushBuilderインターフェースをインポート

サーブレットURL

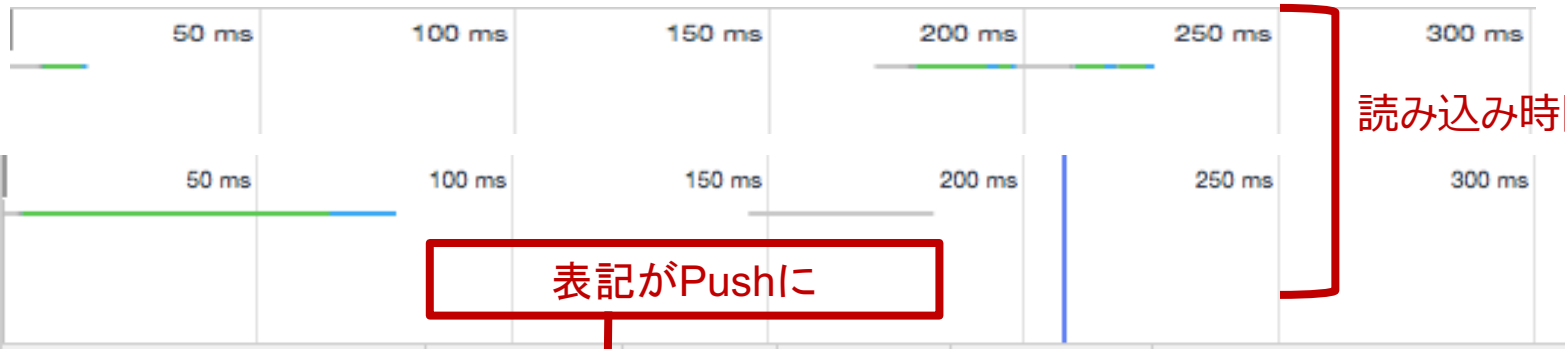
・pushBuilderの中身がNullかを判別
・HTTP/2ではpushbuilderオブジェクトが存在するが、HTTP/1.1では中身がnullになる

Pushするファイルを指定

PushBuilder インターフェースを介し、newPushBuilderメソッドを用いる

3-3-2. サーバー・プッシュ利用時の挙動

HTTP/2インディケーター作動



読み込み時間のさらなる短縮

表記がPushに

Name	Protocol	Remote...	Initiator	Size	Waterfall
WelcomeChatPush localhost/HelloServlet	h2	127.0.0....	Other	2.3 KB 1.6 KB	
style.css /HelloServlet/css	h2	127.0.0....	Push / W... Parser	1.3 KB 1.2 KB	
F.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	1.4 KB 1.3 KB	
E.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	1.4 KB 1.3 KB	
D.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	2.2 KB 2.1 KB	
C.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	3.0 KB 2.9 KB	
B.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	2.5 KB 2.4 KB	
A.png /HelloServlet/images	h2	127.0.0....	Push / W... Parser	2.5 KB 2.4 KB	

Queued at 148.11 ms
Started at 177.60 ms

Waiting 消失

Resource Scheduling — TIME
Queueing — 29.49 ms

Request/Response — TIME
Reading Push — Push の読み込み — 0.29 ms

Explanation — 31.55 ms

3-3-3. マッピング実行時検出

- マッピングの実行時検出

Mapping.java

```
@WebServlet({"", "/", "/WelcomeChatMap",  "*.ext", "/path/*"}
```

Servletが実行される時のURLパターンを複数記述

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpServletMapping mapping = request.getHttpServletMapping();
```

→ HttoServletMappingインターフェースを介し、getHttpServletMappingメソッドを用いる

```
    String map = mapping.getMappingMatch().name();
    String value = mapping.getMatchValue();
    String pattern = mapping.getPattern();
    String servletName = mapping.getServletName();
```

→ 一致するタイプを返す
→ リクエストが一致する原因となったURLパスの該当部分を返す
→ URLパターンを返す
→ サーブレット名を返す

```
String view = "WEB-INF/view/mapping.jsp";
getServletContext().getRequestDispatcher(view).forward(request, response);
```

→ 結果をClientに渡す

```
}
```

3-3-3. マッピング実行時検出

- 検証方法詳細
  - アクセスするURL :localhost:9080/ServletMapping～
  - 「～」を下記表のように変更
    - /
    - /index.html
    - /WelcomeChatMap
    - /Welcome.ext
    - /path/*
  - 「～」の変更によって□の変化を検証

ex)
localhost:9080/ServletMapping/
にアクセス

変化を検証

このサーブレット下記のように呼び出されました
MappingMatch:CONTEXT_ROOT
MappingValue:
MappingPattern:
ServletName:map.Mapping

3-3-3. マッピング実行時検出

- 結果
  - 入力URI Pathに対する、MappingValue、MappingPattern、MappingMatch、ServletNameは以下の通り

入力　URI Path	MappingValue	MappingPattern	MappingMatch	ServletName
/	“”	“”	CONTEXT_ROOT	map.Mapping
/index.html	“”	/	DEFAULT	map.Mapping
/WelcomeChatMap	WelcomeChatMap	/WelcomeChatMap	EXACT	map.Mapping
/Welcome.ext	Welcome	*.ext	EXTENTION	map.Mapping
/path/test	test	/path/*	PATH	map.Mapping

マッピング検出結果

Mapping.java

```
@WebServlet ( { “” , "/” , "/WelcomeChatMap” , "*.ext” , "/path/*” } )
```

3-3-4. HTTP Trailer

- リクエストヘッダーに trailer: foo, bar を指定し、リクエストボディの末尾にfoo: hi, bar: ho を追加して検証

TestTrailer.java

```
protected void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
...
    res.addHeader("TE", "trailers");
    res.addHeader("Trailer", "size, messagefoo, messagebar");

if (req.isTrailerFieldsReady()) {
        Map<String, String> reqTrailerFields = req.getTrailerFields();
        size = reqTrailerFields.size();
        foo = reqTrailerFields.get("foo");
        bar = reqTrailerFields.get("bar");
    }

    final int finalSize = size;
    final String finalFoo1 = foo;
    final String finalFoo2 = bar;
    res.setTrailerFields(() -> {
        Map<String, String> map = new HashMap<>();
        map.put("size", "the size is "+finalSize);
        map.put("messagefoo", finalFoo1);
        map.put("messagebar", finalFoo2);
        return map;
    });
}
```

Trailerを有効にする

レスポンスにおけるTrailerヘッダーを設定

Trailer Fieldが有効か判断(デフォルトでTrue)

Trailer Field の数を取得

リクエストで指定したTrailer

Trailer Field の foo、bar に対する値を取得

Mapクラスを作成し、その属性にtrailerヘッダーを設定 、値を下記のようにセット

size: trailer field数
messagefoo: 上記のfooに対する入力値
messagebar: 上記のbarに対する入力値

3-3-4. HTTP Trailer

- 結果
  - Trailer Fieldの内容の編集、情報の追加が可能
  - trailerヘッダーの指定が必要

```
POST /HelloServlet/TrailerServlet HTTP/1.1
Host: localhost
Transfer-encoding: chunked
Connection: close
trailer: foo, bar

5
hello
0
foo: hi
bar: ho
```

リクエストヘッダー

リクエストボディ

Trailer Field

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/4.0
Content-Type: text/plain;charset=ISO-8859-1
Transfer-Encoding: chunked
TE: trailers
Trailer: size, messagefoo, messagebar
Content-Language: ja-JP
Connection: Close
Date: Fri, 22 Mar 2019 07:34:54 GMT

5
hello
0
size: the size is 2
messagefoo: hi
messagebar: ho
```

Trailer Field

3-5 server.xml 構成

```
<server description="new server">
    <featureManager>
        <feature>javaee-8.0</feature>
        <feature>servlet-4.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="f2914486"/>

    <basicRegistry id="basic" realm="BasicRealm">
        <user name="fukui" password="f2914486"/>
    </basicRegistry>

    <httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint"/>

    <applicationManager autoExpand="true"/>

    <applicationMonitor updateTrigger="mbean"/>

    <webApplication id="HelloServlet" location="HelloServlet.war" name="HelloServlet"/>

    <webApplication id="ServletMapping" location="ServletMapping.war" name="ServletMapping"/>

    <pluginConfiguration pluginInstallRoot="/opt/IHS/plugin"/>
</server>
```

javaee-8.0 → ssl-1.0など、sslに関する構成も含まれる

keyStore → デフォルトの鍵ストアおよび証明書を作成

user → 基本ユーザー・レジストリーのセットアップ

webApplication → 稼働させるアプリケーションの指定

pluginConfiguration → IHSプラグイン構成ファイル plugin-cfg.xml 作成
(IHSを用いる場合追記)

END