HTTP/2 & Servlet 4

Jeff Zhang

weibo.com/findapple

永源中间件 www.useopen.com

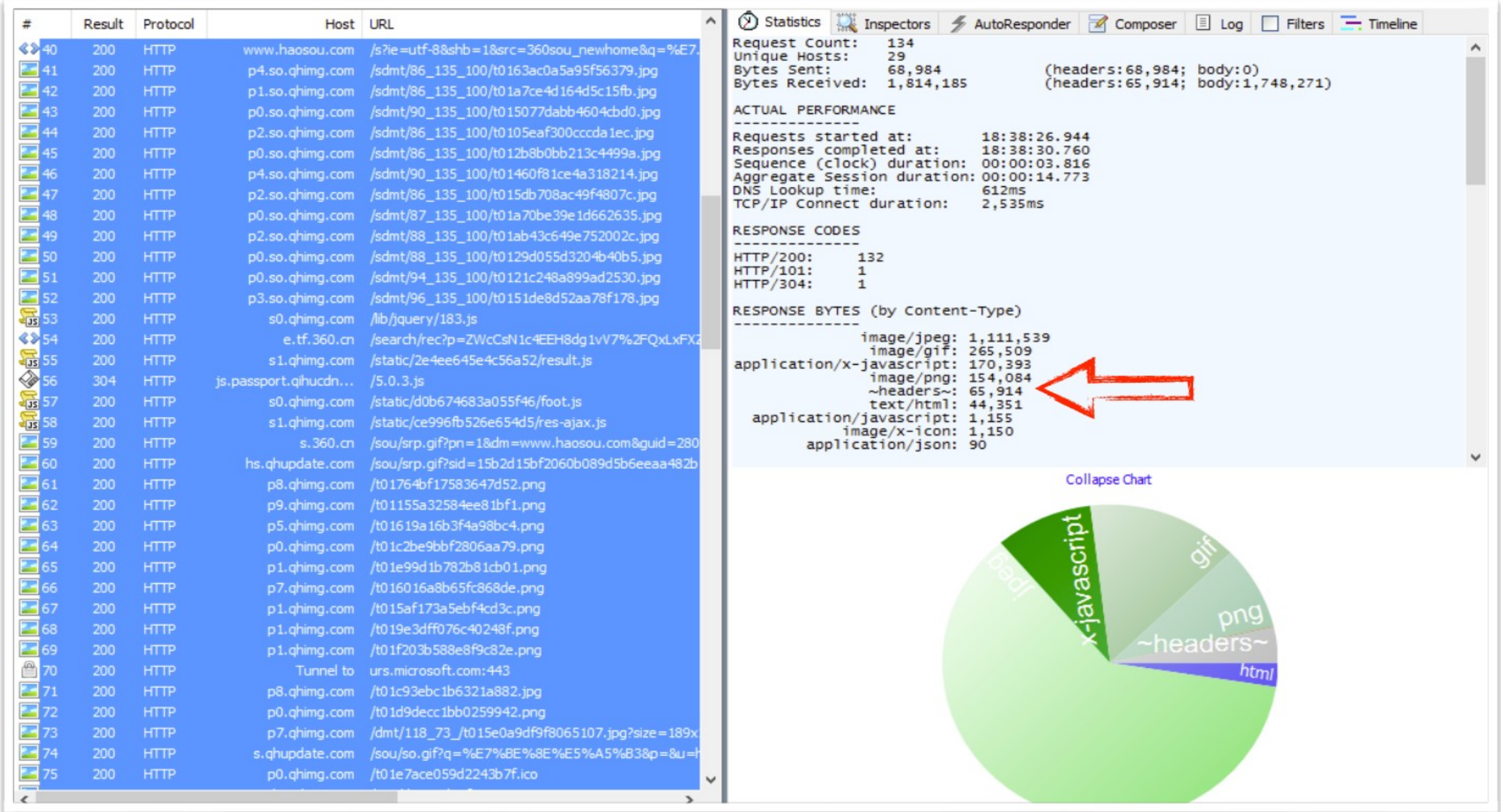
No.26 Beijing *2017/4/23*

HTTP 1.1

- 一个 TCP 连接上只能同时有一个请求 / 响应
- 浏览器对同一个域服务器并发访问有限制
- 协议开销，即使是“空”的响应也有庞大的头部信息

访问请求头部占比

协议开销



#	Result	Protocol	Host	URL
40	200	HTTP	www.haosou.com	/s?ie=utf-8&shb=1&src=360sou_newhome&q=%E7.
41	200	HTTP	p4.so.qhimg.com	/sdmt/86_135_100/t0163ac0a5a95f56379.jpg
42	200	HTTP	p1.so.qhimg.com	/sdmt/86_135_100/t01a7ce4d164d5c15fb.jpg
43	200	HTTP	p0.so.qhimg.com	/sdmt/90_135_100/t015077dabb4604cbd0.jpg
44	200	HTTP	p2.so.qhimg.com	/sdmt/86_135_100/t0105eaf300cccda1ec.jpg
45	200	HTTP	p0.so.qhimg.com	/sdmt/86_135_100/t012b8b0bb213c4499a.jpg
46	200	HTTP	p4.so.qhimg.com	/sdmt/90_135_100/t01460f81ce4a318214.jpg
47	200	HTTP	p2.so.qhimg.com	/sdmt/86_135_100/t015db708ac49f4807c.jpg
48	200	HTTP	p0.so.qhimg.com	/sdmt/87_135_100/t01a70be39e1d662635.jpg
49	200	HTTP	p2.so.qhimg.com	/sdmt/88_135_100/t01ab43c649e752002c.jpg
50	200	HTTP	p0.so.qhimg.com	/sdmt/88_135_100/t0129d055d3204b40b5.jpg
51	200	HTTP	p0.so.qhimg.com	/sdmt/94_135_100/t0121c248a899ad2530.jpg
52	200	HTTP	p3.so.qhimg.com	/sdmt/96_135_100/t0151de8d52aa78f178.jpg
53	200	HTTP	s0.qhimg.com	/lib/jquery/183.js
54	200	HTTP	e.tf.360.cn	/search/rec?p=ZWcCsN1c4EEH8dg1vV7%2FQxLxFX
55	200	HTTP	s1.qhimg.com	/static/2e4ee645e4c56a52/result.js
56	304	HTTP	js.passport.qihucdn...	/5.0.3.js
57	200	HTTP	s0.qhimg.com	/static/d0b674683a055f46/foot.js
58	200	HTTP	s1.qhimg.com	/static/ce996fb526e654d5/res-ajax.js
59	200	HTTP	s.360.cn	/sou/srp.gif?pn=1&dm=www.haosou.com&guid=280
60	200	HTTP	hs.qhupdate.com	/sou/srp.gif?sid=15b2d15bf2060b089d5b6eeaa482b
61	200	HTTP	p8.qhimg.com	/t01764bf17583647d52.png
62	200	HTTP	p9.qhimg.com	/t01155a32584ee81bf1.png
63	200	HTTP	p5.qhimg.com	/t01619a16b3f4a98bc4.png
64	200	HTTP	p0.qhimg.com	/t01c2be9bbf2806aa79.png
65	200	HTTP	p1.qhimg.com	/t01e99d1b782b81cb01.png
66	200	HTTP	p7.qhimg.com	/t016016a8b65fc868de.png
67	200	HTTP	p1.qhimg.com	/t015af173a5ebf4cd3c.png
68	200	HTTP	p1.qhimg.com	/t019e3dff076c40248f.png
69	200	HTTP	p1.qhimg.com	/t01f203b588e8f9c82e.png
70	200	HTTP	Tunnel to	urs.microsoft.com:443
71	200	HTTP	p8.qhimg.com	/t01c93ebc1b6321a882.jpg
72	200	HTTP	p0.qhimg.com	/t01d9decc1bb0259942.png
73	200	HTTP	p7.qhimg.com	/dmt/118_73_/t015e0a9df9f8065107.jpg?size=189x
74	200	HTTP	s.qhupdate.com	/sou/so.gif?q=%E7%BE%8E%E5%A5%B3&p=&u=h
75	200	HTTP	p0.qhimg.com	/t01e7ace059d2243b7f.ico

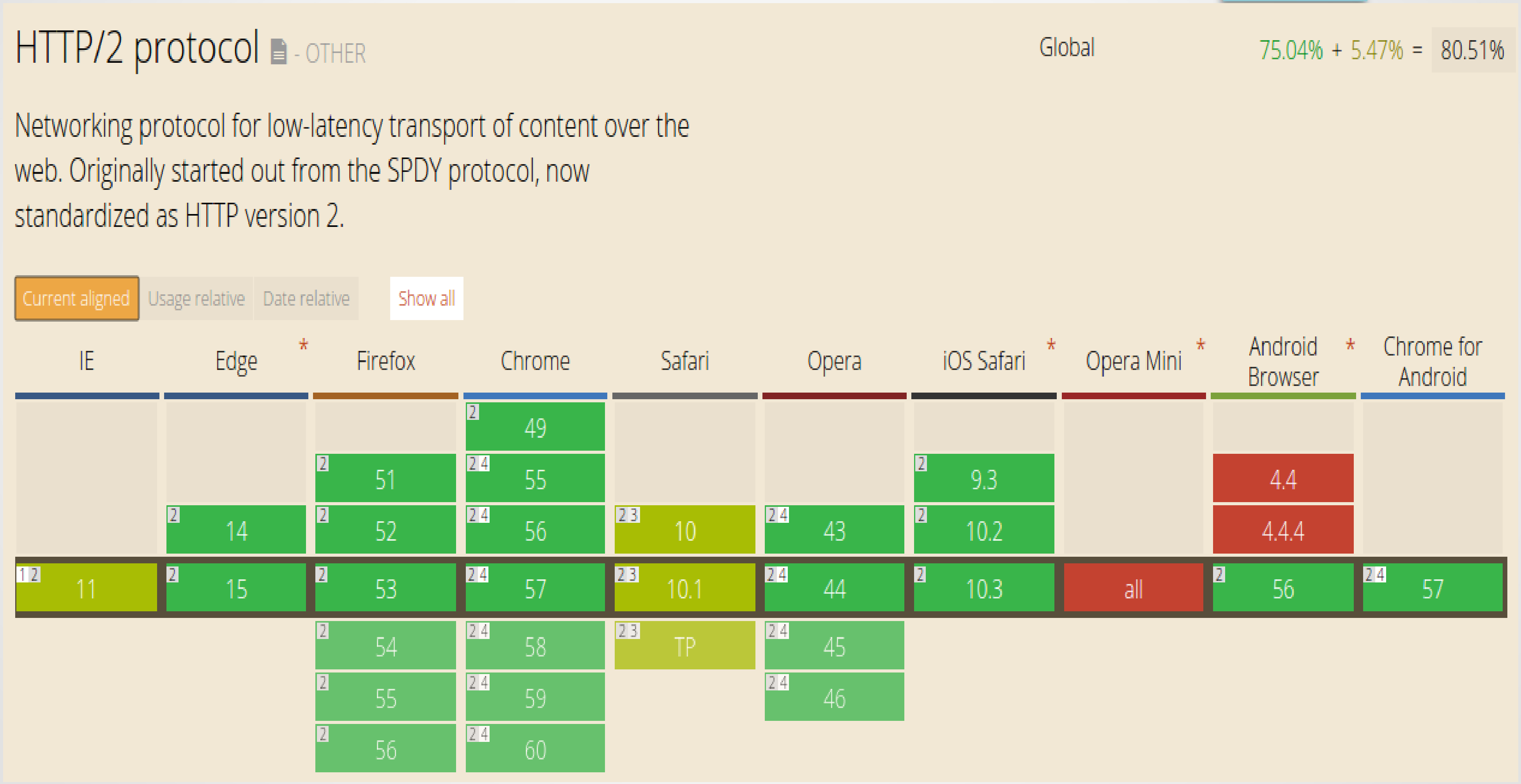
```
Request Count:   134
Unique Hosts:    29
Bytes Sent:      68,984          (headers:68,984; body:0)
Bytes Received:  1,814,185       (headers:65,914; body:1,748,271)

ACTUAL PERFORMANCE
--------------
Requests started at:        18:38:26.944
Responses completed at:     18:38:30.760
Sequence (clock) duration:  00:00:03.816
Aggregate Session duration: 00:00:14.773
DNS Lookup time:            612ms
TCP/IP Connect duration:    2,535ms

RESPONSE CODES
--------------
HTTP/200:     132
HTTP/101:     1
HTTP/304:     1

RESPONSE BYTES (by Content-Type)
--------------
             image/jpeg: 1,111,539
              image/gif: 265,509
application/x-javascript: 170,393
              image/png: 154,084
              ~headers~: 65,914
              text/html: 44,351
  application/javascript: 1,155
           image/x-icon: 1,150
       application/json: 90
```

• *以下很多图片来源于网络，版权归原作者*

HTTP/2 协议

- RFC 7540 Hypertext Transfer Protocol Version 2
- RFC 7541 HPACK: Header Compression for HTTP/2

当前主流浏览器支持



HTTP/2 protocol - OTHER

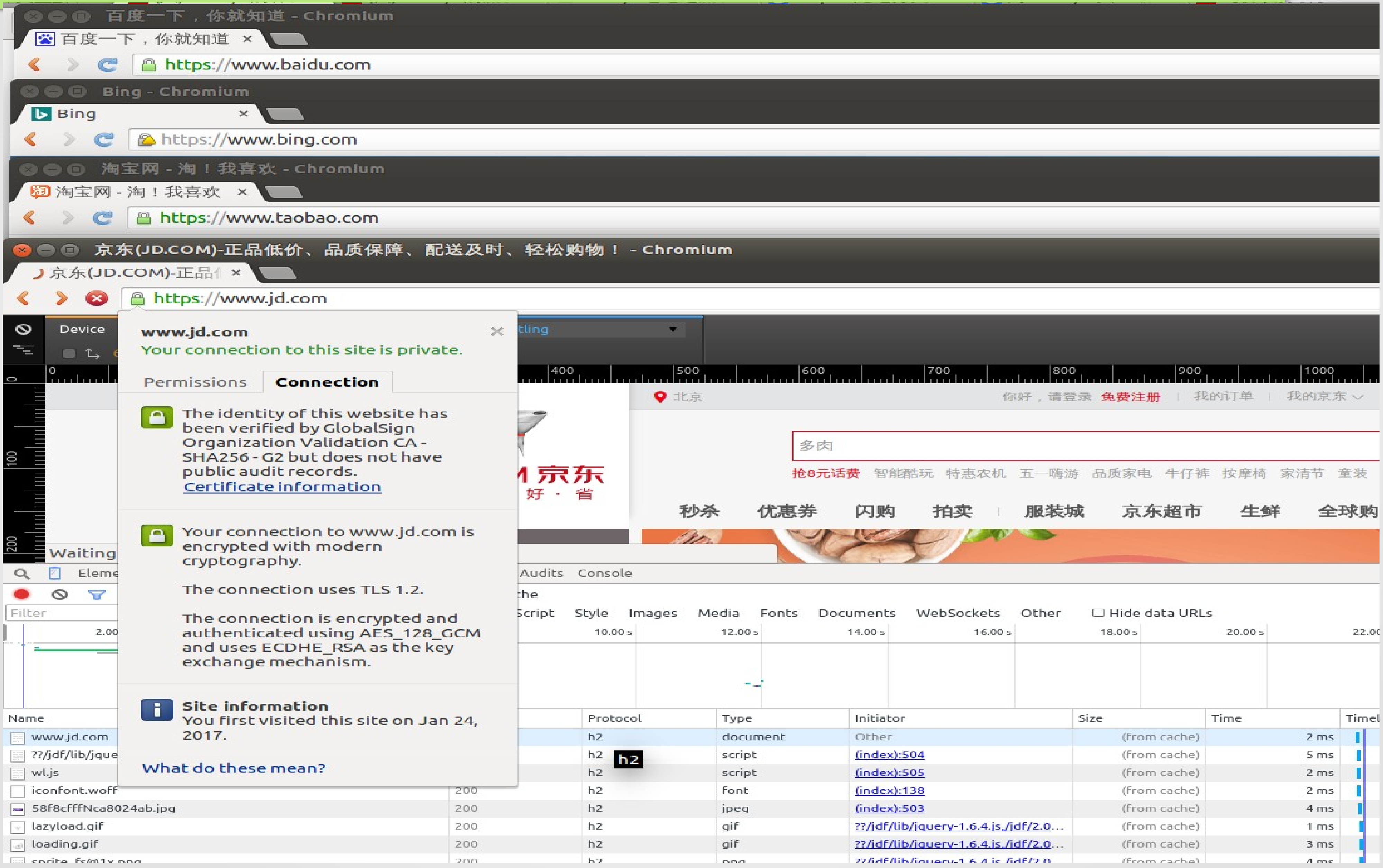
Global 75.04% + 5.47% = 80.51%

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.

Current aligned | Usage relative | Date relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			[2] 49						
		[2] 51	[2,4] 55			[2] 9.3		4.4	
	[2] 14	[2] 52	[2,4] 56	[2,3] 10	[2,4] 43	[2] 10.2		4.4.4	
[1,2] 11	[2] 15	[2] 53	[2,4] 57	[2,3] 10.1	[2,4] 44	[2] 10.3	all	[2] 56	[2,4] 57
		[2] 54	[2,4] 58	[2,3] TP	[2,4] 45				
		[2] 55	[2,4] 59		[2,4] 46				
		[2] 56	[2,4] 60						

国内主要网站支持情况



百度一下，你就知道 - Chromium

百度一下，你就知道

https://www.baidu.com

Bing - Chromium

Bing

https://www.bing.com

淘宝网 - 淘！我喜欢 - Chromium

淘宝网 - 淘！我喜欢

https://www.taobao.com

京东(JD.COM)-正品低价、品质保障、配送及时、轻松购物！ - Chromium

京东(JD.COM)-正品

https://www.jd.com

www.jd.com

Your connection to this site is private.

Permissions | **Connection**

The identity of this website has been verified by GlobalSign Organization Validation CA - SHA256 - G2 but does not have public audit records.
Certificate information

Your connection to www.jd.com is encrypted with modern cryptography.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.

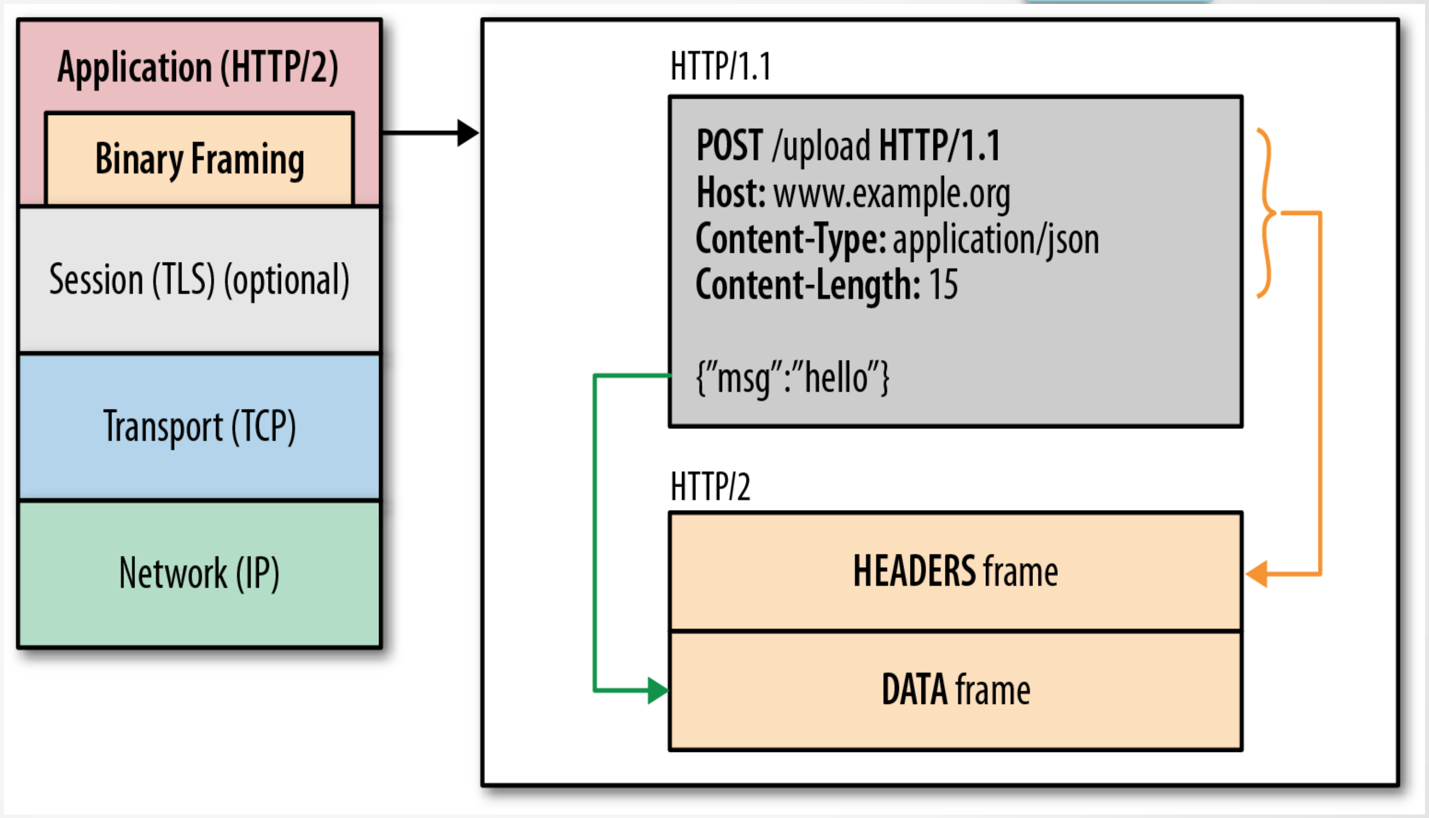
Site information
You first visited this site on Jan 24, 2017.

What do these mean?

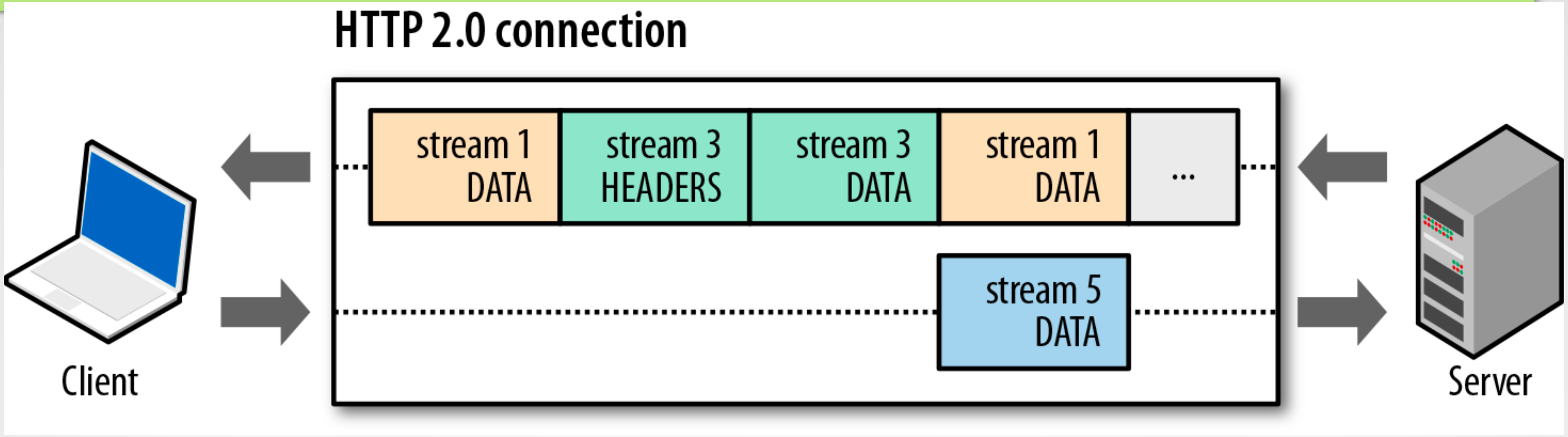
Name	Status	Protocol	Type	Initiator	Size	Time
www.jd.com		h2	document	Other	(from cache)	2 ms
??/jdf/lib/jque		h2	script	(index):504	(from cache)	5 ms
wl.js		h2	script	(index):505	(from cache)	2 ms
iconfont.woff	200	h2	font	(index):138	(from cache)	2 ms
58f8cfffNca8024ab.jpg	200	h2	jpeg	(index):503	(from cache)	4 ms
lazyload.gif	200	h2	gif	??/jdf/lib/jquery-1.6.4.js,/jdf/2.0…	(from cache)	1 ms
loading.gif	200	h2	gif	??/jdf/lib/jquery-1.6.4.js,/jdf/2.0…	(from cache)	3 ms

HTTP/2

- 一个 TCP 连接
- 流式请求
  - 双工的
  - 优先级
- 二进制传输帧
  - 流控
  - 服务器推送
- HPACK 头部压



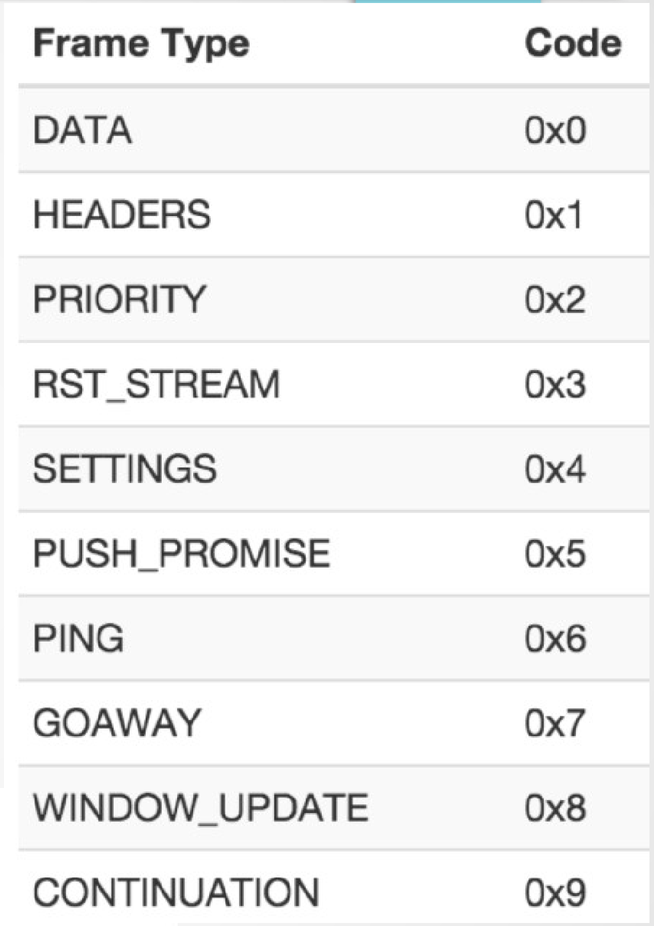
数据流



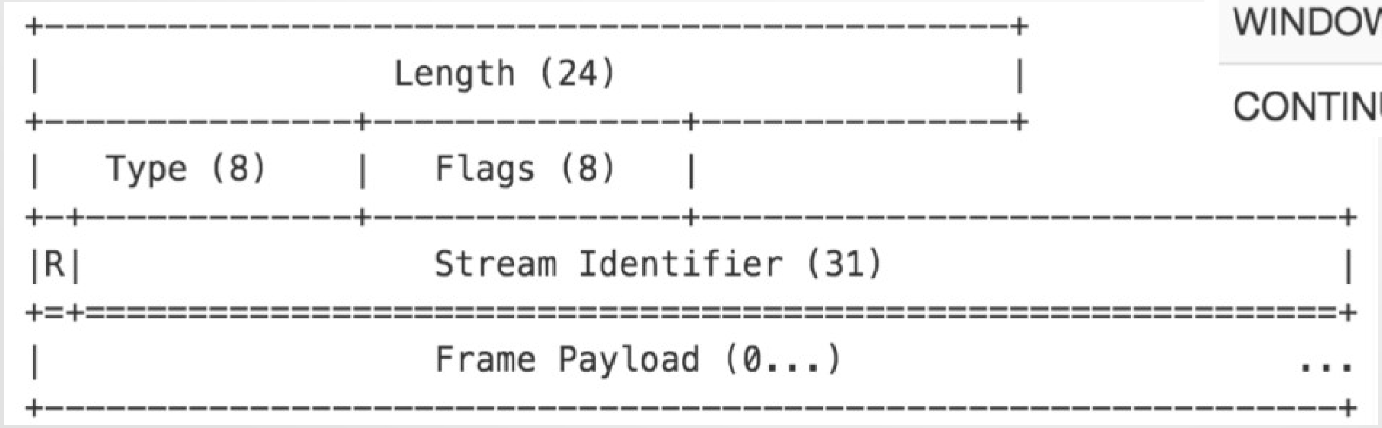
- 多路复用的帧格式
- 在一条 TCP 连接上传输
- 可以具有优先级
- 数据帧具有流控功能

二进制帧格式

- 每个帧有通用的头
  - 9 个字节，长度固定，便于解析
- HTTP 消息分解为多个帧
  - HEADERS 元数据
  - DATA payload 数据
  - RST_STREAM 取消



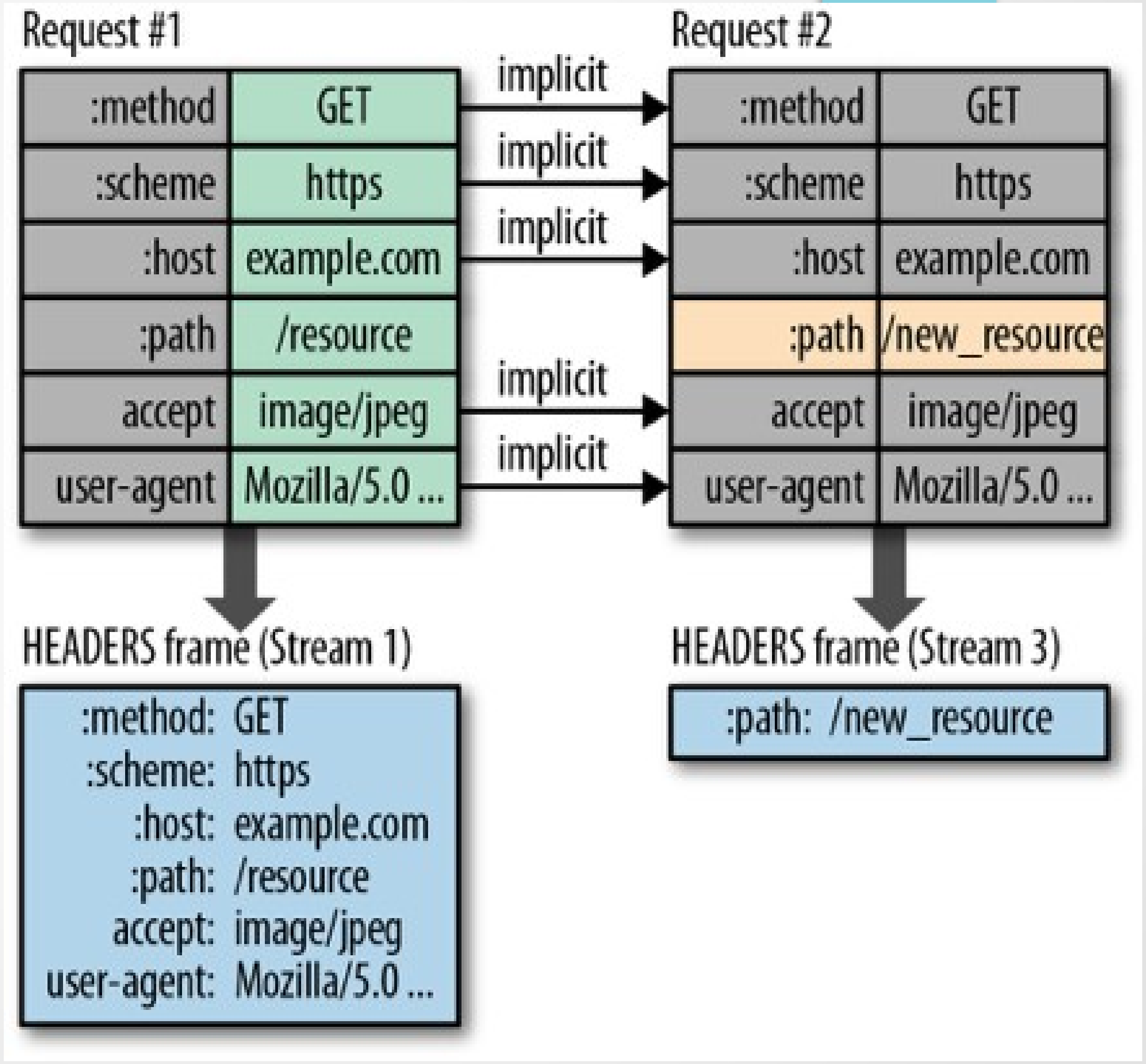
Frame Type	Code
DATA	0x0
HEADERS	0x1
PRIORITY	0x2
RST_STREAM	0x3
SETTINGS	0x4
PUSH_PROMISE	0x5
PING	0x6
GOAWAY	0x7
WINDOW_UPDATE	0x8
CONTINUATION	0x9



```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+=+=============================================================+
|                   Frame Payload (0...)                      ...
+---------------------------------------------------------------+
```

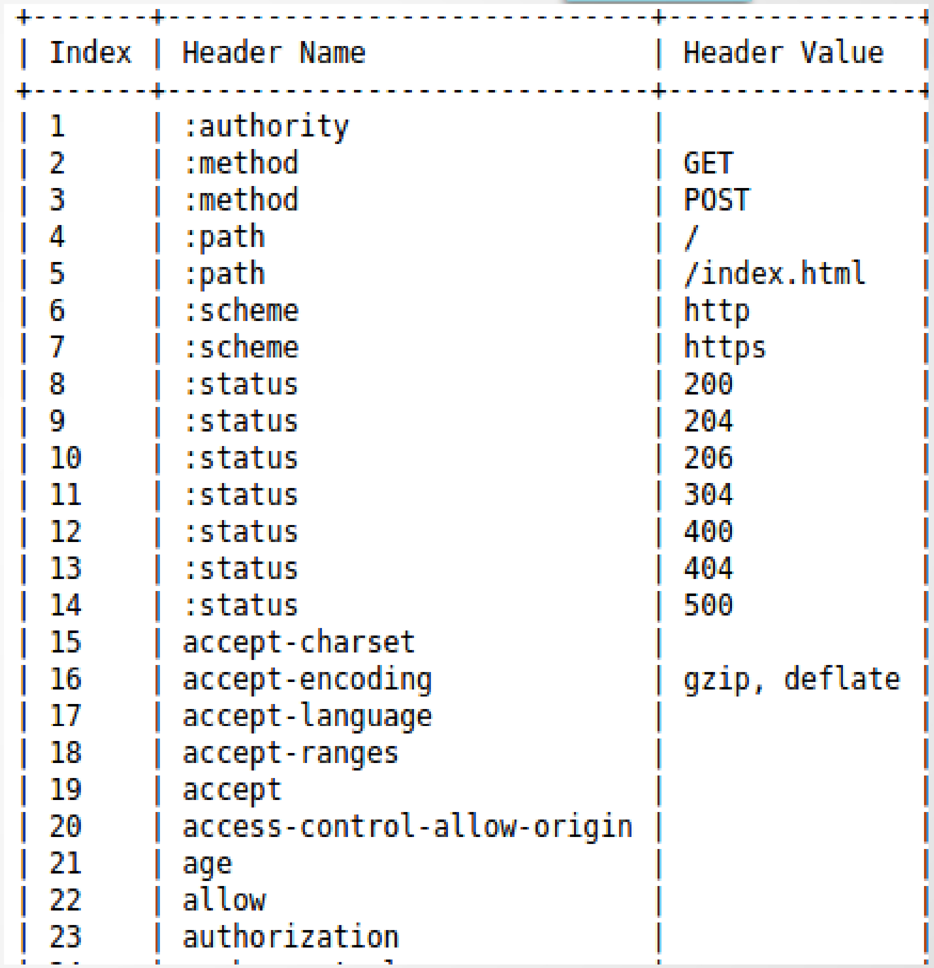
头部压缩

- 使用哈夫曼编码
- 使用过的消息都进行了编码
- 2 个索引表，静态和动态



静态索引表

- 要发送的值符合静态表时，用对应的 Index 替换即可，这样就大大压缩了头部的大小。
- 预先定义好的，只有固定的几十个值，如果遇到不在静态表中的值，就会用到动态表。



Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
17	accept-language	
18	accept-ranges	
19	accept	
20	access-control-allow-origin	
21	age	
22	allow	
23	authorization	

动态索引表

- 每个连接的压缩解压缩的上下文有且仅有一个动态表
- 当一个头部没有出现过的时候，会插入动态表中，下次同名的值就可能会在表中查到到索引并替换掉头部。
- 动态表的最大字节数由 HTTP/2 的 SETTING 帧中的 SETTINGS_HEADER_TABLE_SIZE 来控制

解析范例

- 8286 8441 8cf1 e3c2 e5f2 3a6b a0ab 90f4 ff
  - 82 = 10000010 -> 静态表 Index = 2 -> :method: GET
  - 86 = 10000110 -> 静态表 Index = 6 -> :scheme: http
  - 84 = 10000100 -> 静态表 Index = 4 -> :path: /
  - 41 = 01000001 -> name = 静态表 1 = :authority
- 8c = 10001100 -> 第一个 bit 为 1 ，表示 huffman 编码，接着解析 12 个字节。 huffman 编码后的字符 f1e3 c2e5 f23a 6ba0 ab90 f4ff ， 查表可知为 www.example.com

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

HTTP 帧的类型

DATA	0x0	一个或多个携带 HTTP 的请求和响应的 Payload
HEADERS	0x1	包含一个报文头部分片段
PRIORITY	0x2	明确了发送者建议的流的优先级
RST_STREAM	0x3	允许立即终止一个流
SETTINGS	0x4	设置帧传递通信的配置参数
PUSH_PROMISE	0x5	提前将发送方打算初始化的流通知给对端
PING	0x6	一种检测空闲连接是否可用的机制
GOAWAY	0x7	初始化连接的关闭过程，或者将严重的错误通知给对端
WINDOW_UPDATE	0x8	用于实现流量控制
CONTINUATION	0x9	用于延续一系列报头块

Servlet 4.0

- JSR 369
- Leaders Edward Burns / Shing Wai Chan
- 公布了早期规范草案文本和 API
- Tomcat9/Jetty9.4/Undertow 都已经实现
  - 后两者目前实现的较完备

EE 支持的 HTTP/2 特性

- 以下特性会在 Servlet4 中体现出来
- 请求响应多路复用
- 二进制帧
- 流控
- Stream Prioritization
- Server Push
- 头部压缩
- Upgrade from HTTP 1.1(101 Switch Protocols)

最大的区别：就是原来的每个请求对应一个处理线程的前提发生变化

Servlet 规范把协议的复杂性几乎屏蔽了

最大的区别：就是原来的每个请求对应一个处理线程的前提发生变化

Servlet 规范把协议的复杂性几乎屏蔽了

PushBuilder

```
private void pushResource(String relativeResourcePath) {

    final Request jettyRequest = (Request) getRequest();
    final PushBuilder pushBuilder = jettyRequest.getPushBuilder();

    pushBuilder.setQueryString(…)
            .push(relativeResourcePath);

    //…

}
```

- Index.html → {style.css, app.js}
- server 自动推送有关文件

API 变化

- Add Java SE8 default methods
  - ServletContextAttributeListener, ServletContextListener,
  - ServletRequestAttributeListener, ServletRequestListener,
  - HttpSessionActivationListener, HttpSessionAttributeListener,
  - HttpSessionBindingListener, HttpSessionListener
- Add default to Filter#init, #destroy
- Add GenericFilter and HttpFilter

Mapping API

- 查询现有的 Mapping （映射）
- Mapping javax.servlet.http.HttpServletRequest.getMapping()
- javax.servlet.http.Mapping
  - MappingMatch getMatchType()
  - String getMatchValue()
  - String getPattern()
- javax.servlet.http.MappingMatch enum
  - CONTEXT_ROOT, DEFAULT, EXACT, EXTENSION, IMPLICIT, PATH, UNKNOWN

Priority?

- 新的 Priority 类还没有加入
- 在 HttpServletRequest/HttpServletResponse
  - int getStreamId()
  - Priority getPriority()

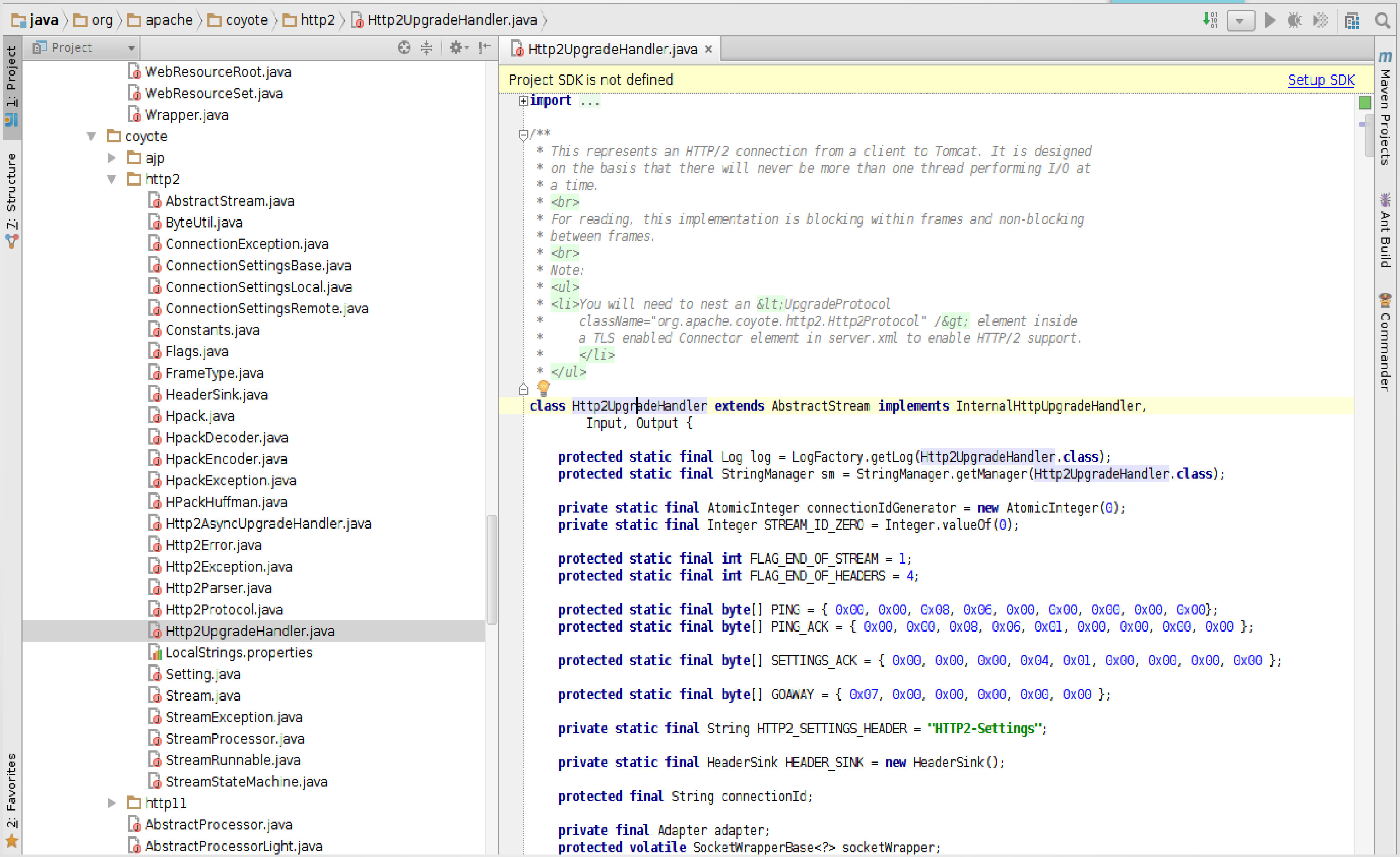
Tomcat9

- org.apache.coyote.http2
- Http2Protocol 协议处理
- Http2Parser 解析，如读取帧格式等
- HpackDecoder 解析头部压缩格式
- Http2UpgradeHandler 协议协商处理器
- Stream/StreamHandler 流处理器

Server.xml 中的配置

- <Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
- maxThreads="150" SSLEnabled="true" >
- <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
- <SSLHostConfig>
- <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
- certificateFile="conf/localhost-rsa-cert.pem"
- certificateChainFile="conf/localhost-rsa-chain.pem"
- type="RSA" />
- </SSLHostConfig>
- </Connector>

Http2UpgradeHandler



java 〉 org 〉 apache 〉 coyote 〉 http2 〉 Http2UpgradeHandler.java 〉

```
WebResourceRoot.java
WebResourceSet.java
Wrapper.java
▼ coyote
  ▶ ajp
  ▼ http2
      AbstractStream.java
      ByteUtil.java
      ConnectionException.java
      ConnectionSettingsBase.java
      ConnectionSettingsLocal.java
      ConnectionSettingsRemote.java
      Constants.java
      Flags.java
      FrameType.java
      HeaderSink.java
      Hpack.java
      HpackDecoder.java
      HpackEncoder.java
      HpackException.java
      HPackHuffman.java
      Http2AsyncUpgradeHandler.java
      Http2Error.java
      Http2Exception.java
      Http2Parser.java
      Http2Protocol.java
      Http2UpgradeHandler.java
      LocalStrings.properties
      Setting.java
      Stream.java
      StreamException.java
      StreamProcessor.java
      StreamRunnable.java
      StreamStateMachine.java
  ▶ http11
    AbstractProcessor.java
    AbstractProcessorLight.java
```

Http2UpgradeHandler.java

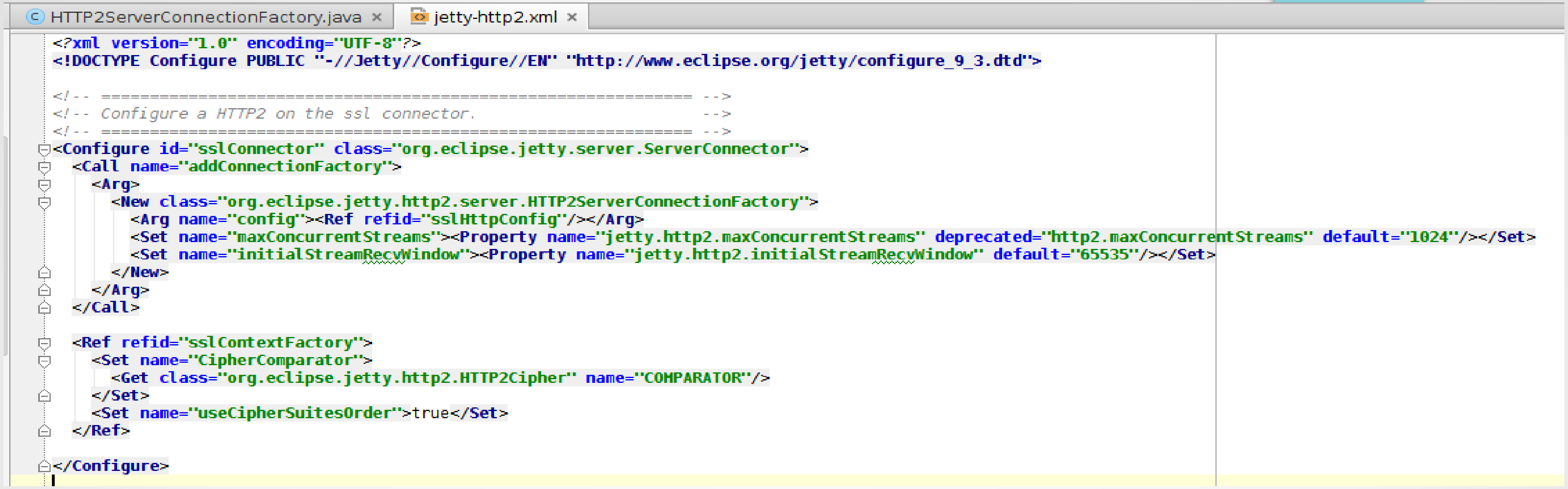
Project SDK is not defined   Setup SDK

```java
import ...

/**
 * This represents an HTTP/2 connection from a client to Tomcat. It is designed
 * on the basis that there will never be more than one thread performing I/O at
 * a time.
 * <br>
 * For reading, this implementation is blocking within frames and non-blocking
 * between frames.
 * <br>
 * Note:
 * <ul>
 * <li>You will need to nest an &lt;UpgradeProtocol
 *     className="org.apache.coyote.http2.Http2Protocol" /&gt; element inside
 *     a TLS enabled Connector element in server.xml to enable HTTP/2 support.
 *     </li>
 * </ul>
 */
class Http2UpgradeHandler extends AbstractStream implements InternalHttpUpgradeHandler,
        Input, Output {

    protected static final Log log = LogFactory.getLog(Http2UpgradeHandler.class);
    protected static final StringManager sm = StringManager.getManager(Http2UpgradeHandler.class);

    private static final AtomicInteger connectionIdGenerator = new AtomicInteger(0);
    private static final Integer STREAM_ID_ZERO = Integer.valueOf(0);

    protected static final int FLAG_END_OF_STREAM = 1;
    protected static final int FLAG_END_OF_HEADERS = 4;

    protected static final byte[] PING = { 0x00, 0x00, 0x08, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00};
    protected static final byte[] PING_ACK = { 0x00, 0x00, 0x08, 0x06, 0x01, 0x00, 0x00, 0x00, 0x00 };

    protected static final byte[] SETTINGS_ACK = { 0x00, 0x00, 0x00, 0x04, 0x01, 0x00, 0x00, 0x00, 0x00 };

    protected static final byte[] GOAWAY = { 0x07, 0x00, 0x00, 0x00, 0x00, 0x00 };

    private static final String HTTP2_SETTINGS_HEADER = "HTTP2-Settings";

    private static final HeaderSink HEADER_SINK = new HeaderSink();

    protected final String connectionId;

    private final Adapter adapter;
    protected volatile SocketWrapperBase<?> socketWrapper;
```

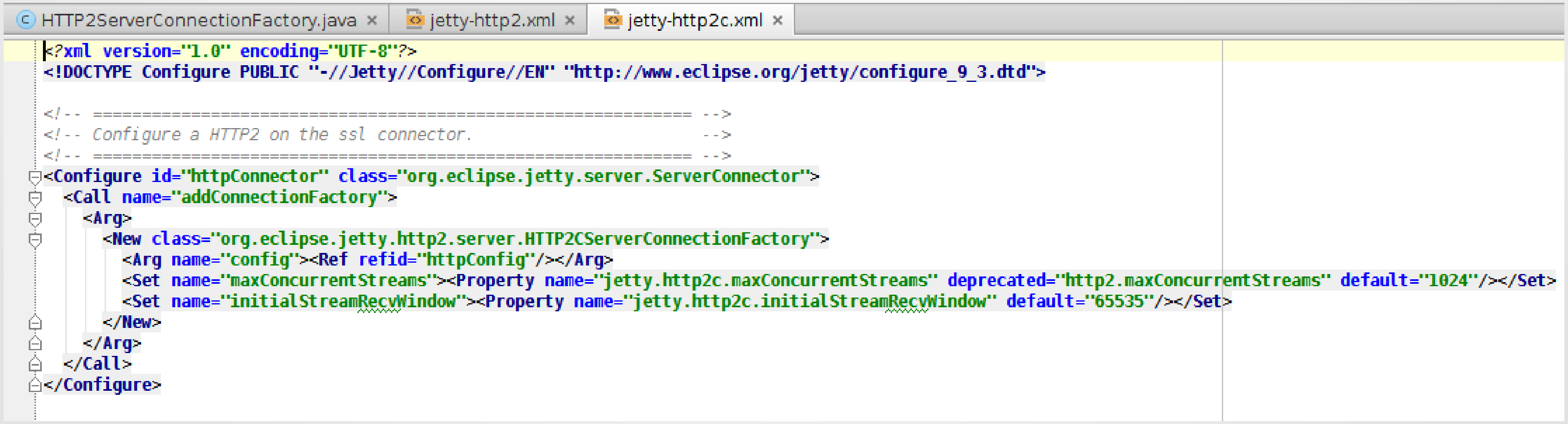
Jetty9.4

- Jetty-http2 包中
- Http2-common
- Http2-server
- Http2-hpack
- Http2-client

两种协议的配置

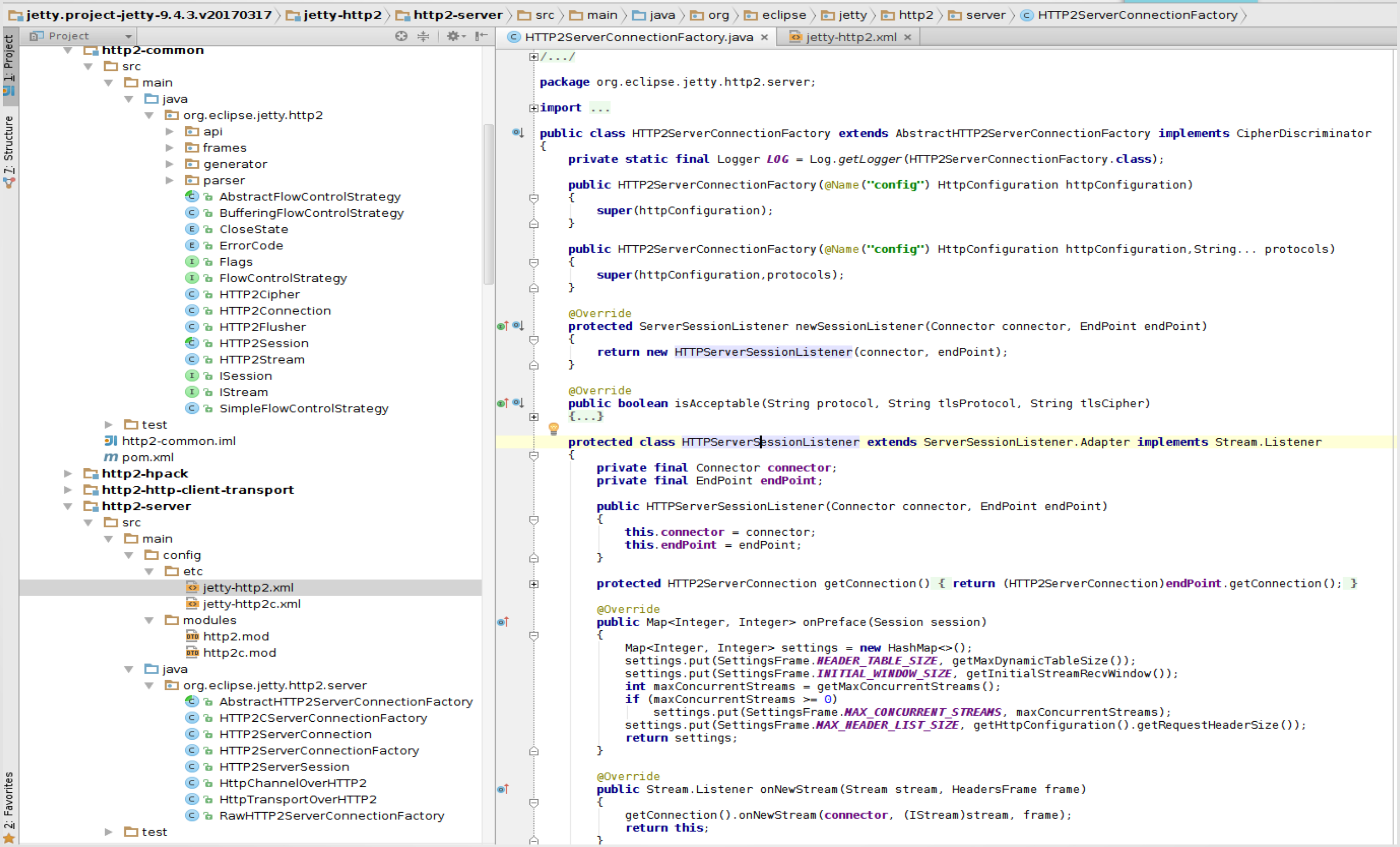


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure_9_3.dtd">

<!-- ============================================================= -->
<!-- Configure a HTTP2 on the ssl connector.                       -->
<!-- ============================================================= -->
<Configure id="sslConnector" class="org.eclipse.jetty.server.ServerConnector">
  <Call name="addConnectionFactory">
    <Arg>
      <New class="org.eclipse.jetty.http2.server.HTTP2ServerConnectionFactory">
        <Arg name="config"><Ref refid="sslHttpConfig"/></Arg>
        <Set name="maxConcurrentStreams"><Property name="jetty.http2.maxConcurrentStreams" deprecated="http2.maxConcurrentStreams" default="1024"/></Set>
        <Set name="initialStreamRecvWindow"><Property name="jetty.http2.initialStreamRecvWindow" default="65535"/></Set>
      </New>
    </Arg>
  </Call>

  <Ref refid="sslContextFactory">
    <Set name="CipherComparator">
      <Get class="org.eclipse.jetty.http2.HTTP2Cipher" name="COMPARATOR"/>
    </Set>
    <Set name="useCipherSuitesOrder">true</Set>
  </Ref>

</Configure>
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure_9_3.dtd">

<!-- ============================================================= -->
<!-- Configure a HTTP2 on the ssl connector.                       -->
<!-- ============================================================= -->
<Configure id="httpConnector" class="org.eclipse.jetty.server.ServerConnector">
  <Call name="addConnectionFactory">
    <Arg>
      <New class="org.eclipse.jetty.http2.server.HTTP2CServerConnectionFactory">
        <Arg name="config"><Ref refid="httpConfig"/></Arg>
        <Set name="maxConcurrentStreams"><Property name="jetty.http2c.maxConcurrentStreams" deprecated="http2.maxConcurrentStreams" default="1024"/></Set>
        <Set name="initialStreamRecvWindow"><Property name="jetty.http2c.initialStreamRecvWindow" default="65535"/></Set>
      </New>
    </Arg>
  </Call>
</Configure>
```

HTTP2ServerConnectionFactory



jetty.project-jetty-9.4.3.v20170317 › jetty-http2 › http2-server › src › main › java › org › eclipse › jetty › http2 › server › HTTP2ServerConnectionFactory ›

```
http2-common
  src
    main
      java
        org.eclipse.jetty.http2
          api
          frames
          generator
          parser
          AbstractFlowControlStrategy
          BufferingFlowControlStrategy
          CloseState
          ErrorCode
          Flags
          FlowControlStrategy
          HTTP2Cipher
          HTTP2Connection
          HTTP2Flusher
          HTTP2Session
          HTTP2Stream
          ISession
          IStream
          SimpleFlowControlStrategy
    test
  http2-common.iml
  pom.xml
http2-hpack
http2-http-client-transport
http2-server
  src
    main
      config
        etc
          jetty-http2.xml
          jetty-http2c.xml
        modules
          http2.mod
          http2c.mod
      java
        org.eclipse.jetty.http2.server
          AbstractHTTP2ServerConnectionFactory
          HTTP2CServerConnectionFactory
          HTTP2ServerConnection
          HTTP2ServerConnectionFactory
          HTTP2ServerSession
          HttpChannelOverHTTP2
          HttpTransportOverHTTP2
          RawHTTP2ServerConnectionFactory
    test
```

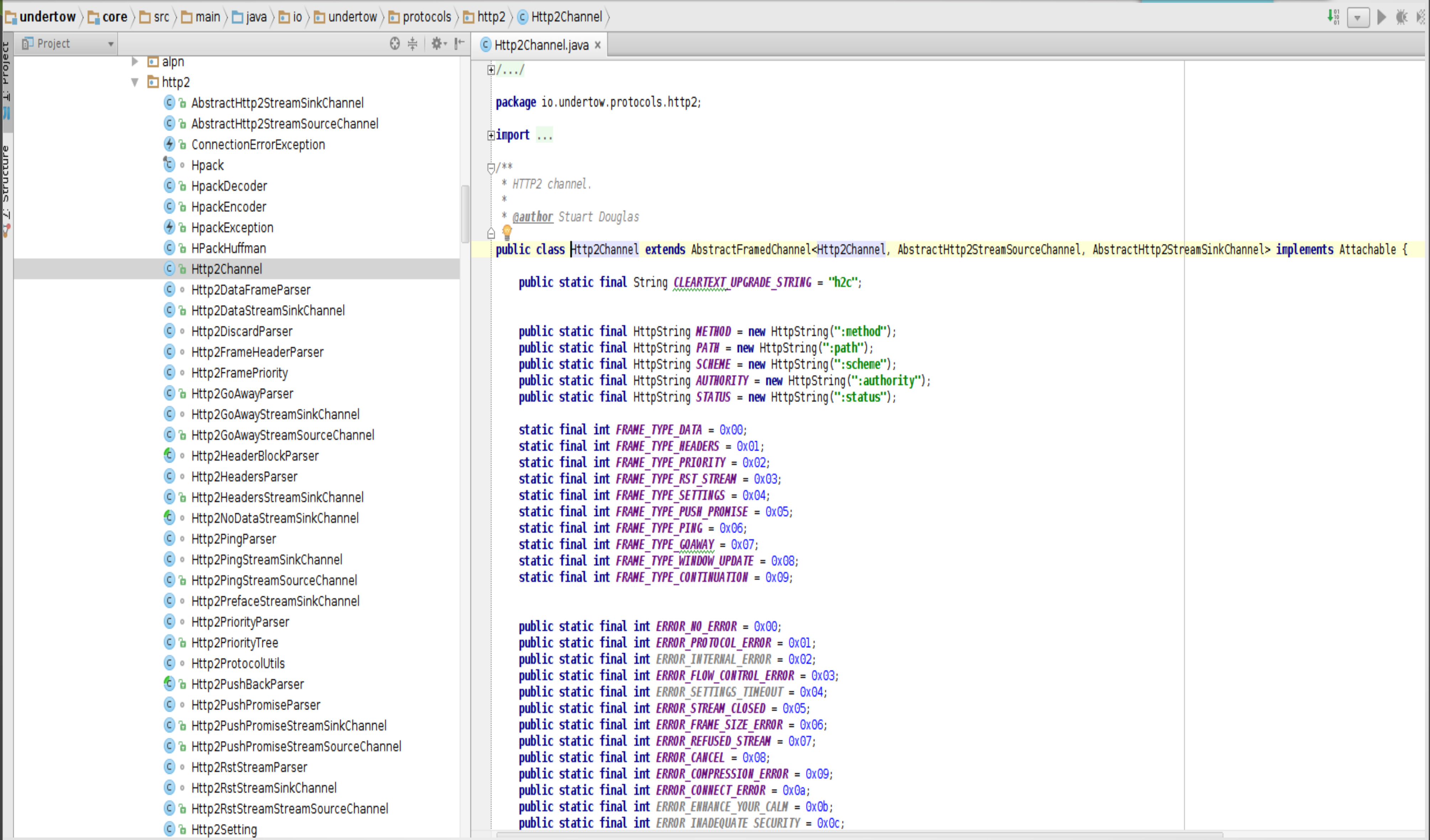
HTTP2ServerConnectionFactory.java × | jetty-http2.xml ×

```java
/.../

package org.eclipse.jetty.http2.server;

import ...

public class HTTP2ServerConnectionFactory extends AbstractHTTP2ServerConnectionFactory implements CipherDiscriminator
{
    private static final Logger LOG = Log.getLogger(HTTP2ServerConnectionFactory.class);

    public HTTP2ServerConnectionFactory(@Name("config") HttpConfiguration httpConfiguration)
    {
        super(httpConfiguration);
    }

    public HTTP2ServerConnectionFactory(@Name("config") HttpConfiguration httpConfiguration,String... protocols)
    {
        super(httpConfiguration,protocols);
    }

    @Override
    protected ServerSessionListener newSessionListener(Connector connector, EndPoint endPoint)
    {
        return new HTTPServerSessionListener(connector, endPoint);
    }

    @Override
    public boolean isAcceptable(String protocol, String tlsProtocol, String tlsCipher)
    {...}

    protected class HTTPServerSessionListener extends ServerSessionListener.Adapter implements Stream.Listener
    {
        private final Connector connector;
        private final EndPoint endPoint;

        public HTTPServerSessionListener(Connector connector, EndPoint endPoint)
        {
            this.connector = connector;
            this.endPoint = endPoint;
        }

        protected HTTP2ServerConnection getConnection() { return (HTTP2ServerConnection)endPoint.getConnection(); }

        @Override
        public Map<Integer, Integer> onPreface(Session session)
        {
            Map<Integer, Integer> settings = new HashMap<>();
            settings.put(SettingsFrame.HEADER_TABLE_SIZE, getMaxDynamicTableSize());
            settings.put(SettingsFrame.INITIAL_WINDOW_SIZE, getInitialStreamRecvWindow());
            int maxConcurrentStreams = getMaxConcurrentStreams();
            if (maxConcurrentStreams >= 0)
                settings.put(SettingsFrame.MAX_CONCURRENT_STREAMS, maxConcurrentStreams);
            settings.put(SettingsFrame.MAX_HEADER_LIST_SIZE, getHttpConfiguration().getRequestHeaderSize());
            return settings;
        }

        @Override
        public Stream.Listener onNewStream(Stream stream, HeadersFrame frame)
        {
            getConnection().onNewStream(connector, (IStream)stream, frame);
            return this;
        }
```

Undertow

- core/src/main/java/io/undertow/protocols/http2
  - 协议处理部分　Http2Stream / Http2PushBackParser / Hpack/Http2Channel/Http2DataFrameParser
- core/src/main/java/io/undertow/server/protocol/http2
  - 服务器处理 Http2OpenListener / Http2ServerConnection /Http2UpgradeHandler
- core/src/main/java/io/undertow/client/http2
  - 客户端处理 Http2ClientConnection / Http2ClientExchange
- servlet/src/main/java/io/undertow/servlet/spec
  - Servlet 4.0 实现

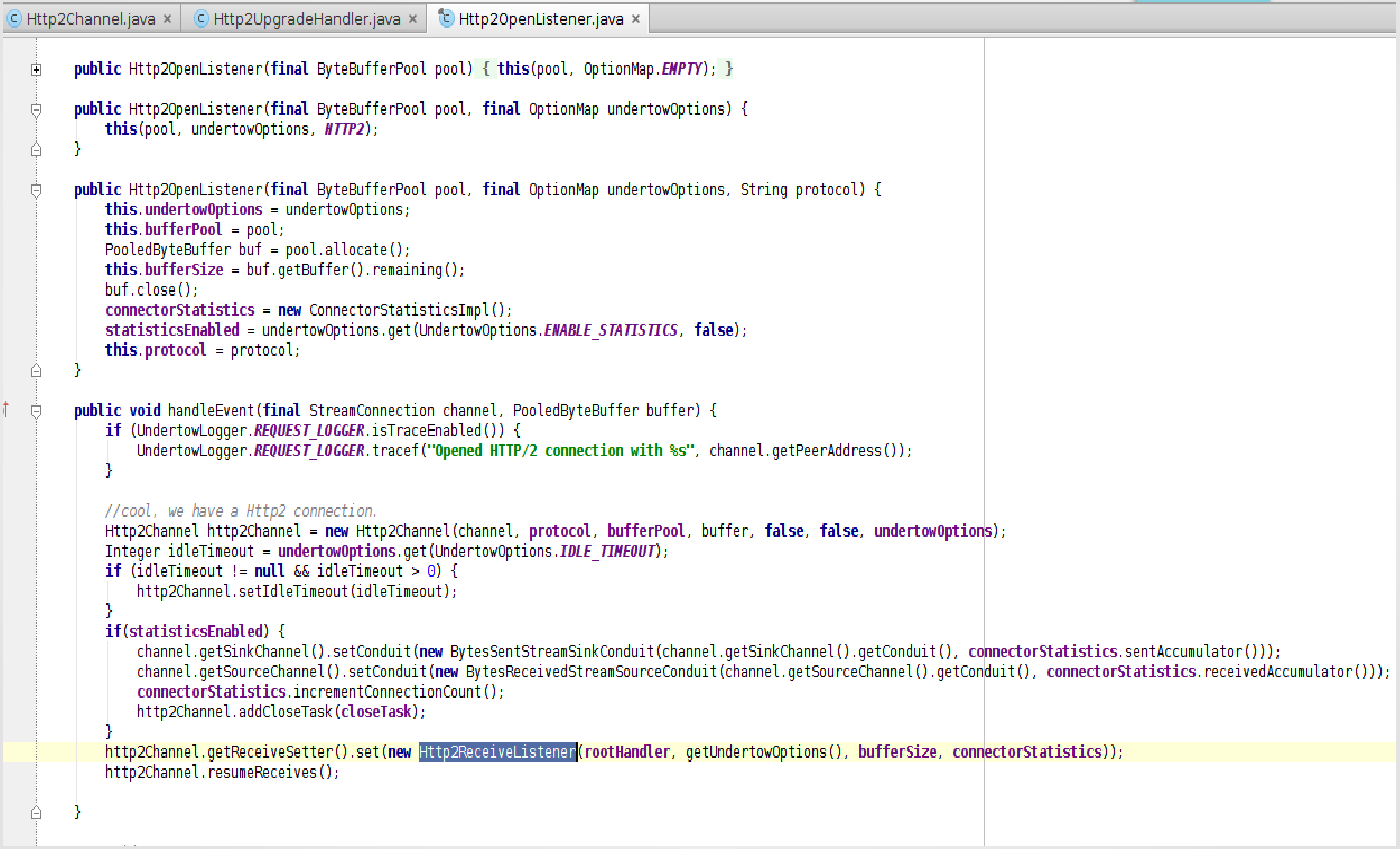
Http2Channel


undertow › core › src › main › java › io › undertow › protocols › http2 › Http2Channel
Project
Http2Channel.java ×
1: Project
7: Structure

```
▸ alpn
▾ http2
    AbstractHttp2StreamSinkChannel
    AbstractHttp2StreamSourceChannel
    ConnectionErrorException
    Hpack
    HpackDecoder
    HpackEncoder
    HpackException
    HPackHuffman
    Http2Channel
    Http2DataFrameParser
    Http2DataStreamSinkChannel
    Http2DiscardParser
    Http2FrameHeaderParser
    Http2FramePriority
    Http2GoAwayParser
    Http2GoAwayStreamSinkChannel
    Http2GoAwayStreamSourceChannel
    Http2HeaderBlockParser
    Http2HeadersParser
    Http2HeadersStreamSinkChannel
    Http2NoDataStreamSinkChannel
    Http2PingParser
    Http2PingStreamSinkChannel
    Http2PingStreamSourceChannel
    Http2PrefaceStreamSinkChannel
    Http2PriorityParser
    Http2PriorityTree
    Http2ProtocolUtils
    Http2PushBackParser
    Http2PushPromiseParser
    Http2PushPromiseStreamSinkChannel
    Http2PushPromiseStreamSourceChannel
    Http2RstStreamParser
    Http2RstStreamSinkChannel
    Http2RstStreamStreamSourceChannel
    Http2Setting
```

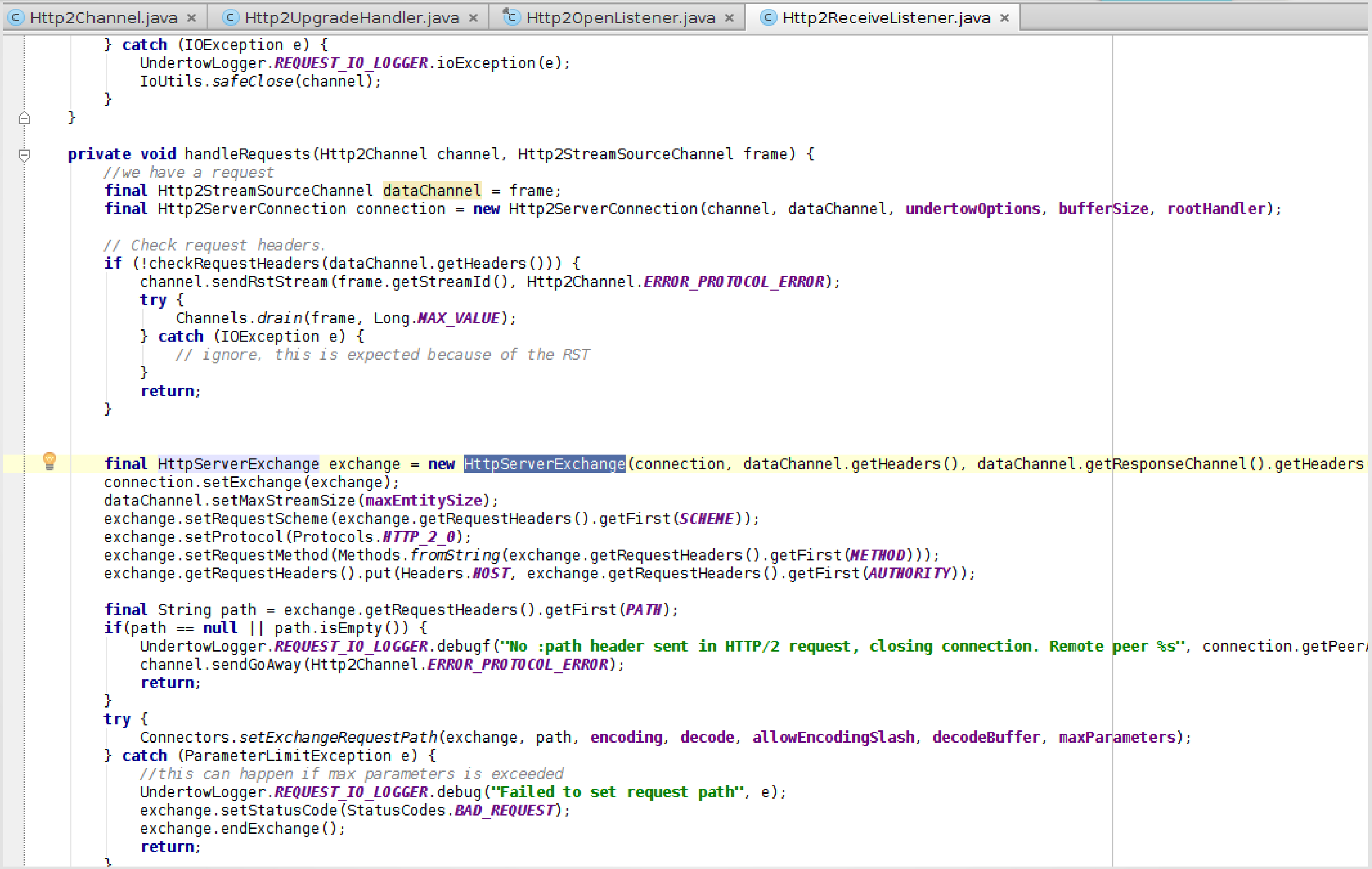
```java
/.../

package io.undertow.protocols.http2;

import ...

/**
 * HTTP2 channel.
 *
 * @author Stuart Douglas

public class Http2Channel extends AbstractFramedChannel<Http2Channel, AbstractHttp2StreamSourceChannel, AbstractHttp2StreamSinkChannel> implements Attachable {

    public static final String CLEARTEXT_UPGRADE_STRING = "h2c";


    public static final HttpString METHOD = new HttpString(":method");
    public static final HttpString PATH = new HttpString(":path");
    public static final HttpString SCHEME = new HttpString(":scheme");
    public static final HttpString AUTHORITY = new HttpString(":authority");
    public static final HttpString STATUS = new HttpString(":status");

    static final int FRAME_TYPE_DATA = 0x00;
    static final int FRAME_TYPE_HEADERS = 0x01;
    static final int FRAME_TYPE_PRIORITY = 0x02;
    static final int FRAME_TYPE_RST_STREAM = 0x03;
    static final int FRAME_TYPE_SETTINGS = 0x04;
    static final int FRAME_TYPE_PUSH_PROMISE = 0x05;
    static final int FRAME_TYPE_PING = 0x06;
    static final int FRAME_TYPE_GOAWAY = 0x07;
    static final int FRAME_TYPE_WINDOW_UPDATE = 0x08;
    static final int FRAME_TYPE_CONTINUATION = 0x09;


    public static final int ERROR_NO_ERROR = 0x00;
    public static final int ERROR_PROTOCOL_ERROR = 0x01;
    public static final int ERROR_INTERNAL_ERROR = 0x02;
    public static final int ERROR_FLOW_CONTROL_ERROR = 0x03;
    public static final int ERROR_SETTINGS_TIMEOUT = 0x04;
    public static final int ERROR_STREAM_CLOSED = 0x05;
    public static final int ERROR_FRAME_SIZE_ERROR = 0x06;
    public static final int ERROR_REFUSED_STREAM = 0x07;
    public static final int ERROR_CANCEL = 0x08;
    public static final int ERROR_COMPRESSION_ERROR = 0x09;
    public static final int ERROR_CONNECT_ERROR = 0x0a;
    public static final int ERROR_ENHANCE_YOUR_CALM = 0x0b;
    public static final int ERROR_INADEQUATE_SECURITY = 0x0c;
```

Http2OpenListener



```java
public Http2OpenListener(final ByteBufferPool pool) { this(pool, OptionMap.EMPTY); }

public Http2OpenListener(final ByteBufferPool pool, final OptionMap undertowOptions) {
    this(pool, undertowOptions, HTTP2);
}

public Http2OpenListener(final ByteBufferPool pool, final OptionMap undertowOptions, String protocol) {
    this.undertowOptions = undertowOptions;
    this.bufferPool = pool;
    PooledByteBuffer buf = pool.allocate();
    this.bufferSize = buf.getBuffer().remaining();
    buf.close();
    connectorStatistics = new ConnectorStatisticsImpl();
    statisticsEnabled = undertowOptions.get(UndertowOptions.ENABLE_STATISTICS, false);
    this.protocol = protocol;
}

public void handleEvent(final StreamConnection channel, PooledByteBuffer buffer) {
    if (UndertowLogger.REQUEST_LOGGER.isTraceEnabled()) {
        UndertowLogger.REQUEST_LOGGER.tracef("Opened HTTP/2 connection with %s", channel.getPeerAddress());
    }

    //cool, we have a Http2 connection.
    Http2Channel http2Channel = new Http2Channel(channel, protocol, bufferPool, buffer, false, false, undertowOptions);
    Integer idleTimeout = undertowOptions.get(UndertowOptions.IDLE_TIMEOUT);
    if (idleTimeout != null && idleTimeout > 0) {
        http2Channel.setIdleTimeout(idleTimeout);
    }
    if(statisticsEnabled) {
        channel.getSinkChannel().setConduit(new BytesSentStreamSinkConduit(channel.getSinkChannel().getConduit(), connectorStatistics.sentAccumulator()));
        channel.getSourceChannel().setConduit(new BytesReceivedStreamSourceConduit(channel.getSourceChannel().getConduit(), connectorStatistics.receivedAccumulator()));
        connectorStatistics.incrementConnectionCount();
        http2Channel.addCloseTask(closeTask);
    }
    http2Channel.getReceiveSetter().set(new Http2ReceiveListener(rootHandler, getUndertowOptions(), bufferSize, connectorStatistics));
    http2Channel.resumeReceives();

}
```

Http2ReceiveListener



Http2Channel.java × Http2UpgradeHandler.java × Http2OpenListener.java × Http2ReceiveListener.java ×

```
        } catch (IOException e) {
            UndertowLogger.REQUEST_IO_LOGGER.ioException(e);
            IoUtils.safeClose(channel);
        }
    }

    private void handleRequests(Http2Channel channel, Http2StreamSourceChannel frame) {
        //we have a request
        final Http2StreamSourceChannel dataChannel = frame;
        final Http2ServerConnection connection = new Http2ServerConnection(channel, dataChannel, undertowOptions, bufferSize, rootHandler);

        // Check request headers.
        if (!checkRequestHeaders(dataChannel.getHeaders())) {
            channel.sendRstStream(frame.getStreamId(), Http2Channel.ERROR_PROTOCOL_ERROR);
            try {
                Channels.drain(frame, Long.MAX_VALUE);
            } catch (IOException e) {
                // ignore, this is expected because of the RST
            }
            return;
        }


        final HttpServerExchange exchange = new HttpServerExchange(connection, dataChannel.getHeaders(), dataChannel.getResponseChannel().getHeaders
        connection.setExchange(exchange);
        dataChannel.setMaxStreamSize(maxEntitySize);
        exchange.setRequestScheme(exchange.getRequestHeaders().getFirst(SCHEME));
        exchange.setProtocol(Protocols.HTTP_2_0);
        exchange.setRequestMethod(Methods.fromString(exchange.getRequestHeaders().getFirst(METHOD)));
        exchange.getRequestHeaders().put(Headers.HOST, exchange.getRequestHeaders().getFirst(AUTHORITY));

        final String path = exchange.getRequestHeaders().getFirst(PATH);
        if(path == null || path.isEmpty()) {
            UndertowLogger.REQUEST_IO_LOGGER.debugf("No :path header sent in HTTP/2 request, closing connection. Remote peer %s", connection.getPeer
            channel.sendGoAway(Http2Channel.ERROR_PROTOCOL_ERROR);
            return;
        }
        try {
            Connectors.setExchangeRequestPath(exchange, path, encoding, decode, allowEncodingSlash, decodeBuffer, maxParameters);
        } catch (ParameterLimitException e) {
            //this can happen if max parameters is exceeded
            UndertowLogger.REQUEST_IO_LOGGER.debug("Failed to set request path", e);
            exchange.setStatusCode(StatusCodes.BAD_REQUEST);
            exchange.endExchange();
            return;
        }
```

总结

- 三大 Java Web 服务器均已经实现 HTTP2 和 Servlet4 草案 API ，可以部署非关键应用时考虑，来提早适应新的技术

可选

- 分析 Netty 对于 HTTP2 协议的实现
- netty/codec-http2/src/main/java/io/netty/handler/codec/http2