超文本传输协议版本 HTTP/2

IETF HTTP2草案(draft-ietf-httpbis-http2-13)

目录

摘要

本规范描述了一种优化的超文本传输协议(HTTP)。HTTP/2通过引进报头字段压缩以及多路复用来更有效利用网络资源、减少感知延迟。另外还介绍了服务器推送规范。

本文档保持对HTTP/1.1的后向兼容，HTTP的现有的语义保持不变。

1.HTTP协议介绍

The Hypertext Transfer Protocol (HTTP) is a wildly successful protocol. However, the HTTP/1.1 message format ([RFC7230], Section 3) was designed to be implemented with the tools at hand in the 1990s, not modern Web application performance. As such it has several characteristics that have a negative overall effect on application performance today.

超文本传输协议（HTTP）是一个非常成功的协议。 但是HTTP/1.1
是针对90年代的情况而不是现代web应用的性能而设计的，导致它的一些特点已经对现代应用程序的性能产生负面影响。

In particular, HTTP/1.0 only allows one request to be outstanding at a time on a given connection. HTTP/1.1 pipelining only partially addressed request concurrency and suffers from head-of-line blocking. Therefore, clients that need to make many requests typically use multiple connections to a server in order to reduce latency.

特别是，HTTP/1.0只允许在一个连接上建立一个当前未完成的请求。HTTP/1.1管道只部分处理了请求并发和报头阻塞的问题。因此客户端需要发起多次请求通过数次连接服务器来减少延迟。

Furthermore, HTTP/1.1 header fields are often repetitive and verbose, which, in addition to generating more or larger network packets, can cause the small initial TCP [TCP] congestion window to quickly fill. This can result in excessive latency when multiple requests are made on a single new TCP connection.

此外，HTTP/1.1的报头字段经常重复和冗长。在产生更多或更大的网络数据包时，可能导致小的初始TCP堵塞窗口被快速填充。这可能在多个请求建立在一个新的TCP连接时导致过度的延迟。

This specification addresses these issues by defining an optimized mapping of HTTP's semantics to an underlying connection. Specifically, it allows interleaving of request and response messages on the same connection and uses an

efficient coding for HTTP header fields. It also allows prioritization of requests, letting more important requests complete more quickly, further improving performance.

本协议通过定义一个优化的基础连接的HTTP语义映射来解决这些问题。
具体地，它允许在同一连接上交错地建立请求和响应消息，并使用高效率编码的HTTP报头字段。 它还允许请求的优先级，让更多的重要的请求更快速的完成，进一步提升了性能。

The resulting protocol is designed to be more friendly to the network, because fewer TCP connections can be used in comparison to HTTP/1.x. This means less competition with other flows, and longer-lived connections, which in turn leads to better utilization of available network capacity.

最终协议设计为对网络更友好，因为它相对HTTP/1.x减少了TCP连接。
这意味着与其他流更少的竞争以及更长时间的连接,从而更有效地利用可用的网络容量。

Finally, this encapsulation also enables more efficient processing of messages through use of binary message framing.

最后，这种封装也通过使用二进制消息帧使信息处理更具扩展性。

2.HTTP/2协议概述

HTTP/2 provides an optimized transport for HTTP semantics. HTTP/2 supports all of the core features of HTTP/1.1, but aims to be more efficient in several ways.

HTTP/2
提供了HTTP语义的传输优化。HTTP/2支持所有HTTP / 1.1的核心特征，并且在不同的方面做的更高效。

The basic protocol unit in HTTP/2 is a frame (Section 4.1). Each frame type serves a different purpose. For example, HEADERS and DATA frames form the basis of HTTP requests and responses (Section 8.1); other frame types like SETTINGS, WINDOW_UPDATE, and PUSH_PROMISE are used in support of other HTTP/2 features.

HTTP/2中基本的协议单位是帧。每个帧都有不同的类型和用途。例如，报头(HEADERS)和数据(DATA)帧组成了基本的HTTP 请求和响应；其他帧例如
设置(SETTINGS),窗口更新(WINDOW_UPDATE),
和推送承诺(PUSH_PROMISE)是用来实现HTTP/2的其他功能。

Multiplexing of requests is achieved by having each HTTP request-response exchanged assigned to a single stream (Section 5). Streams are largely independent of each other, so a blocked or stalled request does not prevent progress on other requests.

请求多路复用是通过在一个流上分配多个HTTP请求响应交换来实现的(章节5)。流在很大程度上是相互独立的，因此一个请求上的阻塞或终止并不会影响其他请求的处理。

Flow control and prioritization ensure that it is possible to properly use multiplexed streams. Flow control (Section 5.2) helps to ensure that only data that can be used by a receiver is transmitted. Prioritization (Section 5.3) ensures that limited resources can be directed to the most important requests first.

流量控制和优先级能确保正确使用复用流。流量控制(章节5.2)有助于确保只传播接受者需要使用的数据数据。优先级(章节5.3)能确保有限的资源能优先被重要的请求使用。

HTTP/2 adds a new interaction mode, whereby a server can push responses to a client (Section 8.2). Server push allows a server to speculatively send a client data that the server anticipates the client will need, trading off some network usage against a potential latency gain. The server does this by synthesizing a request, which it sends as a PUSH_PROMISE frame. The server is then able to send a response to the synthetic request on a separate stream.

HTTP/2添加了一种新的交互模式，即服务器能推送消息给客户端。服务器推送允许服务端预测客户端需
要来发送数据给客户端，交换网络的使用来阻止潜在的延迟增长。服务器通过复用一个以PUSH_PROMISE帧发送的请求来实现推送，然后服务端可以在一
个单独的流里面发送响应给这个合成的请求。

Frames that contain HTTP header fields are compressed (Section 4.3). HTTP requests can be highly redundant, so compression can reduce the size of requests and responses significantly.

帧包含的HTTP报头字段是压缩的。HTTP请求有可能是高度冗余的，因此压缩能显著减少请求和响应的大小。

2.1文档结构

The HTTP/2 specification is split into four parts:

HTTP/2协议被分为以下四个部分：

- Starting HTTP/2 (Section 3) covers how an HTTP/2 connection is initiated.
- The framing (Section 4) and streams (Section 5) layers describe the way HTTP/2 frames are structured and formed into multiplexed streams.
- Frame (Section 6) and error (Section 7) definitions include details of the frame and error types used in HTTP/2.

- HTTP mappings (Section 8) and additional requirements (Section 9) describe how HTTP semantics are expressed using frames and streams.
- 启动HTTP/2(章节3)包含了一个HTTP/2连接是如何初始化的。
- 帧(章节4)和流层(章节5)描述了 HTTP/2流的结构以及如何形成复用流的。
- 帧(章节6)和错误码(章节7)定义了HTTP/2中使用的流和错误类型的详细内容。
- HTTP寻址(章节8)和拓展需求(章节9)描述了HTTP语义化是如何由帧和流表达的。

While some of the frame and stream layer concepts are isolated from HTTP, the intent is not to define a completely generic framing layer. The framing and streams layers are tailored to the needs of the HTTP protocol and server push.

一些帧和流层的概念是与HTTP隔离的，因为意图并不是定义一个完全通用的帧层。这些帧和流层是为了HTTP协议和服务端推送的需求定制的。

2.2约定和术语

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

文档中出现的关键字“必须”，“绝对不能”，“要求”，“应”，“不应”，“应该”，“不应该”，“建议”，“可以”及“可选”可通过在 RFC 2119
的解释进行理解【RFC2119】

All numeric values are in network byte order. Values are unsigned unless otherwise indicated. Literal values are provided in decimal or hexadecimal as appropriate. Hexadecimal literals are prefixed with 0x to distinguish them from decimal literals.

所有的数值都是按网络字节顺序。除非有另外说明，数值是无符号的。按情况提供十进制或十六进制的文本值。十六进制用前缀0x来区分。

The following terms are used:

- client: The endpoint initiating the HTTP/2 connection. connection:A transport-level connection between two endpoints.
- connection error: An error that affects the entire HTTP/2 connection.
- endpoint: Either the client or server of the connection.
- frame: The smallest unit of communication within an HTTP/2 connection, consisting of a header and a variable-length sequence of bytes structured according to the frame type.
- peer: An endpoint. When discussing a particular endpoint, "peer" refers to the endpoint that is remote to the primary subject of discussion.
- receiver: An endpoint that is receiving frames.

- sender: An endpoint that is transmitting frames.
- server: The endpoint which did not initiate the HTTP/2 connection.
- stream: A bi-directional flow of frames across a virtual channel within the HTTP/2 connection.
- stream error: An error on the individual HTTP/2 stream.

文中术语包括：

- 客户端：发起HTTP/2请求的端点
- 连接：在两个端点之间的传输层级别的连接
- 连接错误：整个HTTP/2连接过程中发生的错误
- 端点：连接的客户端或服务器
- 帧：HTTP/2.0通信连接中的最小单元，包括根据帧类型结构的字节的报头和可变长度的序列
- 对等端：一个端点。当讨论特定的端点时，“对等端”指的是讨论的主题的远程端点
- 接收端：正在接收帧的端点
- 发送端：正在传输帧的端点
- 服务端：不是启动HTTP/2连接的端点
- 流：一个双向字节帧流穿过HTTP/2连接中的虚拟通道
- 流错误：一个HTTP/2流中的错误

3.启动HTTP/2

An HTTP/2 connection is an application level protocol running on top of a TCP connection ([TCP]). The client is the TCP connection initiator.

一个HTTP/2连接是运行在TCP连接上的应用层协议。客户端是TCP连接的发起者。

HTTP/2 uses the same "http" and "https" URI schemes used by HTTP/1.1. HTTP/2 shares the same default port numbers: 80 for "http" URIs and 443 for "https" URIs. As a result, implementations processing requests for target resource URIs like http://example.org/foo or https://example.com/bar are required to first discover whether the upstream server (the immediate peer to which the client wishes to establish a connection) supports HTTP/2.

HTTP/2使用与HTTP/1.1相同的"http"和"https"
资源标识符(URI)。使用相同的默认端口："http"
的80端口及“https”的443端口。因此，实现对例如http://example.org/foo或https://example.com/bar目标资源的URI请求处理需要首先确定上游服务端(当前客户端希望建立连接的对等端)是否支持HTTP/2。

The means by which support for HTTP/2 is determined is different for "http" and "https" URIs. Discovery for "http" URIs is described in Section 3.2. Discovery for "https" URIs is described in Section 3.3.

这意味着检测“http” 及“https”
的URIs是否支持HTTP/2的方法是不一样的。检测"http"URIs在章节3.2中描述。检测"https" URIs 在章节3.3中描述。

3.1 HTTP/2版本定义

The protocol defined in this document has two identifiers.

- The string "h2" identifies the protocol where HTTP/2 uses TLS [TLS12]. This identifier is used in the TLS application layer protocol negotiation extension (ALPN) [TLSALPN] field and any place that HTTP/2 over TLS is identified.

  The "h2" string is serialized into an ALPN protocol identifier as the two octet sequence: 0x68, 0x32.

- The string "h2c" identifies the protocol where HTTP/2 is run over cleartext TCP. This identifier is used in the HTTP/1.1 Upgrade header field and any place that HTTP/2 over TCP is identified.

在本文档中定义的协议有两个标识符。

- 字 符"h2"表示HTTP/2协议使用TLS[TLS]。这种方式用在HTTP/1.1的升级字段、TLS应用层协议协商扩展字段以及其他需要定义协议的地方。当在定义ALPN协议(序列化的字节)中序列化时。 "h2"字符序列化到 ALPN
  协议中变成两个字节序列：0x68，0x32。
- 字符"h2c" 表示HTTP/2协议运行在明文TCP上。这个标识用在HTTP/1.1
  升级报头字段以及任何TCP是确定的地方。

Negotiating "h2" or "h2c" implies the use of the transport, security, framing and message semantics described in this document.

用到"h2" 或者 "h2c" 表明使用文档中定义的传输、安全、帧及语义化消息。

Only implementations of the final, published RFC can identify themselves as "h2" or "h2c". Until such an RFC exists, implementations MUST NOT identify themselves using these strings.

只有依据最终发表的RFC的实现能使用"h2"或"h2c"进行标明。在此之前，任何实现绝对不能使用这两个字符进行识别。

Examples and text throughout the rest of this document use "h2" as a matter of editorial convenience only. Implementations of draft versions MUST NOT identify using this string.

本文档内的例子或者文本只作为编辑方便使用"h2"。针对草案版本的实现绝对不能使用这个字符进行识别。

Implementations of draft versions of the protocol MUST add the string "-" and the corresponding draft number to the identifier. For example, draft-ietf-httpbis-http2-11 over TLS is identified using the string "h2-11".

依据草案版本实现的协议必须添加字符"-"及相对应的草案版本进行标识。例如，基于TLS的草案draft-ietf-httpbis-http2-11 需要使用字符"h2-11"进行标识。

Non-compatible experiments that are based on these draft versions MUST append the string "-" and an experiment name to the identifier. For example, an experimental implementation of packet mood-based encoding based on draft-ietf-httpbis-http2-09 might identify itself as "h2-09-emo". Note that any label MUST conform to the "token" syntax defined in Section 3.2.6 of [RFC7230]. Experimenters are encouraged to coordinate their experiments on the ietf-http-wg@w3.org mailing list.

基 于这些草案版本的不兼容的实验必须在标识符中添加字符"-"及实验名称。例如，基于draft-ietf-httpbis-http2-09草案的情绪编码实验实现必须使用类似"h2-09-emo"的标识符。需要注意的是任何标签必须符合[RFC7230]章节3.2.6定义的"token"语法。鼓励实验者提交实验到ietf-http-wg@w3.org 的邮件列表中。

3.2 Starting HTTP/2 for "http" URIs 针对"http"启动HTTP/2

A client that makes a request to an "http" URI without prior knowledge about support for HTTP/2 uses the HTTP Upgrade mechanism (Section 6.7 of [RFC7230]). The client makes an HTTP/1.1 request that includes an Upgrade header field identifying HTTP/2 with the "h2c" token. The HTTP/1.1 request MUST include exactly one HTTP2-Settings (Section 3.2.1) header field.

客户端无法预知服务端是否支持HTTP/2.0 的情况下使用HTTP升级机制发起“http” URI请求([RFC7230]章节6.7)。客户端发起一个http1.1请求，其中包含识别HTTP/2的升级报头字段与h2c token。HTTP/1.1必须包含一个确切的HTTP2-Settings中的报头字段。

例如：

GET /default.htm HTTP/1.1

```
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

Requests that contain an entity body MUST be sent in their entirety before the client can send HTTP/2 frames. This means that a large request entity can block the use of the connection until it is completely sent.

包含主体内容的请求必须在客户端能发送HTTP/2帧前全部发送。这意味着一个大的请求实体能阻塞连接的使用直到其全部被发送。

If concurrency of an initial request with subsequent requests is important, a small request can be used to perform the upgrade to HTTP/2, at the cost of an additional round-trip.

如果一个请求的并发后续请求是重要的，那么可以使用一个小的请求来执行升级到HTTP/2的操作,这样仅消耗一个额外的往返成本。

A server that does not support HTTP/2 can respond to the request as though the Upgrade header field were absent:

不支持HTTP/2的服务端对请求返回一个不包含升级的报头字段的响应：

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
...
```

A server MUST ignore a "h2" token in an Upgrade header field. Presence of a token with "h2" implies HTTP/2 over TLS, which is instead negotiated as described in Section 3.3.

服务端必须忽略升级报头字段中的“h2” token。“h2” token基于TLS实现的HTTP/2,协商方法在章节3.3中定义。

A server that supports HTTP/2 can accept the upgrade with a 101 (Switching Protocols) response. After the empty line that terminates the 101 response, the server can begin sending HTTP/2 frames. These frames MUST include a response to the request that initiated the Upgrade.

支持HTTP/2的服务端可以返回一个101(转换协议)响应来接受升级请求。在101空内容响应终止后，服务端可以开始发送HTTP/2帧。这些帧必须包含一个发起升级的请求的响应。

```
HTTP/1.1 101 Switching Protocols
```

Connection: Upgrade
Upgrade: h2c

[HTTP/2 connection ...

The first HTTP/2 frame sent by the server is a SETTINGS frame (Section 6.5). Upon receiving the 101 response, the client sends a connection preface (Section 3.5), which includes a SETTINGS frame.

第一个被服务端发送的HTTP/2帧是一个设置(SETTINGS)帧。在收到101响应后，客户端发送一个包含设置(SETTINGS)帧的连接序言。

The HTTP/1.1 request that is sent prior to upgrade is assigned stream identifier 1 and is assigned default priority values (Section 5.3.5). Stream 1 is implicitly half closed from the client toward the server, since the request is completed as an HTTP/1.1 request. After commencing the HTTP/2 connection, stream 1 is used for the response.

（*** 升级前所发送的HTTP/1.1请求被派送到标示流1并将赋予最高优先级。）
HTTP/1.1最开始用来升级到2.0的请求用1来标示流并将赋予最高优先级。1流对发送到服务端的客户端是隐式半封闭的，因为这个请求已经作为
HTTP/1.1请求完成了。HTTP/2连接开始后，1流在响应中使用。

3.2.1 HTTP2-Settings Header Field HTTP2-Setting报头字段

A request that upgrades from HTTP/1.1 to HTTP/2 MUST include exactly one HTTP2-Settings header field. The HTTP2-Settings header field is a hop-by-hop header field that includes parameters that govern the HTTP/2 connection, provided in anticipation of the server accepting the request to upgrade.

从HTTP/1.1升级到HTTP/2的请求必须包含一个确切的HTTP2-Settings报头字段。HTTP2-Settings
的报头字段是逐跳报头字段，它包含管理HTTP/2连接参数。这是从对于服务端接受升级请求的预测中所获取的。

HTTP2-Settings   = token68

A server MUST reject an attempt to upgrade if this header field is not present. A server MUST NOT send this header field.

服务端未检测到此报头字段必须拒绝客户端的升级尝试。服务端绝对不能发送此报头字段。

The content of the HTTP2-Settings header field is the payload of a SETTINGS frame (Section 6.5), encoded as a base64url string (that is, the URL- and filename-safe Base64 encoding described in Section 5 of [RFC4648], with any

trailing '=' characters omitted). The ABNF [RFC5234] production for token68 is defined in Section 2.1 of [RFC7235].

HTTP2-
Settings报头字段的内容是设置(SETTINGS)帧的有效载体，使用base64url字符编码(URL及文件名安全的Base64编码，编码 描述在[RFC4648] 章节5中，忽略任何“=”字符。)
ABNF[RFC5234]产品中对token68的定义在[RFC7235] 章节2.1中。

As a hop-by-hop header field, the Connection header field MUST include a value of HTTP2-Settings in addition to Upgrade when upgrading to HTTP/2.

作为一个逐跳的报文报头字段，当升级到HTTP/2时，此连接报头字段必须包含一个HTTP2设置(HTTP2-Settings)的值来完成升级操作。

A server decodes and interprets these values as it would any other SETTINGS frame. Acknowledgement of the SETTINGS parameters (Section 6.5.3) is not necessary, since a 101 response serves as implicit acknowledgment. Providing these values in the Upgrade request ensures that the protocol does not require default values for the above SETTINGS parameters, and gives a client an opportunity to provide other parameters prior to receiving any frames from the server.

服务端就像对任何其他设置(SETTINGS)帧一样对这些值进行解码和解释。因为101响应的隐式声明，对这些设置参数的确认不是必须的。这些升级请求中的值使得协议不需要上述设置参数的默认值，同时使客户端有机会在从服务端接受任何帧之前提供其他参数。

3.3 Starting HTTP/2 for "https" URIs 针对“https”启动HTTP/2

A client that makes a request to an "https" URI without prior knowledge about support for HTTP/2 uses TLS [TLS12] with the application layer protocol negotiation extension [TLSALPN].

客户端在不了解服务端是否支持HTTP/2的时候，会使用TSL [TLS12]
于其应用层协议协商扩展 [TLSALPN]。

HTTP/2 over TLS uses the "h2" application token. The "h2c" token MUST NOT be sent by a client or selected by a server.

使用TLS的HTTP/2
使用"h2"程序token。“h2c”token绝对不能由客户端或者选定的服务端发送。

Once TLS negotiation is complete, both the client and the server send a connection preface (Section 3.5).

TSL协议一旦完成，客户端和服务端都可以发送连接序言(章节3.5)。

3.4 Starting HTTP/2 with Prior Knowledge 先验下启动HTTP/2

A client can learn that a particular server supports HTTP/2 by other means. For example, [ALT-SVC] describes a mechanism for advertising this capability.

客户端可以通过其他方式判断服务端是否支持HTTP/2。例如，AltSvc定义一种机制让HTTP头字段进行广播。

A client MAY immediately send HTTP/2 frames to a server that is known to support HTTP/2, after the connection preface (Section 3.5). A server can identify such a connection by the use of the "PRI" method in the connection preface. This only affects the establishment of HTTP/2 connections over cleartext TCP; implementations that support HTTP/2 over TLS MUST use protocol negotiation in TLS [TLSALPN].

客户端可以对支持
HTTP/2的服务端在连接序言(章节3.5)之后立即发送HTTP/2帧。服务端可以通过连接序言中的“PRI”方法来区分这种连接。这只对基于明文
TCP的HTTP/2连接建立有影响；支持HTTP/2的服务端对“https”URI需要支持TLS中的协商扩展。

Prior support for HTTP/2 is not a strong signal that a given server will support HTTP/2 for future connections. It is possible for server configurations to change; for configurations to differ between instances in clustered server; or network conditions to change.

对HTTP/2之前的支持并不表明一个给定的服务器会在以后的连接中一定支持HTTP/2。服务器配置有可能改变或者集群中不同服务器配置有差异。拦截代理(又叫“透明”代理)是另一个可能得原因。

3.5 HTTP/2 Connection Preface HTTP/2连接序言

Upon establishment of a TCP connection and determination that HTTP/2 will be used by both peers, each endpoint MUST send a connection preface as a final confirmation and to establish the initial SETTINGS parameters for the HTTP/2 connection.

在建立TCP连接并且检测到HTTP/2会被各个对等端使用后，每个端点必须发送一个连接序言最终确认并作为建立HTTP/2连接的初始设置参数。

The client connection preface starts with a sequence of 24 octets, which in hex notation are:

客户端连接序言以24个字节的序列开始，以十六进制表示是：

0x505249202a20485454502f322e300d0a0d0a534d0d0a0d0a

(the string PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n). This sequence is followed by a SETTINGS frame (Section 6.5). The SETTINGS frame MAY be empty. The client sends the client connection preface immediately upon receipt of a 101 Switching Protocols response (indicating a successful upgrade), or as the first application data octets of a TLS connection. If starting an HTTP/2 connection with prior knowledge of server support for the protocol, the client connection preface is sent upon connection establishment.

(字符串PRI *
HTTP/2.0\r\n\r\nSM\r\n\r\n)。这个序列后跟着一个设置帧，其可为空帧。客户端在收到101转换协议响应(升级成功指示)后马
上发送客户端连接序言，或者作为TLS连接的第一个应用数据字节。如果在预先知道服务器支持HTTP/2的情况下启动HTTP/2连接，客户端连接序言在 连接建立后发送。

The client connection preface is selected so that a large proportion of HTTP/1.1 or HTTP/1.0 servers and intermediaries do not attempt to process further frames. Note that this does not address the concerns raised in [TALKING].

客户端连接序言是用来让大部分的HTTP/1.1或者HTTP/1.0服务端以及中介端不试图进一步处理帧。注意这并不能处理【讨论】中提到的问题。

The server connection preface consists of a potentially empty SETTINGS frame (Section 6.5) that MUST be the first frame the server sends in the HTTP/2 connection.

服务端连接序言包含一个有可能是空的设置（SETTING）帧（章节6.5），它必须在HTTP/2连接中首个发送。

To avoid unnecessary latency, clients are permitted to send additional frames to the server immediately after sending the client connection preface, without waiting to receive the server connection preface. It is important to note, however, that the server connection preface SETTINGS frame might include parameters that necessarily alter how a client is expected to communicate with the server. Upon receiving the SETTINGS frame, the client is expected to honor any parameters established.

为
了避免不必要的延迟，允许客户端在发送客户端连接序言之后立即发送其他额外的帧，不需

要等待收到服务端连接序言。不过需要注意的是，服务端连接序言设置(SETTINGS)帧可能包含一些关于期望客户端如何与服务端通信的所必须修改的参数。在收到这些设置(SETTINGS)帧之后，客户端应当遵守所有 设置的参数。

Clients and servers MUST terminate the TCP connection if either peer does not begin with a valid connection preface. A GOAWAY frame (Section 6.8) can be omitted if it is clear that the peer is not using HTTP/2.

如果任何一个端点没有以一个有效的连接序言开头，客户端和服务端必须终止TCP连接。如果端点并没有使用HTTP/2此时可以省略超时(GOAWAY)帧(章节6.8)。

4 HTTP Frames HTTP帧

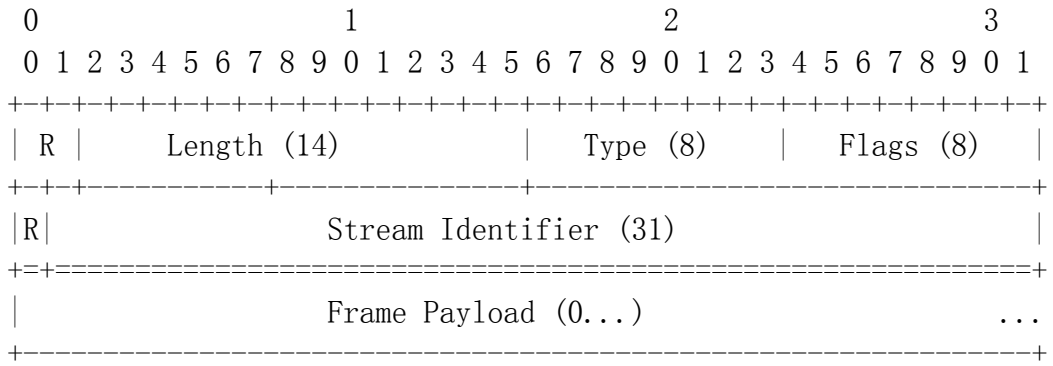
Once the HTTP/2 connection is established, endpoints can begin exchanging frames.

HTTP/2连接一旦建立，端点之间可以马上交换数据帧。

4.1 Frame Format 帧格式

All frames begin with a fixed 8-octet header followed by a payload of between 0 and 16,383 octets.

所有的帧以8字节的报头开始并且跟着0-16,383字节长度的主体。



The fields of the frame header are defined as:

- R :A reserved 2-bit field. The semantics of these bits are undefined and the bits MUST remain unset (0) when sending and MUST be ignored when receiving.
- Length : The length of the frame payload expressed as an unsigned 14-bit integer. The 8 octets of the frame header are not included in this value.

- Type : The 8-bit type of the frame. The frame type determines the format and semantics of the frame. Implementations MUST ignore and discard any frame that has a type that is unknown.
- Flags :An 8-bit field reserved for frame-type specific boolean flags. Flags are assigned semantics specific to the indicated frame type. Flags that have no defined semantics for a particular frame type MUST be ignored, and MUST be left unset (0) when sending.
- R: A reserved 1-bit field. The semantics of this bit are undefined and the bit MUST remain unset (0) when sending and MUST be ignored when receiving.
- Stream Identifier: A 31-bit stream identifier (see Section 5.1.1). The value 0 is reserved for frames that are associated with the connection as a whole as opposed to an individual stream.

帧报头字段定义是：

- R ：
保留的2位字段。这些字节的语义是未定义的，并且在发送的时候必须保持未设置(0)，接收的时候必须被忽略此字段。
- Length ： 14位无符号整数的帧主体长度。8字节长度的帧报头信息不计算在此内。
- Type ：
帧的8位类型。帧类型定义了剩余的帧报头和帧主体将如何被解释。具体实现必须在收到未知帧类型(任何未在文档中定义的帧)时作为连接错误中的类型协议错误(PROTOCOL_ERROR)处理。
- Flags ： 为帧类型保留的8字节字段有具体的布尔标识。
标识针对确定的帧类型赋予特定的语义。确定帧类型定义语义以外的标示必须被忽略，并且必须在发送的时候保留未设置(0)。
- R ：
1位的保留字段。这个字段的语义未设置并且必须在发送的时候保持未设置(0),在接受的时候必须被忽略。
- Stream Identifier ：
31字节的流标识符(见StreamIdentifiers)。0是保留的，标明帧是与连接相关作为一个整体而不是一个单独的流。

The structure and content of the frame payload is dependent entirely on the frame type.

帧主体的结构和内容完全取决于帧类型。

4.2 Frame Size 帧大小

The maximum size of a frame payload varies by frame type. The absolute maximum size of a frame payload is 214-1 (16,383) octets, meaning that the maximum frame

size is 16,391 octets. All implementations MUST be capable of receiving and minimally processing frames up to this maximum size.

帧主体的最大长度限制因不同的帧类型而不同。最大帧主体的绝对长度是 $2^{14}-1$ (16,383)字节，表示最大的帧长度是16,391字节。所有的实现必须具备接收和处理此最大长度帧的能力。

Certain frame types, such as PING (Section 6.7), impose additional limits on the amount of payload data allowed.

某些帧类型，例如PING(章节6.7)，对主体数据大小有额外的限制。同样的，一些特定的应用也可能使用额外的大小限制(见HttpExtra)。

If a frame size exceeds any defined limit, or is too small to contain mandatory frame data, the endpoint MUST send a FRAME_SIZE_ERROR error. A frame size error in a frame that could alter the state of the entire connection MUST be treated as a connection error (Section 5.4.1); this includes any frame carrying a header block (Section 4.3) (that is, HEADERS, PUSH_PROMISE, and CONTINUATION), SETTINGS, and any WINDOW_UPDATE frame with a stream identifier of 0.

如果一个帧大小超过设定的限制，
或者太小无法包含必须的基础帧数据，这个端点必须发送一个帧大小错误(FRAME_SIZE_ERROR)。如果帧大小错误可能修改整个连接状态，必须作
为一个连接错误(章节5.4.1)处理；这包括与0流一起的携带报头区块(即报文头(HEADERS)，推送承诺(PUSH_PROMISE)和延续
(CONTINUATION)帧)、设置(SETTINGS)以及任何窗口更新(WINDOW_UPDATE)帧。

4.3 Header Compression and Decompression 报头压缩和解压缩

A header field in HTTP/2 is a name with one or more associated values. They are used within HTTP request and response messages as well as server push operations (see Section 8.2).

HTTP/2报文报头字段是包含一个或多个相关的键值对。他们在HTTP请求响应消息及服务器推送操作(见章节8.2)中使用。

Header sets are collections of zero or more header fields. When transmitted over a connection, a header set is serialized into a header block using HTTP Header Compression [COMPRESSION]. The serialized header block is then divided into one or more octet sequences, called header block fragments, and transmitted within the payload of HEADERS (Section 6.2), PUSH_PROMISE (Section 6.6) or CONTINUATION (Section 6.10) frames.

报头集合是0个或多个报头字段的集合。当他通过连接传输的时候，报头集合将使用HTTP报头压缩序列化到报文报头块中。序列化的报头块被分割成一个或多个的字节序列，称为报头分区，并在报头、推送承诺及延续帧的载体中传送。

HTTP Header Compression does not preserve the relative ordering of header fields. Header fields with multiple values are encoded into a single header field using a special delimiter (see Section 8.1.2.3), this preserves the relative order of values for that header field.

HTTP报文头压缩并不保留报头字段的相关顺序。具有多个值的报头字段使用特定的分割器被编码分割到一个单独的报头区域（见 章节8.1.2.3
HeaderOrdering），这保留了该报头字段中各种值的对应顺序。

The Cookie header field [COOKIE] is treated specially by the HTTP mapping (see Section 8.1.2.4).

报文头Cookie字段被通过HTTP映射特殊处理；见章节8.1.2.4。

A receiving endpoint reassembles the header block by concatenating its fragments, then decompresses the block to reconstruct the header set.

A complete header block consists of either:

- a single HEADERS or PUSH_PROMISE frame, with the END_HEADERS flag set, or
- a HEADERS or PUSH_PROMISE frame with the END_HEADERS flag cleared and one or more CONTINUATION frames, where the last CONTINUATION frame has the END_HEADERS flag set.

接收端点连接报头区块重新组装，并且解压缩区块后重建报头集合。

一个完整的报头区块包含：

- 一个包含报头终止标记集合的单独的报头HEADERS 或
  推送承诺PUSH_PROMISE帧，或者
- 一个报头终止标记被清除的报头HEADERS 或
  推送承诺PUSH_PROMISE帧以及一个或多个延续CONTINUATION帧，最后一个延续CONTINUATION帧拥有报头终止标记设置。

Header compression is stateful, using a single compression context for the entire connection. Each header block is processed as a discrete unit. Header blocks MUST be transmitted as a contiguous sequence of frames, with no interleaved frames of any other type or from any other stream. The last frame in a sequence of HEADERS or CONTINUATION frames MUST have the END_HEADERS flag set. The last frame in a sequence of PUSH_PROMISE or CONTINUATION frames MUST have

the END_HEADERS flag set. This allows a header block to be logically equivalent to a single frame.

报头压缩是有
状态的并且在整个连接过程中使用同个压缩环境。每个报头区块作为离散的单元处理。报头区块必须作为一个连续的帧序列传输，没有任何类型或任何其他流的交错
帧。一个报头HEADERS或者延续CONTINUATION帧序列的最后一帧必须有报头终止标记设置。
推送承诺PUSH_PROMISE或者延续
CONTINUATION帧序列的最后一帧必须具有报头终止标记设置。

Header block fragments can only be sent as the payload of HEADERS, PUSH_PROMISE or CONTINUATION frames, because these frames carry data that can modify the compression context maintained by a receiver. An endpoint receiving HEADERS, PUSH_PROMISE or CONTINUATION frames MUST reassemble header blocks and perform decompression even if the frames are to be discarded. A receiver MUST terminate the connection with a connection error (Section 5.4.1) of type COMPRESSION_ERROR if it does not decompress a header block.

报头区块必
须被报头HEADERS、推送承诺PUSH_PROMISE或延续CONTINUATION的有效载体发送，因为这些帧中携带了能被接收端修改的压缩上下
文数据。端点在接收报头HEADERS、推送承诺PUSH_PROMISE或延续CONTINUATION帧时必须重新组装报头区块并且执行解压缩，即便
这些帧将被废弃。如何不能重建报头区间，接收端必须终止连接并报类型为解压缩错误的连接错误(章节5.4.1)。

5.Streams and Multiplexing 流和多路复用

A "stream" is an independent, bi-directional sequence of frames exchanged between the client and server within an HTTP/2 connection. Streams have several important characteristics:

- A single HTTP/2 connection can contain multiple concurrently open streams, with either endpoint interleaving frames from multiple streams.
- Streams can be established and used unilaterally or shared by either the client or server.
- Streams can be closed by either endpoint.
- The order in which frames are sent on a stream is significant. Recipients process frames in the order they are received. In particular, the order of HEADERS, and DATA frames is semantically significant.
- Streams are identified by an integer. Stream identifiers are assigned to streams by the endpoint initiating the stream.

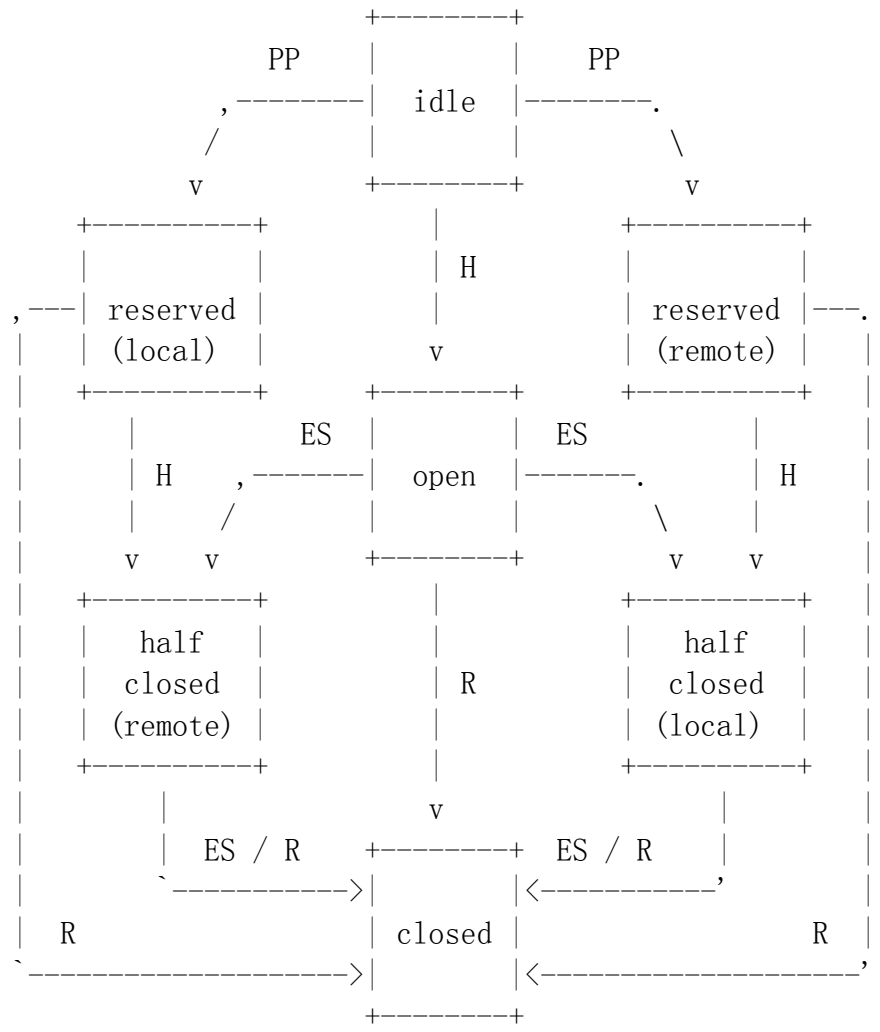
流是一个独立的，客户端和服务端在HTTP/2连接下交换帧的双向序列。流有一下几个重要特点：

- 一个单独的HTTP/2连接能够保持多个同时打开的流，各个端点间从多个流中交换帧。
- 流可以被被客户端或者服务端单方面建立使用或分享。
- 流可以被任何一个连接终端关闭。
- 在流内发送帧的顺序很重要。它们将按被接收的顺序处理。特别是报头及数据帧的顺序语义上是有意义的。
- 流以一个整数标识。标识符有启动流的终端分配。

5.1 Stream States 流状态

The lifecycle of a stream is shown in Figure 1.

流的生存周期如StreamStatesFigure所示：



PP: PUSH_PROMISE frame (with implied CONTINUATIONs)
ES: END_STREAM flag
R: RST_STREAM frame

$$Figure 2: Stream States$$

Note that this diagram shows stream state transitions and frames that affect those transitions only. In this regard, CONTINUATION frames do not result in state transitions and are effectively part of the HEADERS or PUSH_PROMISE that they follow.

请注意该图仅展示了流状态的转换和帧对这些转换的影响。在这方面，延续帧不会影响流状态的转换，但是对后面跟随的报头或者推送承诺是有影响的。

Both endpoints have a subjective view of the state of a stream that could be different when frames are in transit. Endpoints do not coordinate the creation of streams; they are created unilaterally by either endpoint. The negative consequences of a mismatch in states are limited to the "closed" state after sending RST_STREAM, where frames might be received for some time after closing.

当流在传输的时候，各个端点对在传送中的流状态的主观认识可能不同。终端并不协调流的创建；它们是被任意终端单方面创建的。不匹配的状态导致的消极结果是在发送RST_STREAM流之后它们的“关闭”是受限制的，因为可能在关闭之后帧才被接收。

流有以下状态：

Streams have the following states:

idle :

All streams start in the "idle" state. In this state, no frames have been exchanged.

所有流以“空闲”状态开始。在这种状态下，没有任何帧的交换。

The following transitions are valid from this state:

- Sending or receiving a HEADERS frame causes the stream to become "open". The stream identifier is selected as described in Section 5.1.1. The same HEADERS frame can also cause a stream to immediately become "half closed".
- Sending a PUSH_PROMISE frame marks the associated stream for later use. The stream state for the reserved stream transitions to "reserved (local)".

- Receiving a PUSH_PROMISE frame marks the associated stream as reserved by the remote peer. The state of the stream becomes "reserved (remote)".

下列传输在这种状态下是有效的：

- 发送或者接收一个报头HEADERS帧导致流变成“打开”。流标识符如StreamIdentifiers说明。这个报头HEADERS帧同样可能导致流立即变成“半关闭”状态。
- 发送一个推送承诺PUSH_PROMISE帧标记相关的流后续再使用。保留流状态将转换为“保留(本地)”。
- 接收一个推送承诺PUSH_PROMISE帧标记相关的流为远程端点预留的流。这些流的状态变成“保留(远程)”

reserved (local) :

A stream in the "reserved (local)" state is one that has been promised by sending a PUSH_PROMISE frame. A PUSH_PROMISE frame reserves an idle stream by associating the stream with an open stream that was initiated by the remote peer (see Section 8.2).

在“报留(本地)”状态的是已经被承诺发送推送承诺PUSH_PROMISE帧的流。一个推送承诺PUSH_PROMISE帧通过使一个流与一个由远端对等端初始化的打开的流相关联来保留一个空闲流。

In this state, only the following transitions are possible:

- The endpoint can send a HEADERS frame. This causes the stream to open in a "half closed (remote)" state.
- Either endpoint can send a RST_STREAM frame to cause the stream to become "closed". This releases the stream reservation.

在这种状态下，只有下列传输是可能的：

- 端点可以发送报头HEADERS帧，致使流打开到“半封闭(远程)”状态。
- 任意端点能发送一个RST_STREAM帧来使流变成“关闭”。这将释放流的保留。

An endpoint MUST NOT send frames other than HEADERS or RST_STREAM in this state.

在这种状态下一个端绝对不能发送报头HEADERS帧和RST_STREAM以外的帧。

A PRIORITY frame MAY be received in this state. Receiving any frames other than RST_STREAM, or PRIORITY MUST be treated as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

在这种状态下一个优先级PRIORITY帧可能被接收。接收到任何报头HEADERS帧、RST_STREAM帧或者优先级PRIORITY帧以外的帧都将被认为是类型为协议错误PROTOCOL_ERROR的连接错误(章节5.4.1)。

reserved (remote) :

A stream in the "reserved (remote)" state has been reserved by a remote peer.

在“保留(远程)”状态下的流说明已经被远程对等端所保留。

In this state, only the following transitions are possible:

- Receiving a HEADERS frame causes the stream to transition to "half closed (local)".
- Either endpoint can send a RST_STREAM frame to cause the stream to become "closed". This releases the stream reservation.

在这种状态下，只有下列传输是可能的：

- 接收一个报头HEADERS帧并致使流转换到“半封闭(本地)”状态。
- 任意一个端点能发送一个RST_STREAM 帧来使流变成“关闭”。这将释放流的保留。

An endpoint MAY send a PRIORITY frame in this state to reprioritize the reserved stream. An endpoint MUST NOT send any other type of frame other than RST_STREAM or PRIORITY.

这种状态下任意终端可以发送一个优先级PRIORITY帧来变更保留流的优先级顺序。终端绝对不能发送任何RST_STREAM 和优先级PRIORITY以外的帧。

Receiving any other type of frame other than HEADERS or RST_STREAM MUST be treated as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

接收任何RST_STREAM 和优先级PRIORITY以外的帧必须作为类型为
协议错误PROTOCOL_ERROR的连接错误(章节5.4.1)来处理。

open :

A stream in the "open" state may be used by both peers to send frames of any type. In this state, sending peers observe advertised stream level flow control limits (Section 5.2).

处于“打开”状态的流可以被两个对等端来发送任何类型的帧。在这种状态下，发送数据的对等端检查被广播端FlowControl流量控制限制(章节5.2)。

From this state either endpoint can send a frame with an END_STREAM flag set, which causes the stream to transition into one of the "half closed" states: an endpoint sending an END_STREAM flag causes the stream state to become "half closed (local)"; an endpoint receiving an END_STREAM flag causes the stream state to become "half closed (remote)".

在这种
状态下每个终端可以发送一个带有END_STREAM结束流标记的帧来使流转换到其中一种“半关闭”状态：一个终端发送一个结束流END_STREAM标
记使流变成“半封闭”状态；一个终端接收一个结束流END_STREAM标记使流变成“半封闭(远程)”状态。带有结束流END_STREAM标记的报头
HEADERS帧后面可以跟着延续CONTINUATION帧。

Either endpoint can send a RST_STREAM frame from this state, causing it to transition immediately to "closed".

这种状态下各个终端可以发送一个RST_STREAM帧来使流转换到"关闭"状态。

half closed (local) :

A stream that is in the "half closed (local)" state cannot be used for sending frames. Only WINDOW_UPDATE, PRIORITY and RST_STREAM frames can be sent in this state.

“半封闭(本地)”状态下的流不能发送帧。只有窗口更新(WINDOW_UPDATE)、优先级(PRIORITY)和终止流(RST_STREAM)帧能在这种状态下发送。

A stream transitions from this state to "closed" when a frame that contains an END_STREAM flag is received, or when either peer sends a RST_STREAM frame.

这种状态下，当流接收到包含END_STREAM标记的帧或者某个终端发送了RST_STREAM帧，流转换到“关闭”状态。带有结束流END_STREAM标记的报头HEADERS帧后面可以跟着延续CONTINUATION帧。

A receiver can ignore WINDOW_UPDATE frames in this state, which might arrive for a short period after a frame bearing the END_STREAM flag is sent.

这种状态下接收端可以忽略窗口更新WINDOW_UPDATE（或优先级PRIORITY？？？）帧。这种类型的帧有可能在结束流END_STREAM标记到达一小段时间后才收到。

PRIORITY frames received in this state are used to reprioritize streams that depend on the current stream.

优先级(PRIORITY)帧可以在这种状态下接收并用来对依赖当前流的流进行优先级重排序。

half closed (remote) :

A stream that is "half closed (remote)" is no longer being used by the peer to send frames. In this state, an endpoint is no longer obligated to maintain a receiver flow control window if it performs flow control.

"半封闭(远程)"状态下的流不再被对等端用来发送帧。这种状态下，执行流量控制的终端不再承担接收留空控制窗口的工作。

If an endpoint receives additional frames for a stream that is in this state, other than WINDOW_UPDATE, PRIORITY or RST_STREAM, it MUST respond with a stream error (Section 5.4.2) of type STREAM_CLOSED.

如果终端接收到处于这种状态下的流发送的额外的帧，除非是延续CONTINUATION帧，否则必须返回类型为流关闭STREAM_CLOSED的流错误(章节5.4.2)。

A stream can transition from this state to "closed" by sending a frame that contains an END_STREAM flag, or when either peer sends a RST_STREAM frame.

这种状态下，当流发送一个带有终止流END_STREAM标记的帧或者某个终端发送了一个RST_STREAM帧，流将转换到“关闭”状态。

closed :

The "closed" state is the terminal state.

“关闭”状态是终止状态。

An endpoint MUST NOT send frames on a closed stream. An endpoint that receives any frame other than PRIORITY after receiving a RST_STREAM MUST treat that as a stream error (Section 5.4.2) of type STREAM_CLOSED. Similarly, an endpoint that receives any frames after receiving a frame with the END_STREAM flag set MUST treat that as a connection error (Section 5.4.1) of type STREAM_CLOSED, unless the frame is permitted as described below.

终端绝对不能通过关闭的流发送帧。终端在收到RST_STREAM后接收的任何帧必须作为类型为流关闭 STREAM_CLOSED的StreamErrorHandler流错误stream error(章节5.4.2)处理。相似的，终端接收到带有END_STREAM标记设置的数据DATA帧之后的任何帧，或在带有END_STREAM终止流标记且后面没有延续CONTINUATION帧的报头HEADERS帧之后收到任何帧都必须作为类型为流关闭STREAM_CLOSED的连接错误 (章节5.4.1)处理。

WINDOW_UPDATE or RST_STREAM frames can be received in this state for a short period after a DATA or HEADERS frame containing an END_STREAM flag is sent. Until the remote peer receives and processes the frame bearing the END_STREAM

flag, it might send frames of these types. Endpoints MUST ignore WINDOW_UPDATE or RST_STREAM frames received in this state, though endpoints MAY choose to treat frames that arrive a significant time after sending END_STREAM as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

在这种情况下，在带有END_STREAM标记的
DATA或HEADERS帧发送之后一小段时间内可以接收WINDOW_UPDATE或者RST_STREAM帧。在远端对等端接收并处理带有
END_STREAM标记的帧之前，可以发送任意这几种帧。在这种状态下终端必须忽略接收到的WINDOW_UPDATE,PRIORITY, 或
RST_STREAM帧，但终端也可以当作类型为PROTOCOL_ERROR的连接错误(章节5.4.1)处理。

PRIORITY frames can be sent on closed streams to prioritize streams that are dependent on the closed stream. Endpoints SHOULD process PRIORITY frame, though they can be ignored if the stream has been removed from the dependency tree (see Section 5.3.4).

关闭的流上可以发送优先级帧用来对依赖当前关闭流的流进行优先级重排序。终端应该处理优先级帧，但当该流已经从依赖树(章节5.3.4)中移除时可以忽略。

If this state is reached as a result of sending a RST_STREAM frame, the peer that receives the RST_STREAM might have already sent - or enqueued for sending - frames on the stream that cannot be withdrawn. An endpoint MUST ignore frames that it receives on closed streams after it has sent a RST_STREAM frame. An endpoint MAY choose to limit the period over which it ignores frames and treat frames that arrive after this time as being in error.

如果流在发送RST_STREAM帧后转换到这种状态，接收到RST_STREAM的对等端可能已经发送或者队列中准备发送无法取消的帧。终端必须忽略从已经发送RST_STREAM帧的流接收到的帧。终端可以选择设置忽略帧的超时时间并在超过限制后作为错误处理。

Flow controlled frames (i.e., DATA) received after sending RST_STREAM are counted toward the connection flow control window. Even though these frames might be ignored, because they are sent before the sender receives the RST_STREAM, the sender will consider the frames to count against the flow control window.

在发送RST_STREAM之后收到的流量受限帧(如数据DATA帧)转向流量控制窗口连接处理。尽管这些帧可以被忽略，因为他们是在发送端接收到RST_STREAM之前发送的，但发送端会认为这些帧与流量控制窗口不符。

An endpoint might receive a PUSH_PROMISE frame after it sends RST_STREAM. PUSH_PROMISE causes a stream to become "reserved" even if the associated stream

has been reset. Therefore, a RST_STREAM is needed to close an unwanted promised stream.

终端可能在发送RST_STREAM之后接收PUSH_PROMISE帧。即便相关的流已经被重置，推送承诺帧也能使流变成“保留”状态。因此，需要RST_STREAM来关闭一个不想要的被承诺流。

In the absence of more specific guidance elsewhere in this document, implementations SHOULD treat the receipt of a message that is not expressly permitted in the description of a state as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

本文档中没有明确说明的地方，具体实现时接收描述状态中没有明确许可的信息都应作为类型为协议错误(PROTOCOL_ERROR)的连接错误(章节5.4.1)来处理。

5.1.1 Stream Identifiers 流标识

Streams are identified with an unsigned 31-bit integer. Streams initiated by a client MUST use odd-numbered stream identifiers; those initiated by the server MUST use even-numbered stream identifiers. A stream identifier of zero (0x0) is used for connection control messages; the stream identifier zero cannot be used to establish a new stream.

流由31位字节的无符号整数标识。客户端发起的流必须以奇数标示；服务器发起的流必须使用偶数来标示。0(0x0)用来标识连接控制信息流，且绝对不能用来建立一个新流。

HTTP/1.1 requests that are upgraded to HTTP/2 (see Section 3.2) are responded to with a stream identifier of one (0x1). After the upgrade completes, stream 0x1 is "half closed (local)" to the client. Therefore, stream 0x1 cannot be selected as a new stream identifier by a client that upgrades from HTTP/1.1.

HTTP/1.1升级到HTTP/2的请求将收到一个1(0x1)标识的流的响应。升级完成后，0x1流将对客户端处于“半封闭(本地)”状态。因此，0x1流不能被从HTTP/1.1升级的客户端用来作为一个新的流的标识符。

The identifier of a newly established stream MUST be numerically greater than all streams that the initiating endpoint has opened or reserved. This governs streams that are opened using a HEADERS frame and streams that are reserved using PUSH_PROMISE. An endpoint that receives an unexpected stream identifier MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

一个新建立的流标识符必须数值大于任何终端已经打开或者保留的流标识符。规则适用于使用报头帧打开的流以及使用推送承诺帧保留的流。终端收到不规范的流标识符必须响应一个类型为协议错误(PROTOCOL_ERROR)的连接错误。

The first use of a new stream identifier implicitly closes all streams in the "idle" state that might have been initiated by that peer with a lower-valued stream identifier. For example, if a client sends a HEADERS frame on stream 7 without ever sending a frame on stream 5, then stream 5 transitions to the "closed" state when the first frame for stream 7 is sent or received.

新的流标识符第一次被使用时将隐式关闭所有处于“空闲”状态下可能已经被对等端初始化而且流标识符数字小于新标识符的流。例如，一个客户端发送一个流7的报头帧，那么在流7发送或者接收帧后从没有发送帧的流5将转换为“关闭”状态。

Stream identifiers cannot be reused. Long-lived connections can result in an endpoint exhausting the available range of stream identifiers. A client that is unable to establish a new stream identifier can establish a new connection for new streams. A server that is unable to establish a new stream identifier can send a GOAWAY frame so that the client is forced to open a new connection for new streams.

流标识符不能被重复使用。生存期长的连接可能导致流标识符可用范围耗尽。客户端不能新建流标识符时可以针对新流建立一个新的连接。服务端不能新建流标识符时可以发送一个超时帧(GOAWAY)强制客户端对新的流使用新的连接。

5.1.2 Stream Concurrency 流并发

A peer can limit the number of concurrently active streams using the SETTINGS_MAX_CONCURRENT_STREAMS parameter (see Section 6.5.2) within a SETTINGS frame. The maximum concurrent streams setting is specific to each endpoint and applies only to the peer that receives the setting. That is, clients specify the maximum number of concurrent streams the server can initiate, and servers specify the maximum number of concurrent streams the client can initiate.

对等端可以使用设置帧里面的
SETTINGS_MAX_CONCURRENT_STREAMS参数来限制流的并发量。最大并发流设置(章节6.5.2)仅适用于终端并且只对接收到此
设置的对等端有效。也就是说：客户端可以指定服务端能启动的流最大并发量，而且服务端能指定客户端能启动的流最大并发量。终端绝对不能超过对等端设置的限 制。

Streams that are in the "open" state, or either of the "half closed" states count toward the maximum number of streams that an endpoint is permitted to open. Streams in any of these three states count toward the limit advertised in the SETTINGS_MAX_CONCURRENT_STREAMS setting. Streams in either of the "reserved" states do not count toward the stream limit.

处于“打开”或者任意一种“半封闭”状态的流均计入终端被允许启动的流次数中。处于任意这三种状态下的流都将计入SETTINGS_MAX_CONCURRENT_STREAMS设置次数中。处于任意一种“保留”状态下的流不计入打开次数中。

Endpoints MUST NOT exceed the limit set by their peer. An endpoint that receives a HEADERS frame that causes their advertised concurrent stream limit to be exceeded MUST treat this as a stream error (Section 5.4.2). An endpoint that wishes to reduce the value of SETTINGS_MAX_CONCURRENT_STREAMS to a value that is below the current number of open streams can either close streams that exceed the new value or allow streams to complete.

终端绝对不能超过对等端设定的设置。终端接收到报头帧导致他们广
播的并发流超过限制的必须将这作为流错误(章节5.4.2)处理。终端希望将SETTINGS_MAX_CONCURRENT_STREAMS的值减少到
比当前打开的流更小时可以关闭超过新的设置值的流或者允许流结束。

5.2 Flow Control 流量控制

Using streams for multiplexing introduces contention over use of the TCP connection, resulting in blocked streams. A flow control scheme ensures that streams on the same connection do not destructively interfere with each other. Flow control is used for both individual streams and for the connection as a whole.

使用复用流介绍了针对TCP连接的资源争夺导致的流阻塞。流量控制方案等确保同一连接上的流相互之间不会造成破坏性的干扰。流量控制在单个流及整个连接过程中使用。

HTTP/2 provides for flow control through use of the WINDOW_UPDATE frame (Section 6.9).

HTTP/2 通过使用WINDOW_UPDATE帧类型来提供流量控制(章节6.9)。

5.2.1 Flow Control Principles 流量控制规则

HTTP/2 stream flow control aims to allow for future improvements to flow control algorithms without requiring protocol changes. Flow control in HTTP/2 has the following characteristics:

1. Flow control is hop-by-hop, not end-to-end.
2. Flow control is based on window update frames. Receivers advertise how many bytes they are prepared to receive on a stream and for the entire connection. This is a credit-based scheme.

3. Flow control is directional with overall control provided by the receiver. A receiver MAY choose to set any window size that it desires for each stream and for the entire connection. A sender MUST respect flow control limits imposed by a receiver. Clients, servers and intermediaries all independently advertise their flow control window as a receiver and abide by the flow control limits set by their peer when sending.
4. The initial value for the flow control window is 65,535 bytes for both new streams and the overall connection.
5. The frame type determines whether flow control applies to a frame. Of the frames specified in this document, only DATA frames are subject to flow control; all other frame types do not consume space in the advertised flow control window. This ensures that important control frames are not blocked by flow control.
6. Flow control cannot be disabled.
7. HTTP/2 defines only the format and semantics of the WINDOW_UPDATE frame (Section 6.9). This document does not stipulate how a receiver decides when to send this frame or the value that it sends. Nor does it specify how a sender chooses to send packets. Implementations are able to select any algorithm that suits their needs.

HTTP/2流流量控制目标在于允许不需要协议改动的情况下改进流量控制算法。HTTP/2中的流量控制有以下特点：

1. 流量控制是逐跳的，而不是头尾连接的。
2. 流量控制是基于窗口更新帧的。接收端广播自己准备在流及整个连接过程中接收的字节大小。这是一个信用为基础的方案。
3. 流量控制是有方向性的，由接收端全权掌握。接收端可以选择针对流及整个连接设置任意的窗口大小。发送端必须遵守接收端的流量控制限制。客户端、服务端及中端代理作为接收者时都独立的向外广播他们各自的流量控制窗口，作为发送者时遵守接收端的限制。
4. 每个新的流及整个连接的流量控制窗口初始值是65,535字节。
5. 帧类型决定了是否适用流量控制规则。本文档定义的帧中，只有DATA帧受流量控制；所有其他的帧不受广播的流量控制窗口影响。这保证了重要的控制帧不因流量控制所阻塞。
6. 流量控制不能被禁用。
7. HTTP/2只标准化WINDOW_UPDATE帧格式(WINDOW_UPDATE)。它没有规定接收端是何时发送帧或者发送什么值，也没有规定发送端如何选择发送包。具体实现可以选择任何满足需求的算法。

Implementations are also responsible for managing how requests and responses are sent based on priority; choosing how to avoid head of line blocking for requests; and managing the creation of new streams. Algorithm choices for these could interact with any flow control algorithm.

具体实现还负责管理请求和响应是如何基于优先级发送的；如何避免请求头阻塞以及管理新流的创建。这些算法能够与任何流量控制算法相互作用。

5.2.2 Appropriate Use of Flow Control 正确使用流量控制

Flow control is defined to protect endpoints that are operating under resource constraints. For example, a proxy needs to share memory between many connections, and also might have a slow upstream connection and a fast downstream one. Flow control addresses cases where the receiver is unable process data on one stream, yet wants to continue to process other streams in the same connection.

流量控制的定义是用来保护端点在资源约束条件下的操作。例如，一个代理需要在很多连接之间共享内存，也有可能有缓慢的上游连接和快速的下游连接。流量控制解决的情况是接收端在一个流上处理数据的同时同样想继续处理同个连接上的其他流。

Deployments that do not require this capability can advertise a flow control window of the maximum size, incrementing the available space when new data is received. This effectively disables flow control for that receiver. Conversely, a sender is always subject to the flow control window advertised by the receiver.

调度过程中不需要这种能力时可以广播一个最大值的流量控制窗口，增加接收新数据时的可用空间。发送数据时总是受接收端广播的流量控制窗口的管理(见[RFC1323])。

Deployments with constrained resources (for example, memory) can employ flow control to limit the amount of memory a peer can consume. Note, however, that this can lead to suboptimal use of available network resources if flow control is enabled without knowledge of the bandwidth-delay product (see [RFC1323]).

资源约束下(例如内存)的调度可以使用流量来限制一个对等端可以消耗的内存数量。需要注意的是如果在不知道带宽延迟乘积的时候启用流量控制可能导致无法最优的利用可用的网络资源(see RFC1323)。

Even with full awareness of the current bandwidth-delay product, implementation of flow control can be difficult. When using flow control, the receiver MUST read from the TCP receive buffer in a timely fashion. Failure to do so could lead to a deadlock when critical frames, such as WINDOW_UPDATE, are not read and acted upon.

即便是对当前的网络延迟乘积
有充分的认识，流量控制的实现也可能很复杂。当使用流量控制时，接收端必须及时地从TCP接收缓冲区读取数据。这样做可能导致在一些例如

WINDOW_UPDATE的关键帧在HTTP/2不可用时导致死锁。但是流量控制可以保证约束资源能在不需要减少连接利用的情况下得到保护。

5.3 Stream priority 流优先级

A client can assign a priority for a new stream by including prioritization information in the HEADERS frame (Section 6.2) that opens the stream. For an existing stream, the PRIORITY frame (Section 6.3) can be used to change the priority.

新建流的终端可以在报头帧(章节6.2)中包含优先级信息来对流标记优先级。对于已存在的流，优先级帧(章节6.3)可以用来改变流优先级。

The purpose of prioritization is to allow an endpoint to express how it would prefer its peer allocate resources when managing concurrent streams. Most importantly, priority can be used to select streams for transmitting frames when there is limited capacity for sending.

优先级的目的是允许终端表达它如何让对等端管理并发流时分配资源。更重要的是，在发送容量有限时优先级能用来选择流来传输帧。

Explicitly setting the priority for a stream is input to a prioritization process. It does not guarantee any particular processing or transmission order for the stream relative to any other stream. An endpoint cannot force a peer to process concurrent streams in a particular order using priority. Expressing priority is therefore only ever a suggestion.

流的优先级明确设置将输入到优先级处理过程中。它并不能保证能相当其他相关流有特殊的处理或者传输顺序。终端并不能使用优先级强制要求对等端按照特定顺序处理并发流。因此优先级的表达仅仅是一个建议。

Prioritization information can be specified explicitly for streams as they are created using the HEADERS frame, or changed using the PRIORITY frame. Providing prioritization information is optional, so default values are used if no explicit indicator is provided (Section 5.3.5).

优先级信息可以像它们被创建一样使用报头帧或者使用优先级帧来明确指定或者改变。提供优先级信息是可选的，没有明确指定时使用默认值(章节5.3.5)。

5.3.1 Stream Dependencies 流依赖

Each stream can be given an explicit dependency on another stream. Including a dependency expresses a preference to allocate resources to the identified stream rather than to the dependent stream.

每个流可以显式依赖其他流。包含一个依赖偏好设置表示分配资源给特定的流而不是所依赖的流。

A stream that is not dependent on any other stream is given a stream dependency of 0x0. In other words, the non-existent stream 0 forms the root of the tree.

不依赖任何流的流的流依赖为0x0。换句话说，不存在的流标识0组成了树的根。

A stream that depends on another stream is a dependent stream. The stream upon which a stream is dependent is a parent stream. A dependency on a stream that is not currently in the tree - such as a stream in the "idle" state - results in the stream being given a default priority (Section 5.3.5).

依赖其他流的流是一个有依赖流。被依赖的流是父节点流。被依赖的流如果当前不在依赖树中——例如处于“空闲”状态的流——流将会被赋予一个默认的优先级(章节 5.3.5)。

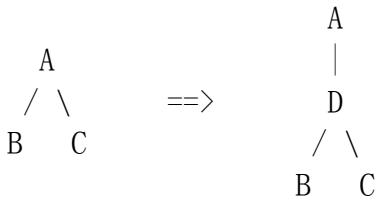
When assigning a dependency on another stream, the stream is added as a new dependency of the parent stream. Dependent streams that share the same parent are not order with respect to each other. For example, if streams B and C are dependent on stream A, and if stream D is created with a dependency on stream A, this results in a dependency order of A followed by B, C, and D in any order.

当指定另一个流的依赖时，这个流将添加到父节点流的子流中。共有相同父节点的流互相之间顺序是不固定的。例如，如果B和C依赖流A,而且如果新创建的流D依赖流A,最终依赖树中的结果就是A被B,C和D以任意顺序依赖。

$$Example of Default Dependency Creation$$

An exclusive flag allows for the insertion of a new level of dependencies. The exclusive flag causes the stream to become the sole dependency of its parent stream, causing other dependencies to become dependent on the prioritized stream. In the previous example, if stream D is created with an exclusive dependency on stream A, this results in D becoming the dependency parent of B and C.

专用标志允许插入一个新的层级的依赖。专用标志导致插入的流成为其父节点流唯一的子节点流，使其他依赖流变成依赖此优先流。在前面这个例子中，如果D流是用专用标志在来创建依赖流A的，那么将导致D流成为了B和C的依赖父节点流。



$$Example of Exclusive Dependency Creation$$

Inside the dependency tree, a dependent stream SHOULD only be allocated resources if all of the streams that it depends on (the chain of parent streams up to 0x0) are either closed, or it is not possible to make progress on them.

在一个依赖树中，一个有依赖的有应该只有在所有其依赖的父节点流(一直到流0x0的所有父节点流)都关闭或者无法取得进展的情况下才能被分配资源。

A stream cannot depend on itself. An endpoint MUST treat this as a stream error (Section 5.4.2) of type PROTOCOL_ERROR.

流不能依赖其自身。终端必须把这种情况当作类型为PROTOCOL_ERROR的流错误(章节5.4.2)处理。

5.3.2 Dependency Weighting 依赖权重

All dependent streams are allocated an integer weight between 1 to 256 (inclusive).

所有有依赖的流都会被分配一个1-256(含)的整数来标识权重。

Streams with the same parent SHOULD be allocated resources proportionally based on their weight. Thus, if stream B depends on stream A with weight 4, and C depends on stream A with weight 12, and if no progress can be made on A, stream B ideally receives one third of the resources allocated to stream C.

具有相同父节点的流应该根据权重比例来分配资源。因此，如果B流依赖流A的权重是4，C流依赖A流的权重是12，那么如果A流上不会有进展了，B流理论上将获取到相对于C流资源的三分之一。

5.3.3 Reprioritization 优先级重组

Stream priorities are changed using the PRIORITY frame. Setting a dependency causes a stream to become dependent on the identified parent stream.

流的优先级是通过使用优先级帧改变的。设置一个依赖将使流变得依赖某个特定的父节点流。

Dependent streams move with their parent stream if the parent is reprioritized. Setting a dependency with the exclusive flag for a reprioritized stream moves all the dependencies of the new parent stream to become dependent on the reprioritized stream.

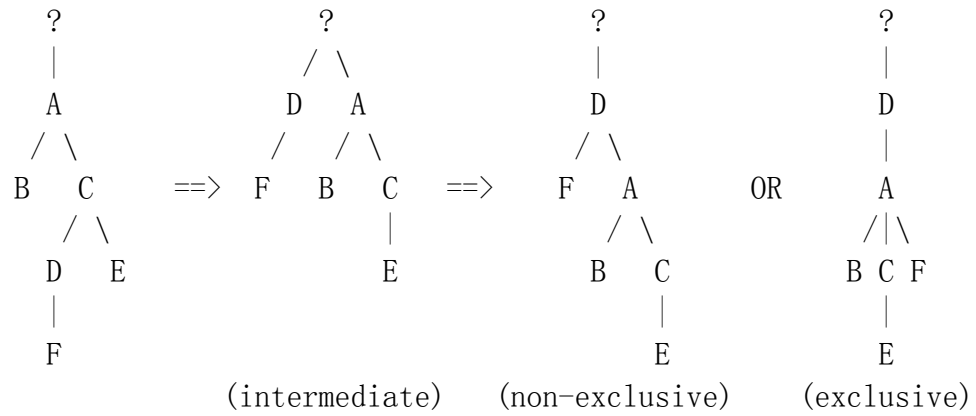
如果父节点流的优先级被修改，子节点流优先级也将改变。使用专用标记来重新设置流优先级将改变所有对其有依赖的流变成对新的优先级改变的流有依赖。

If a stream is made dependent on one of its own dependencies, the formerly dependent stream is first moved to be dependent on the reprioritized stream's previous parent. The moved dependency retains its weight.

如果流被设置成依赖其子流，之前依赖这个流的所有流将首先转成依赖优先级改变的流的之前的父节点流。依赖的改变保持其权重不变。

For example, consider an original dependency tree where B and C depend on A, D and E depend on C, and F depends on D. If A is made dependent on D, then D takes the place of A. All other dependency relationships stay the same, except for F, which becomes dependent on A if the reprioritization is exclusive.

例如，考虑原始依赖树中B和C依赖A,D和E依赖C,且F依赖D。如果A改成依赖D，那么D替换A的位置。其他所有的依赖关系保持不变，不过如果优先级修改使用的是专用标记，那么F将变成依赖A。



5.3.4 Prioritization State Management 优先级状态管理

When a stream is removed from the dependency tree, its dependencies can be moved to become dependent on the parent of the closed stream. The weights of new dependencies are recalculated by distributing the weight of the dependency of the closed stream proportionally based on the weights of its dependencies.

当流从依赖树中移走后，依赖它的子流可以转变成依赖被关闭流的父节点流。新的依赖的权重将根据关闭流的权重以及流自身的权重重新计算。

Streams that are removed from the dependency tree cause some prioritization information to be lost. Resources are shared between streams with the same parent stream, which means that if a stream in that set closes or becomes blocked, any spare capacity allocated to a stream is distributed to the immediate neighbors of the stream. However, if the common dependency is removed from the tree, those streams share resources with streams at the next highest level.

从依赖树中移除的流导致某些优先级信息丢失。资源在具有相同父流的流之间共享，这意味着如果这个集合中的某个流关闭或者阻塞，任何空闲容量将分配给最近的邻居流。然而，如果子流的共有依赖被从树中移除，这些子流将与上一层的流共享资源。

For example, assume streams A and B share a parent, and streams C and D both depend on stream A. Prior to the removal of stream A, if streams A and D are unable to proceed, then stream C receives all the resources dedicated to stream A. If stream A is removed from the tree, the weight of stream A is divided between streams C and D. If stream D is still unable to proceed, this results in stream C receiving a reduced proportion of resources. For equal starting weights, C receives one third, rather than one half, of available resources.

例如，假定A流和B流共有同
个父依赖，且C和D流都依赖A流。在A流移除之前，如果A和D流都无法继续进行，那么C流就会接收所有分配给流A的资源。如果A流从树中移除，流A的权重
将分配给C流和D流。如果D流依旧无法进行，将导致C流获取到的资源比例变少。对于同等的初始权重，C流获取到三分之一而不是二分之一的可用资源。

It is possible for a stream to become closed while prioritization information that creates a dependency on that stream is in transit. If a stream identified in a dependency has had any associated priority information destroyed, then the dependent stream is instead assigned a default priority. This potentially creates suboptimal prioritization, since the stream could be given a priority that is higher than intended.

流有可能在优先级信息在自身创建的依赖还在传输的时候变成关闭状态。如果依赖关系中的一个流存在任何相关的优先级信息被销毁，那么依赖它的流将被分配为默认的优先级。这有可能导致不理想的优先级，因为流可能被赋予一个高于预期的优先级。

To avoid these problems, an endpoint SHOULD retain stream prioritization state for a period after streams become closed. The longer state is retained, the lower the chance that streams are assigned incorrect or default priority values.

为了避免这些问题，终端应该在流关闭后的一段时间内保留流优先级信息。状态被保留的时间越长，流被分配错误的或者默认的优先级值的可能性就越小。

This could create a large state burden for an endpoint, so this state MAY be limited. An endpoint MAY apply a fixed upper limit on the number of closed streams for which prioritization state is tracked to limit state exposure. The amount of additional state an endpoint maintains could be dependent on load; under high load, prioritization state can be discarded to limit resource commitments. In extreme cases, an endpoint could even discard prioritization state for active or reserved streams. If a fixed limit is applied, endpoints SHOULD maintain state for at least as many streams as allowed by their setting for SETTINGS_MAX_CONCURRENT_STREAMS.

这可能增加终端的负担，因此这种状态可以被限制。终端可能会对跟踪状态
的关闭的流的个数使用一个固定的上限来限制状态溢出。终端可能根据负荷来决定保留的额外的状态的数目；在高负荷下，可以丢弃额外的优先级状态来限制资源的
任务。在极端情况下，终端甚至可以丢弃激活或者保留状态流的优先级信息。如果使用了固定的限制，终端应当至少保留跟
SETTINGS_MAX_CONCURRENT_STREAMS设置一样大小的流状态。

An endpoint receiving a PRIORITY frame that changes the priority of a closed stream SHOULD alter the dependencies of the streams that depend on it, if it has retained enough state to do so.

如果有足够的空间，终端接收到优先级帧修改关闭的流的优先级的应该修改流所的依赖的流的优先级，

5.4 Error Handling 错误处理

HTTP/2 framing permits two classes of error:

- An error condition that renders the entire connection unusable is a connection error.
- An error in an individual stream is a stream error.

A list of error codes is included in Section 7.

HTTP/2框架允许两类错误：

- 使整个连接不可用的错误。
- 单个流中出现的错误。

错误码列表可以在"ErrorCodes"找到。

5.4.1 Connection Error Handling 连接错误处理

A connection error is any error which prevents further processing of the framing layer, or which corrupts any connection state.

流错误是阻止帧层更进一步进行处理或者破坏任何流状态的错误。

An endpoint that encounters a connection error SHOULD first send a GOAWAY frame (Section 6.8) with the stream identifier of the last stream that it successfully received from its peer. The GOAWAY frame includes an error code that indicates why the connection is terminating. After sending the GOAWAY frame, the endpoint MUST close the TCP connection.

发送流错误的终端应当首先发送一个超时(GOAWAY)帧(章节6.8)，并带有最近的一个成功从对等端接收帧的流的标识符。GOAWAY超时帧包含链接终端的错误码。发送GOAWAY后，终端必须关闭TCP连接。

It is possible that the GOAWAY will not be reliably received by the receiving endpoint. In the event of a connection error, GOAWAY only provides a best effort attempt to communicate with the peer about why the connection is being terminated.

超时(GOAWAY)帧有可能不被接收端有效接收。在连接错误事件中，超时(GOAWAY)帧是尝试跟对等端通信告知连接终止原因的最佳实践。

An endpoint can end a connection at any time. In particular, an endpoint MAY choose to treat a stream error as a connection error. Endpoints SHOULD send a GOAWAY frame when ending a connection, providing that circumstances permit it.

终端可以在任何时候终止一个连接。类似的，终端可以选择将流错误作为连接错误处理。只要环境许可，终端在终止连接时应当发送一个GOAWAY帧。

4.2 Stream Error Handling 流错误处理

A stream error is an error related to a specific stream that does not affect processing of other streams.

流错误是与特定流相关的错误，并且不会影响其他流的处理。

An endpoint that detects a stream error sends a RST_STREAM frame (Section 6.4) that contains the stream identifier of the stream where the error occurred. The RST_STREAM frame includes an error code that indicates the type of error.

终端检测到流错误时发送一个带有错误发生时的流标识符的RST_STREAM帧(RST_STREAM,章节6.4)。RST_STREAM帧带有表示错误类型的错误码。

A RST_STREAM is the last frame that an endpoint can send on a stream. The peer that sends the RST_STREAM frame MUST be prepared to receive any frames that were sent or enqueued for sending by the remote peer. These frames can be ignored, except where they modify connection state (such as the state maintained for header compression (Section 4.3), or flow control).

RST_STREAM是终端可以发送一个流的最后一帧。发送RST_STREAM帧的对等端必须准备好接收任何由远端对等端发送或者准备发送的帧。这些帧可以被忽略，除非连接状态被修改（例如报头压缩(章节4.3)中的状态）。

Normally, an endpoint SHOULD NOT send more than one RST_STREAM frame for any stream. However, an endpoint MAY send additional RST_STREAM frames if it receives frames on a closed stream after more than a round-trip time. This behavior is permitted to deal with misbehaving implementations.

通常，终端不应该在任何流上发送多个RST_STREAM帧。但是，终端如果在一个关闭的流上超过rtt时间后收到帧，则可以发送的额外的RST_STREAM帧。这种做法是被允许用来处理这种非常规情况。

An endpoint MUST NOT send a RST_STREAM in response to an RST_STREAM frame, to avoid looping.

终端绝不能在收到RST_STREAM帧后响应一个RST_STREAM帧，避免死循环。

5.4.3 Connection Termination 连接终止

If the TCP connection is torn down while streams remain in open or half closed states, then the endpoint MUST assume that those streams were abnormally interrupted and could be incomplete.

如果TCP连接在流仍然保持打开或者半封闭状态下断开，那么终端必须假定这些流是异常终端且不完整的。

5.5 Extending HTTP/2 HTTP/2扩展

HTTP/2 permits extension of the protocol. Protocol extensions can be used to provide additional services or alter any aspect of the protocol, within the limitations described in this section. Extensions are effective only within the scope of a single HTTP/2 connection.

HTTP/2协议允许扩展。协议扩展可用在提供额外的服务或者在此节定义的限制内修改协议的任何方面。扩展只在单个HTTP/2连接范围内有效。

Extensions are permitted to use new frame types (Section 4.1), new settings (Section 6.5.2), new error codes (Section 7), or new header fields that start with a colon (:). Of these, registries are established for frame types (Section 11.2), settings (Section 11.3) and error codes (Section 11.4).

扩展可以允许使用新的帧类型(章节4.1),新的设置(章节6.5.2)，新的错误码(章节7)，或者新的(:)开头的报头字段。这里面，注册将会在帧类型(章节11.2)、设置(章节11.3)和错误码(章节11.2)中建立。

Implementations MUST ignore unknown or unsupported values in all extensible protocol elements. Implementations MUST discard frames that have unknown or unsupported types. This means that any of these extension points can be safely used by extensions without prior arrangement or negotiation.

具体实现必须忽略扩展协议元素中未知或者不支持的值。实现必须丢弃包含未知或者不支持的类型的帧。这意味着任意这些扩展点能被扩展安全的使用，不需要事先安排或协商。

However, extensions that could change the semantics of existing protocol components MUST be negotiated before being used. For example, an extension that changes the layout of the HEADERS frame cannot be used until the peer has given a positive signal that this is acceptable. In this case, it could also be necessary to coordinate when the revised layout comes into effect. Note that treating any frame other than DATA frames as flow controlled is such a change in semantics, and can only be done through negotiation.

然而，可能导致现有协议元素语义改变的扩展必须经过协商后才能使用。例如，一个修改报头帧布局的扩展使用之前必须收到对等端发送的允许使用的信号。在这种情况下，经过修订的布局生效的时候也可能有必要进行协商。注意对待任意数据帧以外的帧作为受流量控制的修改也是语义上的修改，也必须在通过协商才能完成。

This document doesn't mandate a specific method for negotiating the use of an extension, but notes that a setting (Section 6.5.2) could be used for that purpose. If both peers set a value that indicates willingness to use the extension, then the extension can be used. If a setting is used for extension

negotiation, the initial value MUST be defined so that the extension is initially disabled.

本文档并不强制要求使用特定的方法来完成使用扩展的协商，但注意可以通过使用设置(章节6.5.2)来实现这个目的。如
果对等端双方都设置了指示愿意使用扩展的值，那么这个扩展就可以被使用。如果某个设置用来协商扩展的使用，那么必须定义初始值来设置扩展初始时是被禁用 的。

6.Frame Definitions 帧定义

This specification defines a number of frame types, each identified by a unique 8-bit type code. Each frame type serves a distinct purpose either in the establishment and management of the connection as a whole, or of individual streams.

本规范定义了一系列的帧类型，每种类型由独特的8位类型代码标记。不管是在连接管理或单独的流中，每种帧都为了特定的目的而服务。

The transmission of specific frame types can alter the state of a connection. If endpoints fail to maintain a synchronized view of the connection state, successful communication within the connection will no longer be possible. Therefore, it is important that endpoints have a shared comprehension of how the state is affected by the use any given frame.

特定帧类型的传输可以修改连接的状态。如果终端不能保持同步的连接状态，连接中的通信将失败。因此，终端对于如何使用给定帧修改状态达成共识很重要。

6.1 DATA 数据帧

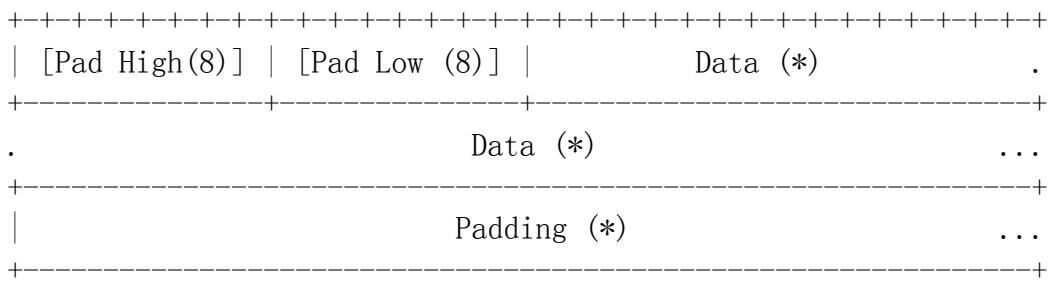
DATA frames (type=0x0) convey arbitrary, variable-length sequences of octets associated with a stream. One or more DATA frames are used, for instance, to carry HTTP request or response payloads.

数据帧（类型=0x0）表示随意，由伴随流的可变长度序列组成。例如，一个或多个数据帧被用来携带HTTP请求或者响应的载体。

DATA frames MAY also contain arbitrary padding. Padding can be added to DATA frames to obscure the size of messages.

数据帧也可以包含任意填充物。填充物可以添加到数据帧中来隐藏消息的大小。

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```



The DATA frame contains the following fields:

- Pad Length:An 8-bit field containing the length of the frame padding in units of octets. This field is optional and is only present if the PADDED flag is set.
- Data:Application data. The amount of data is the remainder of the frame payload after subtracting the length of the other fields that are present.
- Padding:Padding octets that contain no application semantic value. Padding octets MUST be set to zero when sending and ignored when receiving.

数据帧包含以下字段：

- Pad Length ：
  包含字节为单位的帧填充长度的8位字段。这个字段是可选的，并且只在设置了PADDED标记的情况下呈现。
- Data ： 应用数据。数据量的大小是帧的有效载荷减去其他呈现字段的长度。
- Padding ：
  填充字节不包含任何应用语义值。天聪字节必须在发送的时候设置为0，在接收的时候忽略。

The DATA frame defines the following flags:

- END_STREAM (0x1):Bit 1 being set indicates that this frame is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to enter one of the "half closed" states or the "closed" state (Section 5.1).
- END_SEGMENT (0x2):Bit 2 being set indicates that this frame is the last for the current segment. Intermediaries MUST NOT coalesce frames across a segment boundary and MUST preserve segment boundaries when forwarding frames.
- PADDED (0x8):Bit 4 being set indicates that the Pad Length field is present.

数据帧定义了以下标记：

- END_STREAM (0x1)：
  位1用来表示当前帧是确定的流发送的最后一帧。设置这个标记时流进入到一种半封闭状态或者关闭状态(章节5.1)。
- END_SEGMENT (0x2)：
  位2表示是当前端的最后一帧。代理端绝对不能跨越多个端的边界来合并帧，转发帧的时候代理端必须保持片段的边界。
- PADDED (0x8)： 位4用来表示Pad Length 字段是可见的。

DATA frames MUST be associated with a stream. If a DATA frame is received whose stream identifier field is 0x0, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

数据帧绝对需要与流相关联。如果接收到流标记字段是0x0的数据帧，必须响应一个类型为协议错误的连接错误(章节5.4.1)。

DATA frames are subject to flow control and can only be sent when a stream is in the "open" or "half closed (remote)" states. The entire DATA frame payload is included in flow control, including Pad Length and Padding fields if present. If a DATA frame is received whose stream is not in "open" or "half closed (local)" state, the recipient MUST respond with a stream error (Section 5.4.2) of type STREAM_CLOSED.

数据帧遵从流量控制，并且只有在流是打开或者半封闭(远端)状态下才能够被发送。填充同样包含在流量控制中。如果数据帧在相关流不是在打开和半封闭(本地)状态下被接收，接收端必须响应一个类型为流关闭的流错误(章节5.4.2)。

The total number of padding octets is determined by the value of the Pad Length field. If the length of the padding is greater than the length of the remainder of the frame payload, the recipient MUST treat this as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

填充字节的总数取决于Pad Length
的值。如果填充物的大小大于帧有效载荷的大小，接收端必须作为类型为协议错误的连接错误(章节5.4.1)处理。

Note: A frame can be increased in size by one octet by including a Pad Length field with a value of zero.

请注意： 为帧的大小加1字节可通过增加一个值为0的Pad Length 值的方法。

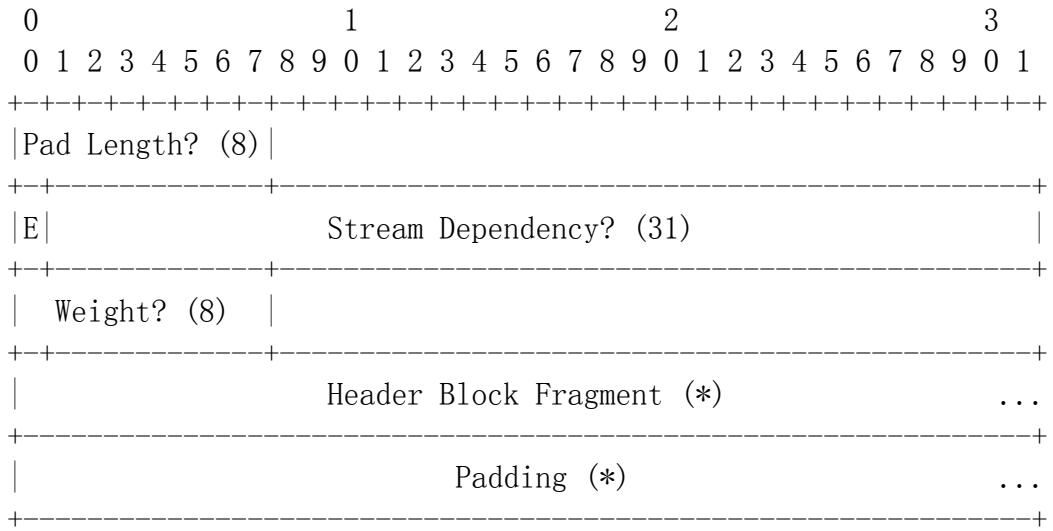
Use of padding is a security feature; as such, its use demands some care, see Section 10.7.

使用填充是一种安全手段；例如，用来满足特定需求，见章节10.7.

6.2 HEADERS 报头

The HEADERS frame (type=0x1) carries name-value pairs. It is used to open a stream (Section 5.1). HEADERS frames can be sent on a stream in the "open" or "half closed (remote)" states.

报头帧(类型=0x1)由键值对组成。它用来打开一个流(章节5.1)。报头帧能在流打开或者半封闭(远程)的状态下发送。



The HEADERS frame payload has the following fields:

- Pad Length:An 8-bit field containing the length of the frame padding in units of octets. This field is optional and is only present if the PADDED flag is set.
- E:A single bit flag indicates that the stream dependency is exclusive, see Section 5.3. This field is optional and is only present if the PRIORITY flag is set.
- Stream Dependency:A 31-bit stream identifier for the stream that this stream depends on, see Section 5.3. This field is optional and is only present if the PRIORITY flag is set.
- Weight:An 8-bit weight for the stream, see Section 5.3. Add one to the value to obtain a weight between 1 and 256. This field is optional and is only present if the PRIORITY flag is set.
- Header Block Fragment:A header block fragment (Section 4.3).
- Padding:Padding octets.

报头帧主体有以下字段：

- Pad Length ：
  8位的包含单位为字节帧填充长度字段。这个字段是可选的并只有在设置了PADDED标记的情况下才呈现。
- E ：
  1位的标记用于标识流依赖是否是专用的，见章节5.3。这个字段是可选的，并且只在优先级标记设置的情况下才呈现。
- Stream Dependency ：
  31位流所依赖的流的标识符的字段，见章节5.3。这个字段是可选的，并且只在优先级标记设置的情况下才呈现。
- Weight ： 流的8位权重标记，见章节5.3。添加一个1-256的值来存储流的权重。这个字段是可选的，并且只在优先级标记设置的情况下才呈现。
- Header Block Fragment ： 报头块碎片。
- Padding ： 填充字节

The HEADERS frame defines the following flags:

- END_STREAM (0x1):Bit 1 being set indicates that the header block (Section 4.3) is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to enter one of "half closed" states (Section 5.1).
- A HEADERS frame that is followed by CONTINUATION frames carries the END_STREAM flag that signals the end of a stream. A CONTINUATION frame cannot be used to terminate a stream.
- END_SEGMENT (0x2):Bit 2 being set indicates that this frame is the last for the current segment. Intermediaries MUST NOT coalesce frames across a segment boundary and MUST preserve segment boundaries when forwarding frames. END_SEGMENT, like END_STREAM, applies to a complete sequence of HEADERS and CONTINUATION frames.
- END_HEADERS (0x4):Bit 3 being set indicates that this frame contains an entire header block (Section 4.3) and is not followed by any CONTINUATION frames. A HEADERS frame without the END_HEADERS flag set MUST be followed by a CONTINUATION frame for the same stream. A receiver MUST treat the receipt of any other type of frame or a frame on a different stream as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.
- PADDED (0x8):Bit 4 being set indicates that the Pad Length field is present.
- PRIORITY (0x20):Bit 6 being set indicates that the Exclusive Flag (E), Stream Dependency, and Weight fields are present; see Section 5.3.

报头帧定义了以下标记：

- END_STREAM (0x1) ：
  位1用来标识这是发送端对确定的流发送的最后报头区块(章节4.3)。设置这个标记将

使流进入一种半封闭状态(章节5.1)。
后面伴随带有END_STREAM标记的延续帧的报头帧表示流的终止。延续帧不用来用终止流。

- END_SEGMENT (0x2)：
  位2表示这是当前端的最后一帧。中介端绝对不能跨片段来合并帧，且在转发帧的时候必须保持片段的边界。
- END_HEADERS (0x4)：
  位3表示帧包含了整个的报头块(章节4.3)，且后面没有延续帧。
  不带有END_HEADERS标记的报头帧在同个流上后面必须跟着延续帧。接收端接收到任何其他类型的帧或者在其他流上的帧必须作为类型为协议错误的连接 错误处理。
- PADDED (0x8)： 位4表示Pad Length字段会呈现。
- PRIORITY (0x8)：
  位6设置指示专用标记(E),流依赖及权重字段将会呈现；见章节5.3

The payload of a HEADERS frame contains a header block fragment (Section 4.3). A header block that does not fit within a HEADERS frame is continued in a CONTINUATION frame (Section 6.10).

报头帧的主体包含一个报头区块碎片(章节4.3)。报头区块大于一个报头帧的将在延续帧中(章节6.10)继续传送。

HEADERS frames MUST be associated with a stream. If a HEADERS frame is received whose stream identifier field is 0x0, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

报头帧必须与一个流相关联。如果一个接收到一个流标示识0x0得报头帧，接收端必须响应一个类型为协议错误的连接错误(章节5.4.1)。

The HEADERS frame changes the connection state as described in Section 4.3.

报头帧改变连接状态在章节4.3此中表述。

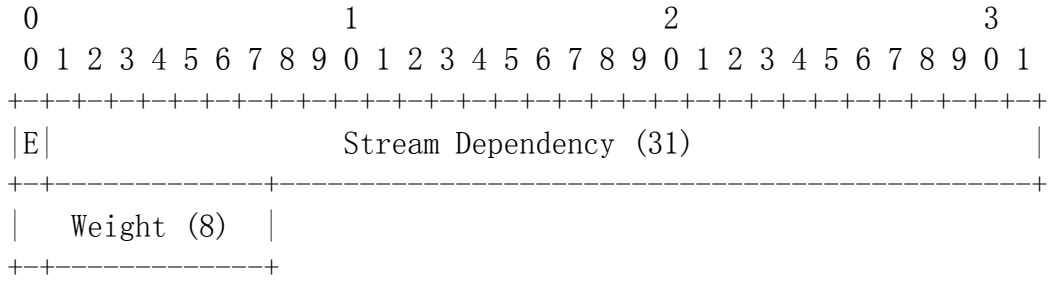
The HEADERS frame includes optional padding. Padding fields and flags are identical to those defined for DATA frames (Section 6.1).

报头帧包含可选的填充段。填充字段和标记同数据帧中描述的相同(章节6.1)。

6.3 PRIORITY 优先级帧

The PRIORITY frame (type=0x2) specifies the sender-advised priority of a stream (Section 5.3). It can be sent at any time for an existing stream, including closed streams. This enables reprioritization of existing streams.

优先级帧(type=0x2)明确了发送者建议的流的优先级(章节5.3)。它可以任意时间在存在的流中发送，包括已闭合的流。这个允许了在已存在的流中重新排列优先级次序。



The payload of a PRIORITY frame contains the following fields:

- E: A single bit flag indicates that the stream dependency is exclusive, see Section 5.3.
- Stream Dependency: A 31-bit stream identifier for the stream that this stream depends on, see Section 5.3.
- Weight: An 8-bit weight for the identified stream dependency, see Section 5.3. Add one to the value to obtain a weight between 1 and 256.

优先级帧实体包含下列字段：

- E ：一位的标记，指示流的依赖是专有的，见章节5.3
- Stream Dependency ： 标识流所依赖的流的31位流标识符，见章节5.3
- Weight: 流的依赖的的权重(8位)，见章节5.3。添加一个1-256的权重值。

The PRIORITY frame does not define any flags.

优先级不定义任何标记。

The PRIORITY frame is associated with an existing stream. If a PRIORITY frame is received with a stream identifier of 0x0, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

优先级与存在的流相关联。如果接收端收到流标识为0的优先级帧，必须响应一个类型为协议错误的连接错误(章节5.4.1)。

The PRIORITY frame can be sent on a stream in any of the "reserved (remote)", "open", "half closed (local)", "half closed (remote)", or "closed" states, though it cannot be sent between consecutive frames that comprise a single header block (Section 4.3). Note that this frame could arrive after processing or frame sending has completed, which would cause it to have no effect on the current stream. For a stream that is in the "half closed (remote)" or "closed" - state, this frame can only affect processing of the current stream and not frame transmission.

优先级帧可以在流状态为“保留(远端)”、“打开”、“半封闭(本地)”或者“半封闭(远程)”状态下发送,但
它不能在由单个报头区块组成的连续帧之间发送。需要注意的是这个帧可能在处理或者帧发送已经完成之后才到达，这可能导致没有效果。对于处于“半封闭(远
程)”状态下的流，优先级帧只能影响流的处理而不是传输。

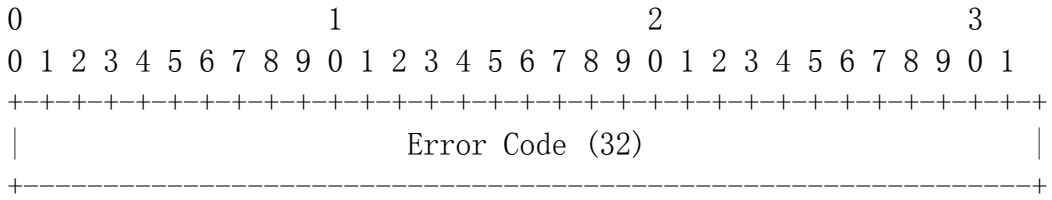
The PRIORITY frame is the only frame that can be sent for a stream in the "closed" state. This allows for the reprioritization of a group of dependent streams by altering the priority of a parent stream, which might be closed. However, a PRIORITY frame sent on a closed stream risks being ignored due to the peer having discarded priority state information for that stream.

优先级帧是关闭的流上唯一允许发送的帧。这允许通过修改可能已关闭的父节点流的优先级来重新设置一组有依赖的流的优先级。然而，关闭流上发送的优先级帧可能存在被对等端忽略的风险，因为对等端可能已经丢弃了这个流的优先级状态信息。

6.4 RST_STREAM RST_STREAM帧

The RST_STREAM frame (type=0x3) allows for abnormal termination of a stream. When sent by the initiator of a stream, it indicates that they wish to cancel the stream or that an error condition has occurred. When sent by the receiver of a stream, it indicates that either the receiver is rejecting the stream, requesting that the stream be cancelled, or that an error condition has occurred.

RST_STREAM帧(type=0x3)允许流的异常终止。当被流的指示器发送时，它表示期望取消流或者错误条件发生。当被接收端的流发送时，它表示接收者希望拒绝流、流被取消或者发生了错误。



The RST_STREAM frame contains a single unsigned, 32-bit integer identifying the error code (Section 7). The error code indicates why the stream is being terminated.

RST_STREAM 帧由一个无符号的32位整数标记错误码。错误码指明流被终止的原因。

The RST_STREAM frame does not define any flags.

RST_STREAM 帧未定义任何标记。

The RST_STREAM frame fully terminates the referenced stream and causes it to enter the closed state. After receiving a RST_STREAM on a stream, the receiver MUST NOT send additional frames for that stream. However, after sending the RST_STREAM, the sending endpoint MUST be prepared to receive and process additional frames sent on the stream that might have been sent by the peer prior to the arrival of the RST_STREAM.

RST_STREAM
帧完全终止相关的流并使其转入关闭状态。在接收到流的RST_STREAM帧后，接收端绝对不能在流上发送额外的帧。然而，在发送RST_STREAM帧
后，发送端必须要准备接收并处理流上的其他帧，因为对等端有可能在收到RST_STREAM帧前就已经发送这些帧。

RST_STREAM frames MUST be associated with a stream. If a RST_STREAM frame is received with a stream identifier of 0x0, the recipient MUST treat this as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

RST_STREAM 帧必须与流相关联。如果接收端收到流标示符为0x0的RST_STREAM
帧，必须作为类型为协议错误的连接错误(章节5.4.1)处理。

RST_STREAM frames MUST NOT be sent for a stream in the "idle" state. If a RST_STREAM frame identifying an idle stream is received, the recipient MUST treat this as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

RST_STREAM帧绝对不能在流处于“空闲”状态下发送。如果接收端收到流状态为空闲的RST_STREAM帧，必须作为类型为协议错误的连接错误(章节5.4.1)处理。

6.5 SETTINGS 设置帧

The SETTINGS frame (type=0x4) conveys configuration parameters that affect how endpoints communicate, such as preferences and constraints on peer behavior. The SETTINGS frame is also used to acknowledge the receipt of those parameters. Individually, a SETTINGS parameter can also be referred to as a "setting".

设置帧(type=0x4)包含影响如何与终端通信的设置参数(例如偏好设置以及对等端的行为约束)，并且用来确认这些参数的接收。单个的设置参数也可以被认为是“设置”。

SETTINGS parameters are not negotiated; they describe characteristics of the sending peer, which are used by the receiving peer. Different values for the same parameter can be advertised by each peer. For example, a client might set a high initial flow control window, whereas a server might set a lower value to conserve resources.

设置参数不是通过协商确定的；它们描述发送端的特点，并被接收端使用。相同的参数对不同的对等端设置可能不同。例如，一个客户端可能设置一个较高的流量控制窗口，而服务器为了保存资源可能设置一个较低的值。

A SETTINGS frame MUST be sent by both endpoints at the start of a connection, and MAY be sent at any other time by either endpoint over the lifetime of the connection. Implementations MUST support all of the parameters defined by this specification.

设置帧必须由两个终端在连接开始的时候发送，并且可以由各个终端在连接生存期的任意时间发送。具体实现必须支持本规范定义的所有参数。

Each parameter in a SETTINGS frame replaces any existing value for that parameter. Parameters are processed in the order in which they appear, and a receiver of a SETTINGS frame does not need to maintain any state other than the current value of its parameters. Therefore, the value of a SETTINGS parameter is the last value that is seen by a receiver.

设置帧的所有参数将替换参数中现有值。参数由他们出现的顺序来处理，而且接收设置帧并不需要保存当前值以外的任何状态。因此，设置参数的值是接收端接收到的最后一个值。

SETTINGS parameters are acknowledged by the receiving peer. To enable this, the SETTINGS frame defines the following flag:

设置参数是被接收端公认的。为了实现这个，设置帧定义了以下标记：

ACK (0x1) ： Bit 1 being set indicates that this frame acknowledges receipt and application of the peer's SETTINGS frame. When this bit is set, the payload of the SETTINGS frame MUST be empty. Receipt of a SETTINGS frame with the ACK flag set and a length field value other than 0 MUST be treated as a connection error (Section 5.4.1) of type FRAME_SIZE_ERROR. For more info, see Settings Synchronization (Section 6.5.3).

ACK (0x1) ：
位1表示设置帧已被接收端接收并应用。如果这个位设置了，设置帧的载体必须为空。接收到字段长度不是0的带有ACK标记的设置帧必须作为类型为帧大小错误的连接错误(章节5.4.1)处理，更多信息，见同步设置(章节6.5.3)。

SETTINGS frames always apply to a connection, never a single stream. The stream identifier for a SETTINGS frame MUST be zero (0x0). If an endpoint receives a SETTINGS frame whose stream identifier field is anything other than 0x0, the endpoint MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

设置帧总是应用于连接，而不是一个单独的流。流的设置帧标识必须为0.如果终端接收到流设置帧标识不是0的设置帧，必须响应一个类型为协议错误的连接错误(章节5.4.1)。

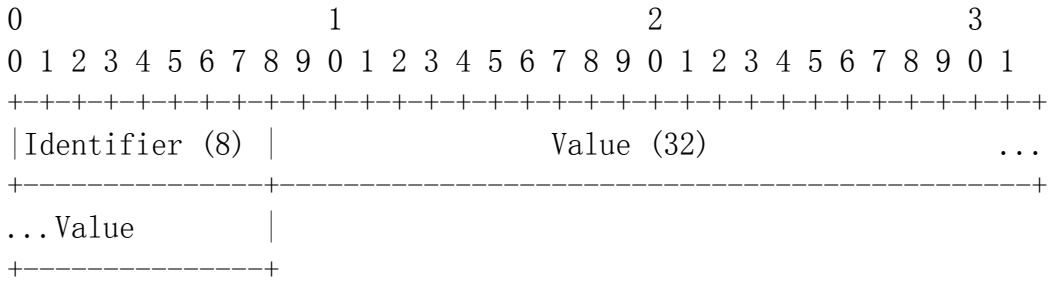
The SETTINGS frame affects connection state. A badly formed or incomplete SETTINGS frame MUST be treated as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

设置帧影响连接状态。格式错误或者未完成的设置帧必须作为类型为协议错误的连接错误处理。

6.5.1 SettingFormat 设置帧格式

The payload of a SETTINGS frame consists of zero or more parameters, each consisting of an unsigned 16-bit setting identifier and an unsigned 32-bit value.

设置帧载体包含0个或多个参数，每个包含一个无符号的16位标识以及一个无符号的32位值。



6.5.2 Defined SETTINGS Parameters 设置帧参数

The following parameters are defined:

- SETTINGS_HEADER_TABLE_SIZE (0x1):Allows the sender to inform the remote endpoint of the maximum size of the header compression table used to decode header blocks. The encoder can select any size equal to or less than this value by using signaling specific to the header compression format inside a header block. The initial value is 4,096 bytes.
- SETTINGS_ENABLE_PUSH (0x2):This setting can be use to disable server push (Section 8.2). An endpoint MUST NOT send a PUSH_PROMISE frame if it receives this parameter set to a value of 0. An endpoint that has both set this parameter to 0 and had it acknowledged MUST treat the receipt of a PUSH_PROMISE frame as a connection error (Section 5.4.1) of type PROTOCOL_ERROR. The initial value is 1, which indicates that server push

is permitted. Any value other than 0 or 1 MUST be treated as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

- SETTINGS_MAX_CONCURRENT_STREAMS (0x3):Indicates the maximum number of concurrent streams that the sender will allow. This limit is directional: it applies to the number of streams that the sender permits the receiver to create. Initially there is no limit to this value. It is recommended that this value be no smaller than 100, so as to not unnecessarily limit parallelism. A value of 0 for SETTINGS_MAX_CONCURRENT_STREAMS SHOULD NOT be treated as special by endpoints. A zero value does prevent the creation of new streams, however this can also happen for any limit that is exhausted with active streams. Servers SHOULD only set a zero value for short durations; if a server does not wish to accept requests, closing the connection could be preferable.
- SETTINGS_INITIAL_WINDOW_SIZE (0x4):Indicates the sender's initial window size (in bytes) for stream level flow control. The initial value is 65,535. This setting affects the window size of all streams, including existing streams, see Section 6.9.2. Values above the maximum flow control window size of 231 - 1 MUST be treated as a connection error (Section 5.4.1) of type FLOW_CONTROL_ERROR.

定义了以下参数：

- SETTINGS_HEADER_TABLE_SIZE (1) :
  允许发送端通知远端终端解码报头区块的报头压缩表的最大承载量。这个编码器可以选择在报头区块中使用特定信号来减少报头压缩的大小（？？？）。初始值是4,096个字节。
- SETTINGS_ENABLE_PUSH (2) :
  这个参数可以用来关闭服务器推送。终端在接收到此参数为0的情况下绝对不能发送服务器推送承诺帧。终端在已经设置此参数为0并且承认的情况下必须对接收到的服务器推送作为类型为协议错误的连接错误处理。
  初始值是1，表示推送是许可的。任何不是0或1的值必须作为类型为协议错误的连接错误处理。
- SETTINGS_MAX_CONCURRENT_STREAMS (3) :
  标明发送者允许的最大并发流。此限制是定向的：它适用于发送端允许接收端创建的最大并发流的数量。初始化时这个值没有限制。建议值不要大于100,以免不必要的限制并行。
  此设置为0的值不应该被终端认为是特殊的。0的值阻止了新的流的创建，另外它也适用于被激活的流用尽的任何限制。对于短连接不应该设置此参数为0；如果服务端不希望接收任何请求，最佳的做法是关闭连接。
- SETTINGS_INITIAL_WINDOW_SIZE (4) :
  表示发送端对流层流量控制的初始窗口大小(字节单位)。初始值是65,535。
  这个参数影响了所有流的窗口大小，包括现有的流。见章节6.9.2.
  流量控制窗口大小值大于2的31次方-1的必须被作为流量控制错误的连接错误处理。

An endpoint that receives a SETTINGS frame with any unknown or unsupported identifier MUST ignore that setting.

终端收到其他标记的设置帧必须作为类型为协议错误的连接错误处理。

6.5.3 Settings Synchronization 设置同步

Most values in SETTINGS benefit from or require an understanding of when the peer has received and applied the changed the communicated parameter values. In order to provide such synchronization timepoints, the recipient of a SETTINGS frame in which the ACK flag is not set MUST apply the updated parameters as soon as possible upon receipt.

大部分设置值收益于或者需要了解对等端接收到并且改变了通信过的参数的值的时机。为了提供这样一种同步的时间点，接收到没有设置ACK标记的设置帧必须尽快将更新过的参数适用于接收端上。

The values in the SETTINGS frame MUST be applied in the order they appear, with no other frame processing between values. Once all values have been applied, the recipient MUST immediately emit a SETTINGS frame with the ACK flag set. Upon receiving a SETTINGS frame with the ACK flag set, the sender of the altered parameters can rely upon their application.

设置帧的值必须按照它们出现的顺序被使用，在处理值中间不能处理其他帧。一旦所有的值被应用，接收端必须马上发送一个带有ACK标记的设置帧。在接收到带有ACK标记的设置帧后，修改参数的发送端可以认为修改已生效。

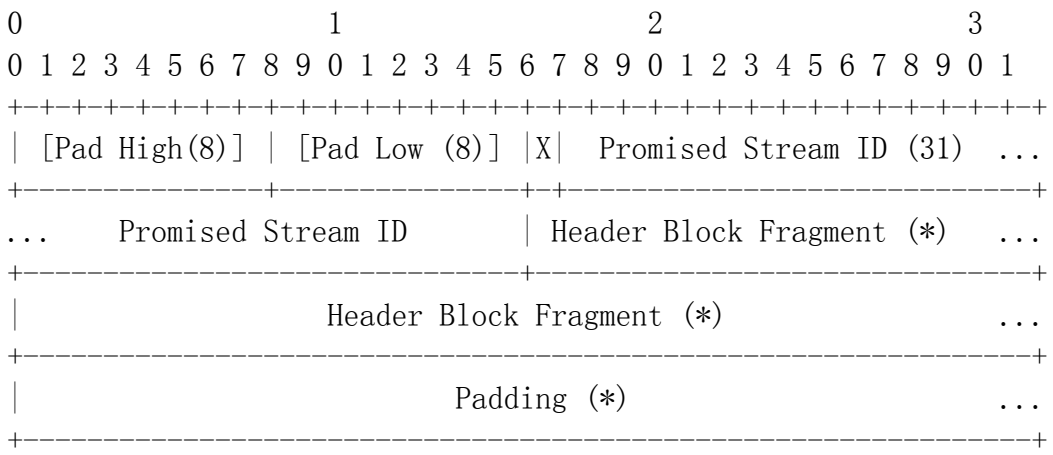
If the sender of a SETTINGS frame does not receive an acknowledgement within a reasonable amount of time, it MAY issue a connection error (Section 5.4.1) of type SETTINGS_TIMEOUT.

如果设置参数的发送端没有在一定时间内收到确认响应，它可以发出一个类型为设置超时的连接错误(章节5.4.1)。

6.6 PUSH_PROMISE 推送承诺帧

The PUSH_PROMISE frame (type=0x5) is used to notify the peer endpoint in advance of streams the sender intends to initiate. The PUSH_PROMISE frame includes the unsigned 31-bit identifier of the stream the endpoint plans to create along with a set of headers that provide additional context for the stream. Section 8.2 contains a thorough description of the use of PUSH_PROMISE frames.

推送承诺帧(type=0x5)用来在流发送者准备发送流之前告知对等端。推送承诺帧包含了终端准备创建的长流的31位无符号标记以及提供附加上下文的报头的集合。章节8.2详细描述了推送承诺帧的使用。



The PUSH_PROMISE frame payload has the following fields:

- Pad Length:An 8-bit field containing the length of the frame padding in units of octets. This field is optional and is only present if the PADDED flag is set.
- R:A single reserved bit.
- Promised Stream ID:This unsigned 31-bit integer identifies the stream the endpoint intends to start sending frames for. The promised stream identifier MUST be a valid choice for the next stream sent by the sender (see new stream identifier (Section 5.1.1)).
- Header Block Fragment: header block fragment (Section 4.3) containing request header fields.
- Padding:Padding octets.

报头帧载体包含以下字段：

- Pad High ： 填充大小高位。这个字段只有在PAD_HIGH标记设置的情况下才呈现。
- Pad Low ： 填充大小低位。这个字段只有在PAD_LOW标记设置的情况下才呈现。
- X ： 单独的保留位。
- Promised Stream ID ：
  这个无符号31位整数表示终端准备发送的流标记。被承诺的流标记必须对发送端准备发送的下一个流来说是有效选择。
- Header Block Fragment ： 包含请求头字段的报头区块碎片(章节5.1.1)。
- Padding ： 填充字节。

The PUSH_PROMISE frame defines the following flags:

- END_HEADERS (0x4): Bit 3 being set indicates that this frame contains an entire header block (Section 4.3) and is not followed by any CONTINUATION frames. A PUSH_PROMISE frame without the END_HEADERS flag set MUST be followed by a CONTINUATION frame for the same stream. A receiver MUST treat the receipt of any other type of frame or a frame on a different stream as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.
- PADDED (0x8): Bit 4 being set indicates that the Pad Length field is present.

推送承诺帧定义了以下标记：

- END_HEADERS (0x4) : 位3
  表明帧包含了整个报头区块(章节4.3)并且不跟着延续帧。
  不带有END_HEADERS标记的推送承诺帧在同个流上面后面必须跟着延续帧。接收端接收到任何其他类型或者其他流觞的帧必须作为类型为协议错误的连接
  错误(章节5.4.1)处理。
- PADDED (0x8) : 位4 表明Pad长度字段是已设置。

PUSH_PROMISE frames MUST be associated with an existing, peer-initiated stream. The stream identifier of a PUSH_PROMISE frame indicates the stream it is associated with. If the stream identifier field specifies the value 0x0, a recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

推送承诺帧必须与现有的由对等端初始化的流相关联。如果流标识字段为0x0，接收端必须响应一个类型为协议错误的连接错误(章节5.4.1)。

Promised streams are not required to be used in the order they are promised. The PUSH_PROMISE only reserves stream identifiers for later use.

被承诺的流并不需要以被承诺的顺序使用。推送承诺只保留接下来会使用的流的标识符。

PUSH_PROMISE MUST NOT be sent if the SETTINGS_ENABLE_PUSH setting of the peer endpoint is set to 0. An endpoint that has set this setting and has received acknowledgement MUST treat the receipt of a PUSH_PROMISE frame as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

如果对等端的SETTINGS_ENABLE_PUSH设置是0那么推送承诺绝对不能发送。终端已经设置了禁止推送承诺并且收到确认的必须将接收到推送承诺帧作为类型为协议错误的连接错误处理(章节5.4.1)。

Recipients of PUSH_PROMISE frames can choose to reject promised streams by returning a RST_STREAM referencing the promised stream identifier back to the sender of the PUSH_PROMISE.

推送承诺的接收端可以选择给推送承诺的发送端返回一个与被承诺的流标识符相关的RST_STREAM标记来拒绝接收承诺流。

A PUSH_PROMISE frame modifies the connection state in two ways. The inclusion of a header block (Section 4.3) potentially modifies the state maintained for header compression. PUSH_PROMISE also reserves a stream for later use, causing the promised stream to enter the "reserved" state. A sender MUST NOT send a PUSH_PROMISE on a stream unless that stream is either "open" or "half closed (remote)"; the sender MUST ensure that the promised stream is a valid choice for a new stream identifier (Section 5.1.1) (that is, the promised stream MUST be in the "idle" state).

PUSH_PROMISE通过两种方式修改连接状态。这包括一个报头区块(章节4.3)可能修改压缩状态。
PUSH_PROMISE同样保留流后续使用，导致被推送的流进入到“保留”状态。发送端绝对不能在流上发送PUSH_PROMISE除非流是“打开”或
者“半封闭(远程)”状态；发送端绝对要保证被承诺的流对于新的流标示(章节5.1.1)来说是一个有效的选择(就是说，被承诺的流必须进入"空闲"状 态)。

Since PUSH_PROMISE reserves a stream, ignoring a PUSH_PROMISE frame causes the stream state to become indeterminate. A receiver MUST treat the receipt of a PUSH_PROMISE on a stream that is neither "open" nor "half closed (local)" as a connection error (Section 5.4.1) of type PROTOCOL_ERROR. Similarly, a receiver MUST treat the receipt of a PUSH_PROMISE that promises an illegal stream identifier (Section 5.1.1) (that is, an identifier for a stream that is not currently in the "idle" state) as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

由 于PUSH_PROMISE保留了一个流、忽略一个PUSH_PROMISE
帧都会导致流状态变得不确定。接收端接收到流状态不是“打开”或者“半封闭(本地)”的流的推送承诺帧必须作为类型为协议错误的连接错误(章节
5.4.1)处理。相似的，接收端必须对在一个非法标示(章节5.1.1)的流(即流的标识当前不在空闲状态)上建立的推送承诺作为类型为协议错误的连接 错误(章节5.4.1)处理。

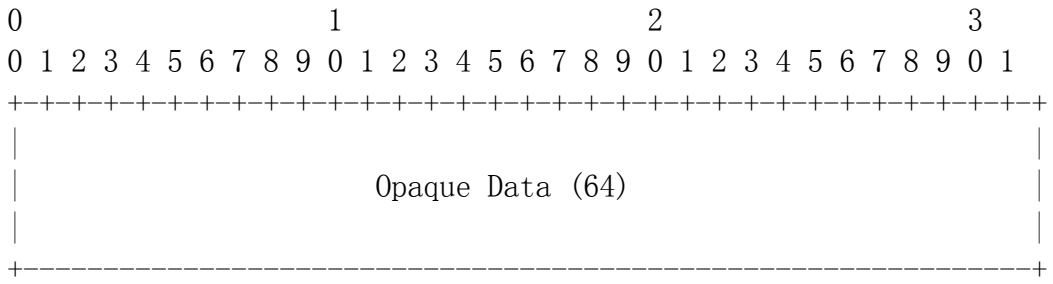
The PUSH_PROMISE frame includes optional padding. Padding fields and flags are identical to those defined for DATA frames (Section 6.1).

PUSH_PROMISE帧填充是可选的。填充字段及标记同数据帧(章节6.1)中定义。

6.7 PING PING帧

The PING frame (type=0x6) is a mechanism for measuring a minimal round trip time from the sender, as well as determining whether an idle connection is still functional. PING frames can be sent from any endpoint.

PING帧(type=0x6)是一种从发送端测量最小的RTT时间的机制，同样也是一种检测连接是否可用的方法。PING帧可以被任何终端发送。



In addition to the frame header, PING frames MUST contain 8 octets of data in the payload. A sender can include any value it chooses and use those bytes in any fashion.

除了帧报头之外，PING帧必须在载体中包含一个8字节长度的数据。发送端可以选择使用任何指并在任何时候使用。

Receivers of a PING frame that does not include an ACK flag MUST send a PING frame with the ACK flag set in response, with an identical payload. PING responses SHOULD be given higher priority than any other frame.

接收到不包含ACK标记的PING帧必须发送一个带有ACK比标记的PING帧响应，以及一个相同的载荷。PING响应应当设置比其他帧更高的优先级。

The PING frame defines the following flags:

- ACK (0x1):Bit 1 being set indicates that this PING frame is a PING response. An endpoint MUST set this flag in PING responses. An endpoint MUST NOT respond to PING frames containing this flag.

  PING frames are not associated with any individual stream. If a PING frame is received with a stream identifier field value other than 0x0, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

PING帧定义了以下标记：

ACK (0x1) :
位1表示PING帧是一个PING响应。终端必须在PING响应中设置此标记。终端绝对不能对包含此标记的PING帧做出响应。

PING帧捕鱼任何独立的流相关联。如果收到流标示字段不是0x0的PING帧，接收端必须响应一个类型为协议错误的连接错误。

Receipt of a PING frame with a length field value other than 8 MUST be treated as a connection error (Section 5.4.1) of type FRAME_SIZE_ERROR.

接收到字段长度不是8的PING帧必须作为类型为帧大小错误的连接错误处理。

6.8 GOAWAY 超时帧

The GOAWAY frame (type=0x7) informs the remote peer to stop creating streams on this connection. GOAWAY can be sent by either the client or the server. Once sent, the sender will ignore frames sent on any new streams with identifiers higher than the included last stream identifier. Receivers of a GOAWAY frame MUST NOT open additional streams on the connection, although a new connection can be established for new streams.

超时帧(type=0x7)通知远端对等端不要在这个连接上建立新流。超时帧可以由客户端或者服务端发送。一旦发送，发动端将忽略当前连接上新的和标示符大于上一个流的帧的发送。接收端接收到超时帧后绝对不能在这个连接上打开新的流，但是可以针对新的流创建一个新的连接。

The purpose of this frame is to allow an endpoint to gracefully stop accepting new streams, while still finishing processing of previously established streams. This enables administrative actions, like server maintainence.

这个帧的目的是允许终端优雅的停止接收新的流，但仍可以继续完成之前已经建立的流的处理。这使得管理员行为可用，如服务器维护。

There is an inherent race condition between an endpoint starting new streams and the remote sending a GOAWAY frame. To deal with this case, the GOAWAY contains the stream identifier of the last stream which was or might be processed on the sending endpoint in this connection. If the receiver of the GOAWAY has sent data on streams with a higher stream identifier than what is indicated in the GOAWAY frame, those streams are not or will not be processed. The receiver of the GOAWAY frame can treat the streams as though they had never been created at all, thereby allowing those streams to be retried later on a new connection.

在终
端启动新的流及远端发送超时帧之间有一个内在的竞争条件。为了处理这种情况，超时帧带

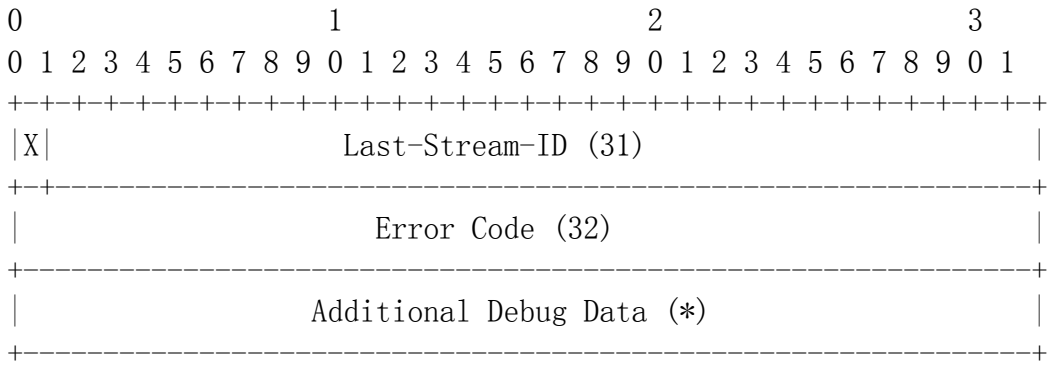
有当前连接中发送终端处理的最后一个流的标识。如果超时帧的接收端使
用了比指定的流更新的流，它们将不会被发送端处理，而且接收端可以认为这些流根本没有被创建(因此接收端可以稍后在新的连接上重新创建这些流)。

Endpoints SHOULD always send a GOAWAY frame before closing a connection so that the remote can know whether a stream has been partially processed or not. For example, if an HTTP client sends a POST at the same time that a server closes a connection, the client cannot know if the server started to process that POST request if the server does not send a GOAWAY frame to indicate what streams it might have acted on.

终端在关
闭一个连接之前总是应当发送一个超时帧，这样远端就能知道一个流是否已被部分处理。例如，如果一个HTTP客户端在服务端关闭连接的时候发送了一个
POST请求，如果服务端不发送一个指示它在哪里停止工作的超时帧，客户端将不知道这个POST请求是否已开始被处理。对于不规范的对等端，终端可以选择
不发送超时帧的情况下关闭连接。

An endpoint might choose to close a connection without sending GOAWAY for misbehaving peers.

对于行为不规范的对等端，终端可以选择不发送超时帧直接关闭连接。



The GOAWAY frame does not define any flags.

超时帧没有定义任何标记。

The GOAWAY frame applies to the connection, not a specific stream. An endpoint MUST treat a GOAWAY frame with a stream identifier other than 0x0 as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

超时帧适用于连接而不是特定的流。终端接收到流标识符不是0x0的超时帧必须作为类型为协议错误的连接错误(章节5.4.1)处理。

The last stream identifier in the GOAWAY frame contains the highest numbered stream identifier for which the sender of the GOAWAY frame might have taken some action on, or might yet take action on. All streams up to and including the identified stream might have been processed in some way. The last stream identifier can be set to 0 if no streams were processed.

超时帧中最后一个流的标识包含了接收端接收到并可能已经进行某些处理的流的标识的最大值。所有小于或等于此指定标识符的流都可能通过某种方式被处理。如果没有流被处理，最后流的标识符设置为0。

Note: In this context, "processed" means that some data from the stream was passed to some higher layer of software that might have taken some action as a result.

注意：这个案例中，“已处理”表示流中的某些数据已经被传到软件的更高的层并可能被进行某些处理。

If a connection terminates without a GOAWAY frame, the last stream identifier is effectively the highest possible stream identifier.

如果连接在没有超时帧的情况下终止，这个值有效的是最大的流标识符。

On streams with lower or equal numbered identifiers that were not closed completely prior to the connection being closed, re-attempting requests, transactions, or any protocol activity is not possible, with the exception of idempotent actions like HTTP GET, PUT, or DELETE. Any protocol activity that uses higher numbered streams can be safely retried using a new connection.

连接关闭前小于或等于标识符上的流没有完全关闭的，重试请求，交换，或者任何协议活动都是不可能的(例如HTTP
GET,PUT,或者删除等等幂行为例外)。任何使用更高的流数值的协议行为可以在新的连接上安全地重试。

Activity on streams numbered lower or equal to the last stream identifier might still complete successfully. The sender of a GOAWAY frame might gracefully shut down a connection by sending a GOAWAY frame, maintaining the connection in an open state until all in-progress streams complete.

小于或等于最后流标识符上的流的活动可能仍然能成功完成。超时帧的发送端可能通过发送超时帧优雅地关闭了连接，保持连接在打开状态直到正在处理的流全部处理完成。

An endpoint MAY send multiple GOAWAY frames if circumstances change. For instance, an endpoint that sends GOAWAY with NO_ERROR during graceful shutdown could subsequently encounter an condition that requires immediate termination of the connection. The last stream identifier from the last GOAWAY frame received

indicates which streams could have been acted upon. Endpoints MUST NOT increase the value they send in the last stream identifier, since the peers might already have retried unprocessed requests on another connection.

如果环境改变终端可能发送多个超时帧。例如，终
端发送带有NO_ERROR标记的超时帧来优雅关闭时随后可能遇到的情况需要理解终止连接。从最后一个超时帧接收到的最后的流标识符标识表示这些流可能已
经被处理了。终端绝对不能增加他们最后发送的流标识的值，因为对等端可能已经有在其他连接上未处理的重试请求。

A client that is unable to retry requests loses all requests that are in flight when the server closes the connection. This is especially true for intermediaries that might not be serving clients using HTTP/2. A server that is attempting to gracefully shut down a connection SHOULD send an initial GOAWAY frame with the last stream identifier set to 231-1 and a NO_ERROR code. This signals to the client that a shutdown is imminent and that no further requests can be initiated. After waiting at least one round trip time, the server can send another GOAWAY frame with an updated last stream identifier. This ensures that a connection can be cleanly shut down without losing requests.

服务端关闭连接时，终端无法重试请求的将丢失所有
正在发送的请求。尤其是针对中介端无法使用HTTP/2服务客户端的时候。无负担尝试优雅关闭连接时应当发送一个携带2^{31} -1
大小的流标识符和一个错误码的初始超时帧。这个信号对客户端来说意味着即将关闭连接并且不能建立更多请求了。在等待至少一个RTT时间之后，服务端能发送
另外一个带有更新后的最终流标识符超时帧。这个确保了连接能彻底的关闭而不用丢失请求。

After sending a GOAWAY frame, the sender can discard frames for streams with identifiers higher than the identified last stream. However, any frames that alter connection state cannot be completely ignored. For instance, HEADERS, PUSH_PROMISE and CONTINUATION frames MUST be minimally processed to ensure the state maintained for header compression is consistent (see Section 4.3); similarly DATA frames MUST be counted toward the connection flow control window. Failure to process these frames can cause flow control or header compression state to become unsynchronized.

在
发送超时帧后，发送端能丢弃流标识符大于最终流标识的流的帧。然而，任何修改流状态的帧不能被全部忽略。例如，报头帧、推送承诺帧和延续帧必须被最低限度
的处理来保证维持的报头压缩是连续的(章节4.3);类似的数据帧必须被计入连接流程控制窗口中。这些处理失败可能导致流量控制或者报头压缩状态不同步。

The GOAWAY frame also contains a 32-bit error code (Section 7) that contains the reason for closing the connection.

超时帧同样包含一个32位的错误码（章节7），里面包含了关闭连接的原因。

Endpoints MAY append opaque data to the payload of any GOAWAY frame. Additional debug data is intended for diagnostic purposes only and carries no semantic value. Debug information could contain security- or privacy-sensitive data. Logged or otherwise persistently stored debug data MUST have adequate safeguards to prevent unauthorized access.

终端可以在超时帧载体上附加不透明数据。额外的调试数据仅用来诊断没有语义值。调试信息可以包含安全或者隐私敏感的数据。登录或者其他持续存储的数据必须有足够的保障措施，以防止未经授权的访问。

6.9 WINDOW_UPDATE 窗口更新帧

The WINDOW_UPDATE frame (type=0x8) is used to implement flow control; see Section 5.2 for an overview.

WINDOW_UPDATE帧(type=0x8)用来实现流量控制；概述见章节5.2。

Flow control operates at two levels: on each individual stream and on the entire connection.

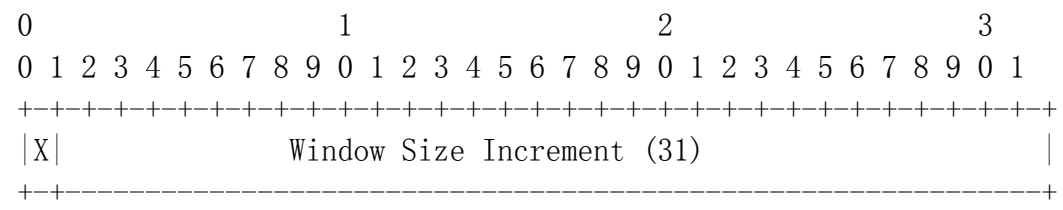
流量控制在两种层面上操作：每个单独的流或者整个连接。

Both types of flow control are hop-by-hop; that is, only between the two endpoints. Intermediaries do not forward WINDOW_UPDATE frames between dependent connections. However, throttling of data transfer by any receiver can indirectly cause the propagation of flow control information toward the original sender.

所有类型的流量控制都是逐跳的；就是说，只在两个终端之间作用。中介端不在依赖的连接上转接WINDOW_UPDATE帧。接收端对数据的显示可以直接导致流量控制信息的传播转到原始发送端。

Flow control only applies to frames that are identified as being subject to flow control. Of the frame types defined in this document, this includes only DATA frames. Frames that are exempt from flow control MUST be accepted and processed, unless the receiver is unable to assign resources to handling the frame. A receiver MAY respond with a stream error (Section 5.4.2) or connection error (Section 5.4.1) of type FLOW_CONTROL_ERROR if it is unable to accept a frame.

流量控制只适用于确定受流量控制影响的帧。文档中定义的帧类型中，只包括数据帧。不受流量控制的帧必须被接收和处理，除非接收端无法为帧分配资源。接收端如果无法接收帧，可以响应一个流错误(章节5.4.2)或者类型为流量控制错误的连接错误(章节5.4.1)。



The payload of a WINDOW_UPDATE frame is one reserved bit, plus an unsigned 31-bit integer indicating the number of bytes that the sender can transmit in addition to the existing flow control window. The legal range for the increment to the flow control window is 1 to 231 - 1 (0x7fffffff) bytes.

WINDOW_UPDATE帧的载体是一个保留字节，加上一个无符号31为整数表明发送端除了现有的流量控制窗口可以发送的字节数。留空控制窗口有效的增量范围是$1 至 2^{31}-1$(0x7fffffff) 字节。

The WINDOW_UPDATE frame does not define any flags.

WINDOW_UPDATE帧没有定义任何标记。

The WINDOW_UPDATE frame can be specific to a stream or to the entire connection. In the former case, the frame's stream identifier indicates the affected stream; in the latter, the value "0" indicates that the entire connection is the subject of the frame.

WINDOW_UPDATE可以专指某个流或者整个连接。在前者的情况下，帧的流标识符指的是被影响的流；在后者情况下，值"0"表示整个连接都受这个帧的影响。

WINDOW_UPDATE can be sent by a peer that has sent a frame bearing the END_STREAM flag. This means that a receiver could receive a WINDOW_UPDATE frame on a "half closed (remote)" or "closed" stream. A receiver MUST NOT treat this as an error, see Section 5.1.

WINDOW_UPDATE可以由一个已经发送带有END_STREAM标记的帧的对等端来发送。这意味着接收端可以在“半封闭(远程)”或者“关闭”的流上接收WINDOW_UPDATE帧。接收端绝对不能作为错误处理，见章节5.1

A receiver that receives a flow controlled frame MUST always account for its contribution against the connection flow control window, unless the receiver treats this as a connection error (Section 5.4.1). This is necessary even if the frame is in error. Since the sender counts the frame toward the flow control

window, if the receiver does not, the flow control window at sender and receiver can become different.

接收端收到受流量控制的帧必须总是计算流量对整个连接流量控制的影响量，除非接收端将这作为连接错误处理。即使帧出错这也是必须的。因为发送端将这个帧计入了流量控制窗口，如果接收端没有这样做，发送端和接收端的流量控制会不相同。

6.9.1 The Flow Control Window 流量控制窗口

Flow control in HTTP/2 is implemented using a window kept by each sender on every stream. The flow control window is a simple integer value that indicates how many bytes of data the sender is permitted to transmit; as such, its size is a measure of the buffering capacity of the receiver.

HTTP/2中流量控制是通过每个发送端在每个流上携带一个窗口来实现的。流量控制窗口是一个简单的整数值，指示发送端被允许传输的字节数；因此，它的大小是接收端的缓存能力的衡量。

Two flow control windows are applicable: the stream flow control window and the connection flow control window. The sender MUST NOT send a flow controlled frame with a length that exceeds the space available in either of the flow control windows advertised by the receiver. Frames with zero length with the END_STREAM flag set (that is, an empty DATA frame) MAY be sent if there is no available space in either flow control window.

流量控制窗口对流和连接的流量控制窗口都适用。发送端绝对不能发送超出接收端广播的流量控制窗口大小的可用空间长度的受流量控制影响的帧。在各个流量控制窗口中没有可用空间时，可以发送带有END_STREAM标记的长度为0的帧(例如，空数据帧)。

For flow control calculations, the 8 byte frame header is not counted.

流量控制计算中，8字节的帧报头不被计入。

After sending a flow controlled frame, the sender reduces the space available in both windows by the length of the transmitted frame.

在发送一个流量控制帧后，发送端在各个窗口中可用空间中减去发送的帧长度。

The receiver of a frame sends a WINDOW_UPDATE frame as it consumes data and frees up space in flow control windows. Separate WINDOW_UPDATE frames are sent for the stream and connection level flow control windows.

接收端发送一个WINDOW_UPDATE帧当它接受信息并释放了流量控制窗口的空间。单独的WINDOW_UPDATE帧用于流及连接层面的流量控制窗口中。

A sender that receives a WINDOW_UPDATE frame updates the corresponding window by the amount specified in the frame.

发送端收到WINDOW_UPDATE后按帧中指定的大小更新到正确的窗口。

A sender MUST NOT allow a flow control window to exceed 231 - 1 bytes. If a sender receives a WINDOW_UPDATE that causes a flow control window to exceed this maximum it MUST terminate either the stream or the connection, as appropriate. For streams, the sender sends a RST_STREAM with the error code of FLOW_CONTROL_ERROR code; for the connection, a GOAWAY frame with a FLOW_CONTROL_ERROR code.

发送端绝对不允许流量控制窗口超过 2^{31}-
1字节。如果发送端接收到WINDOW_UPDATE使得流量控制窗口超过这个最大值，它必须适当地终止这个流或者这个连接。对于流，发送
端发送一个带有流量控制错误的错误码的ST_STREAM帧；对于连接，发送一个带有流量控制错误码的超时帧。

Flow controlled frames from the sender and WINDOW_UPDATE frames from the receiver are completely asynchronous with respect to each other. This property allows a receiver to aggressively update the window size kept by the sender to prevent streams from stalling.

发送端发送的受流量控制的帧以及接收端收到的WINDOW_UPDATE帧是完全相互异步的。这种属性让接收端积极的更新发送端携带的窗口大小来防止流停转。

6.9.2 Initial Flow Control Window Size 流量控制窗口初始值

When an HTTP/2 connection is first established, new streams are created with an initial flow control window size of 65,535 bytes. The connection flow control window is 65,535 bytes. Both endpoints can adjust the initial window size for new streams by including a value for SETTINGS_INITIAL_WINDOW_SIZE in the SETTINGS frame that forms part of the connection preface. The connection flow control window can only be changed using WINDOW_UPDATE frames.

HTTP/2连接初次建立时，新的流创建的初始化流量控制大小是
65,535字节。连接的流量控制大小是65,535字节。两个终端都能通过在组成连接序言的设置帧中携带一个
SETTINGS_INITIAL_WINDOW_SIZE设置调整新流的初始化窗口大小。连接的流量控制窗口只能被WINDOWS_UPDATE帧修 改。

Prior to receiving a SETTINGS frame that sets a value for SETTINGS_INITIAL_WINDOW_SIZE, an endpoint can only use the default initial window size when sending flow controlled frames. Similarly, the connection flow

control window is set to the default initial window size until a WINDOW_UPDATE frame is received.

在收到设置帧指定SETTINGS_INITIAL_WINDOW_SIZE前，终端只能只有流量控制的默认窗口值。类似的，连接的流量控制窗口初始化时也是默认值知道收到WINDOW_UPDATE帧。

A SETTINGS frame can alter the initial flow control window size for all current streams. When the value of SETTINGS_INITIAL_WINDOW_SIZE changes, a receiver MUST adjust the size of all stream flow control windows that it maintains by the difference between the new value and the old value.

设置帧可以针对所有当前的流修改流量控制初始化大小。当SETTINGS_INITIAL_WINDOW_SIZE值改变时，接收端必须将根据新旧值调整其保留的所有流的窗口大小设置帧不能修改连接的流量控制窗口。

A change to SETTINGS_INITIAL_WINDOW_SIZE can cause the available space in a flow control window to become negative. A sender MUST track the negative flow control window, and MUST NOT send new flow controlled frames until it receives WINDOW_UPDATE frames that cause the flow control window to become positive.

SETTINGS_INITIAL_WINDOW_SIZE的修改可能导致流量控制窗口的可用空间为负数。发送端必须跟踪负数的流量控制窗口，并且绝对不能发送新的受流量控制的帧直到接收到窗口更新帧使得流量控制窗口变成正数。

For example, if the client sends 60KB immediately on connection establishment, and the server sets the initial window size to be 16KB, the client will recalculate the available flow control window to be -44KB on receipt of the SETTINGS frame. The client retains a negative flow control window until WINDOW_UPDATE frames restore the window to being positive, after which the client can resume sending.

例如，如果客户端在建立的连接上立即发送60KB的数据，而终端将初始的窗口大小设置成16KB,客户端将重新计算流量控制窗口的可用空间为-
44KB。终端将保持一个负数的流量控制窗口直到窗口更新帧恢复窗口到正数，这个时候客户端才能恢复数据发送。

A SETTINGS frame cannot alter the connection flow control window.

设置帧不能修改链接流量控制窗口。

An endpoint MUST treat a change to SETTINGS_INITIAL_WINDOW_SIZE that causes any flow control window to exceed the maximum size as a connection error (Section 5.4.1) of type FLOW_CONTROL_ERROR.

终端必须将SETTINGS_INITIAL_WINDOW_SIZE的修改导致流量控制窗口超过最大值的情况作为类型为流量控制错误的连接错误(章节 5.4.1)处理。

6.9.3 Reducing the Stream Window Size 减少流量窗口大小

A receiver that wishes to use a smaller flow control window than the current size can send a new SETTINGS frame. However, the receiver MUST be prepared to receive data that exceeds this window size, since the sender might send data that exceeds the lower limit prior to processing the SETTINGS frame.

接收端希望使用比当前大小更小的流量控制窗口可以发送一个新的设置帧。然而，接收端必须准备好接收超过窗口大小的数据，因为发送端在处理设置帧之前发送了低于最低下限的数据。

After sending a SETTINGS frame that reduces the initial flow control window size, a receiver has two options for handling streams that exceed flow control limits:

在发送减小初始化流量控制窗口大小的设置帧后，接收端有两种选择处理流超过流量限制的情况：

1. The receiver can immediately send RST_STREAM with FLOW_CONTROL_ERROR error code for the affected streams.
2. The receiver can accept the streams and tolerate the resulting head of line blocking, sending WINDOW_UPDATE frames as it consumes data.
3. 接收端可以针对受影响的流立即发送带有流量控制错误错误码的RST_STREAM帧。
4. 接收端如果在消耗数据可以接受流并且忍受报头阻塞的结果，并发送WINDOW_UPDATE帧。

6.10 CONTINUATION 延续帧

The CONTINUATION frame (type=0x9) is used to continue a sequence of header block fragments (Section 4.3). Any number of CONTINUATION frames can be sent on an existing stream, as long as the preceding frame is on the same stream and is a HEADERS, PUSH_PROMISE or CONTINUATION frame without the END_HEADERS flag set.

延续帧(type=0x9)用来延续一个报头区块(章节4.3)碎片序列。在现有流上可以发送到一个已存在延续帧，只要相同流上的前一阵是报头帧、推送承诺帧或者不带有END_HEADERS标记的延续帧。

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                   Header Block Fragment (*)                   ...
+---------------------------------------------------------------+
```

The CONTINUATION frame payload contains a header block fragment (Section 4.3).

延续帧实体包含一个报头区块碎片(章节4.3)。

The CONTINUATION frame defines the following flag:

- END_HEADERS (0x4):Bit 3 being set indicates that this frame ends a header block (Section 4.3). If the END_HEADERS bit is not set, this frame MUST be followed by another CONTINUATION frame. A receiver MUST treat the receipt of any other type of frame or a frame on a different stream as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

延续帧载体有以下字段：

- END_HEADERS (0x4) : 位3设置指示这个帧的报头区块的终止(章节4.3)。
  如果END_HEADERS位没有被设置，这个帧必须跟着另一个延续帧。接收到必须将收到其他类型的帧或者其他流上的帧错位类型为协议错误的连接错误(章节5.4.1)处理。

The CONTINUATION frame changes the connection state as defined in Section 4.3.

延续帧改变连接状态如章节4.3中定义。

CONTINUATION frames MUST be associated with a stream. If a CONTINUATION frame is received whose stream identifier field is 0x0, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

延续帧必须与流相关联。如果延续帧的相关流表示字段是0x0，终端必须响应一个类型为协议错误的连接错误。

A CONTINUATION frame MUST be preceded by a HEADERS, PUSH_PROMISE or CONTINUATION frame without the END_HEADERS flag set. A recipient that observes violation of this rule MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

延续帧必须跟在不带有END_HEADERS设置的报头帧、推送承诺帧或延续帧后面。终端接收到不符合此规则的必须响应一个类型为协议错误的连接错误(章节5.4.1)。

7.Error Codes 错误码

Error codes are 32-bit fields that are used in RST_STREAM and GOAWAY frames to convey the reasons for the stream or connection error.

错误码是32位字段，用在RST_STREAM和超时帧中用来标识流或者链接错误的原因。

Error codes share a common code space. Some error codes apply only to either streams or the entire connection and have no defined semantics in the other context.

错误码共享一个功能的代码空间。一些错误代码只适用于特定的条件，在某些帧类型没有定义的语义。

The following error codes are defined:

- NO_ERROR (0x0):The associated condition is not as a result of an error. For example, a GOAWAY might include this code to indicate graceful shutdown of a connection.
- PROTOCOL_ERROR (0x1):The endpoint detected an unspecific protocol error. This error is for use when a more specific error code is not available.
- INTERNAL_ERROR (0x2):The endpoint encountered an unexpected internal error.
- FLOW_CONTROL_ERROR (0x3):The endpoint detected that its peer violated the flow control protocol.
- SETTINGS_TIMEOUT (0x4):The endpoint sent a SETTINGS frame, but did not receive a response in a timely manner. See Settings Synchronization (Section 6.5.3).
- STREAM_CLOSED (0x5):The endpoint received a frame after a stream was half closed.
- FRAME_SIZE_ERROR (0x6):The endpoint received a frame that was larger than the maximum size that it supports.
- REFUSED_STREAM (0x7):The endpoint refuses the stream prior to performing any application processing, see Section 8.1.4 for details.
- CANCEL (0x8):Used by the endpoint to indicate that the stream is no longer needed.
- COMPRESSION_ERROR (0x9):The endpoint is unable to maintain the header compression context for the connection.
- CONNECT_ERROR (0xa):The connection established in response to a CONNECT request (Section 8.3) was reset or abnormally closed.
- ENHANCE_YOUR_CALM (0xb):The endpoint detected that its peer is exhibiting a behavior that might be generating excessive load.
- INADEQUATE_SECURITY (0xc):The underlying transport has properties that do not meet minimum security requirements (see Section 9.2).

定义了以下错误码：

- NO_ERROR (0) ：
  相关的条件并不是错误的结果。例如超时帧可以携带此错误码指示连接的平滑关闭。

- PROTOCOL_ERROR (1) :
  终端检测到一个不确定的协议错误。这个错误用在一个更具体的错误码不可用的时候。
- INTERNAL_ERROR (2) : 终端遇到意外的内部错误。
- FLOW_CONTROL_ERROR (3) : 终端检测到对等端违反了流量控制协议。
- SETTINGS_TIMEOUT (4) : 终端发送了设置帧，但是没有及时收到响应。见Settings Synchronization。
- STREAM_CLOSED (5) : 终端在流半封闭的时候收到帧。
- FRAME_SIZE_ERROR (6) : 终端收到大小超过最大尺寸的帧。
- REFUSED_STREAM (7) :
  终端拒绝流在它执行任何应用处理之前，详见Reliability（章节 8.1.4）
- CANCEL (8) : 终端使用这个标示某个流不再需要。
- COMPRESSION_ERROR (9) : 终端无法维持报头压缩上下文的连接
- CONNECT_ERROR (10) : 响应某个连接请求建立的连接被服为异常关闭。
- ENHANCE_YOUR_CALM (11) :
  终端检测出对等端在表现出可能会产生过大负荷的行为。
- INADEQUATE_SECURITY (12) :
  基础传输包含属性不满足文档或者终端申明的最小要求。

Unknown or unsupported error codes MUST NOT trigger any special behavior. These MAY be treated by an implementation as being equivalent to INTERNAL_ERROR.

未知或者不支持的错误码绝对不能触发任何特殊的行为。这可以被作为等同于INTERNAL_ERROR来实现。

8. HTTP Message Exchanges HTTP消息交换

HTTP/2 is intended to be as compatible as possible with current uses of HTTP. This means that, from the application perspective, the features of the protocol are largely unchanged. To achieve this, all request and response semantics are preserved, although the syntax of conveying those semantics has changed.

HTTP/2的目的是尽可能的兼容目前使用的HTTP。这意味着，从服务端或者客户端应用的角度来看，该协议的特点是不变的。为了实现这点，所有响应与请求的语义都将保留，尽管包含这些语义的语法已经改变。

Thus, the specification and requirements of HTTP/1.1 Semantics and Content [RFC7231], Conditional Requests [RFC7232], Range Requests [RFC7233], Caching [RFC7234] and Authentication [RFC7235] are applicable to HTTP/2. Selected portions of HTTP/1.1 Message Syntax and Routing [RFC7230], such as the HTTP and HTTPS URI schemes, are also applicable in HTTP/2, but the expression of those semantics for this protocol are defined in the sections below.

因此，HTTP/1.1语义与内容、有条件的请求、范围请求、缓存与验证定义的规范与要求同样适用于HTTP/2。HTTP/1.1消息语法与路由选定的内容，例如HTTP与HTTPS URI方案，也同样适用于HTTP/2，但是表达这些协议的语义在下面的章节定义。

8.1 HTTP Request/Response Exchange HTTP *请求/响应交换*

A client sends an HTTP request on a new stream, using a previously unused stream identifier (Section 5.1.1). A server sends an HTTP response on the same stream as the request.

客户端在一个新的流上发起HTTP请求，使用以前未使用的流标示。服务端在同个流上发起HTTP请求。

An HTTP message (request or response) consists of:

1. one HEADERS frame (followed by zero or more CONTINUATION frames) containing the message headers (see [RFC7230], Section 3.2), and
2. zero or more DATA frames containing the message payload (see [RFC7230], Section 3.3), and
3. optionally, one HEADERS frame, followed by zero or more CONTINUATION frames containing the trailer-part, if present (see [RFC7230], Section 4.1.2).

一个HTTP消息(请求或相应)包含：

1. 一个报头帧，后面跟着0个或多个延续帧(包含消息报头;见RFC7230),以及
2. 0个或多个数据帧(包含消息载荷，见RFC7230章节3.3),以及
3. 一个可选的版头，后面跟着0个或多个延续帧(如果存在，包含尾部部分，见RFC7230章节4.1.2)

The last frame in the sequence bears an END_STREAM flag, noting that a HEADERS frame bearing the END_STREAM flag can be followed by CONTINUATION frames that carry any remaining portions of the header block.

序列中的最后一帧具有END_STREAM标记，但是包含END_STREAM标记的报头帧后面可以跟着包含任意报报头分的延续帧。

Other frames (from any stream) MUST NOT occur between either HEADERS frame and any CONTINUATION frames that might follow.

其他帧(来自任何流的)绝对不能出现在任意报头帧和延续帧(如果存在)之间,也不能出现在延续帧中间。

Otherwise, frames MAY be interspersed on the stream between these frames, but those frames do not carry HTTP semantics. In particular, HEADERS frames (and any CONTINUATION frames that follow) other than the first and optional last frames in this sequence do not carry HTTP semantics.

否则，流上的这些帧可能被打散，但是那些帧并不包含HTTP语义。特别是，报头帧(及任何跟在后面的延续帧)序列中第一帧及可选的最后一帧以外的并不包含HTTP语义。

Trailing header fields are carried in a header block that also terminates the stream. That is, a sequence starting with a HEADERS frame, followed by zero or more CONTINUATION frames, where the HEADERS frame bears an END_STREAM flag. Header blocks after the first that do not terminate the stream are not part of an HTTP request or response.

报头区块中的报头尾部字段同样终止流。就是说，一个报头帧开始的序列，后面跟着0个或者多个带有END_STREAM标记的延续帧。第一个不终止流后面的报头区块不是当前HTTP请求与响应的一部分。

An HTTP request/response exchange fully consumes a single stream. A request starts with the HEADERS frame that puts the stream into an "open" state. The request ends with a frame bearing END_STREAM, which causes the stream to become "half closed (local)" for the client and "half closed (remote)" for the server. A response starts with a HEADERS frame and ends with a frame bearing END_STREAM, which places the stream in the "closed" state.

一个HTTP
请求/响应的数据交换在同一个流上进行。一个请求由是流进入打开状态的报头帧开始，并由一个携带使流对客户端进入半封闭的END_STREAM标记的帧结
束，另外可选的后面可以跟着延续帧，使流进入关闭状态。一个响应从HEADERS帧开始并且结束于一个放在封闭状态中支撑END_STREAM的帧

8.1.1 Informational Responses 响应信息

The 1xx series of HTTP response status codes ([RFC7231], Section 6.2) are not supported in HTTP/2.

1xx系列的HTTP响应状态码在HTTP/2中不支持。

The most common use case for 1xx is using an Expect header field with a 100-continue token (colloquially, "Expect/continue") to indicate that the client expects a 100 (Continue) non-final response status code, receipt of which indicates that the client should continue sending the request body if it has not already done so.

1xx最常见的用法是使用一个带有100-
继续token(通俗的，“期待/继续”)的期望报头字段来表明客户端期望一个100(继续)非最终响应的状态码，收到这个表示客户端如果没有发送完应当继续发送请求正文。

Typically, Expect/continue is used by clients wishing to avoid sending a large amount of data in a request body, only to have the request rejected by the origin server, thereby leaving the connection potentially unusable.

通常来说，期望/继续帧被客户端用来希望避免在请求正文中发送大量数据，只用来让请求被源服务器拒绝(因此让连接可能不可用)。

HTTP/2 does not enable the Expect/continue mechanism; if the server sends a final status code to reject the request, it can do so without making the underlying connection unusable.

HTTP/2不支持期望/继续机制；如果服务端发送一个最终状态码来拒绝请求，它可以在不需要使当前连接不可用的情况下做到这个。

Note that this means HTTP/2 clients sending requests with bodies may waste at least one round trip of sent data when the request is rejected. This can be mitigated by restricting the amount of data sent for the first round trip by bandwidth-constrained clients, in anticipation of a final status code.

需要注意的是这意味着，HTTP/2客户端发送带有请求正文的请求在连接被拒绝时可能浪费至少一个发送数据的RTT时间。这可以通过限制带宽受限的客户端第一个RTT的数据大小来缓和，通过预期的最终状态码实现。

Other defined 1xx status codes are not applicable to HTTP/2. For example, the semantics of 101 (Switching Protocols) aren't suitable to a multiplexed protocol. Likewise, 102 (Processing) [RFC2518] is no longer necessary to ensure connection liveness, because HTTP/2 has a separate means of keeping the connection alive. The use of the 102 (Processing) status code for progress reporting has since been deprecated and is not retained.

其他定义的1xx状态码也不适用于HTTP/2。例如，101(转换协议)的语义不适用于多路复用协议。同样的，102(处理中)也不在需要确保连接的活跃性，因为HTTP/2有单独的保持连接可用的方式。

This difference between protocol versions necessitates special handling by intermediaries that translate between them:

- An intermediary that translates HTTP/1.1 requests to HTTP/2 MUST generate a 100 (Continue) response if a received request includes and Expect header field with a 100-continue token ([RFC7231], Section 5.1.1), unless it can

immediately generate a final status code. It MUST NOT forward the 100-continue expectation in the request header fields.
- An intermediary that translates HTTP/2 to HTTP/1.1 MAY add an Expect header field with a 100-continue expectation when forwarding a request that has a body; see [RFC7231], Section 5.1.1 for specific requirements.
- An intermediary that gateways HTTP/2 to HTTP/1.1 MUST discard all other 1xx informational responses.

这些不同协议版本之间的差异需要被中介端在转换时特殊处理：

- 转换HTTP/1.1到HTTP/2的中介网关如果收到请求包含带有100-继续token(RFC7231,章节5.1.1)的期望报头字段，必须生成一个100(继续)响应，除非它能马上生成一个最终状态码。绝对不能转发请求报头中的100-继续期望字段。
- 转换HTTP/2到HTTP/1.1的中介网关在转发一个带有正文的请求时可以添加一个带有100-继续的期望报头字段。特定要求见RFC7231.
- 转换HTTP/2到HTTP/1.1的中介网关必须丢弃所有1xx以外的响应信息。

8.1.2 HTTP Header Fields HTTP报头字段

HTTP header fields carry information as a series of key-value pairs. For a listing of registered HTTP headers, see the Message Header Field Registry maintained at https://www.iana.org/assignments/message-headers.

HTTP报头字段携带一系列键-值对的信息。对于注册的HTTP报头列表，见https://www.iana.org/assignments/message-headers中消息报头字段注册。

While HTTP/1.x used the message start-line (see [RFC7230], Section 3.1) to convey the target URI and method of the request, and the status code for the response, HTTP/2 uses special pseudo-headers beginning with ':' character (ASCII 0x3a) for this purpose.

HTTP/1.x使用消息起始线(见RFC7230,章节3.1)来传达URI目标和请求的方法、以及响应的状态码，HTTP/2使用特殊的':'(ASCII 0x3a)开头的伪头字符来实现这个目的。

Just as in HTTP/1.x, header field names are strings of ASCII characters that are compared in a case-insensitive fashion. However, header field names MUST be converted to lowercase prior to their encoding in HTTP/2. A request or response containing uppercase header field names MUST be treated as malformed (Section 8.1.2.5).

正如HTTP/1.x中，报头字段名称是ASCII字符，且不区分大小写。然而，HTTP/2中报头字段名称必须转成使用同样编码的小写字符。带有大写报头字段的请求或者响应必须被认为是不规范的(章节8.1.2.5)。

This means that an intermediary transforming an HTTP/1.x message to HTTP/2 will need to remove any header fields nominated by the Connection header field, along with the Connection header field itself. Such intermediaries SHOULD also remove other connection-specific header fields, such as Keep-Alive, Proxy-Connection, Transfer-Encoding and Upgrade, even if they are not nominated by Connection.

这意味着中介端转换一个
HTTP/1.x消息到HTTP/2需要移除由连接报头字段指定的任何报头字段，包含连接报头字段本身。这样的中介端同样应当移除其他连接特定的报头字 段，例如Keep-Alive、Proxy-Connection、Transfer-Encoding和Upgrade，即便它们不是由连接指定的。

One exception to this is the TE header field, which MAY be present in an HTTP/2 request, but when it is MUST NOT contain any value other than "trailers".

一个例外是TE报头字段，这个可能在 HTTP/2
请求中保留，但是它不能包含“trailers”以外的值。

Note: : HTTP/2 purposefully does not support upgrade to another protocol. The handshake methods described in Section 3 are believed sufficient to negotiate the use of alternative protocols.

Note: :
HTTP/2不支持升级到其他协议。3章节中描述的握手协议被认为足够用来作为替代协议使用。

8.1.2.1 Request Header Fields 请求报头字段

HTTP/2 defines a number of pseudo header fields starting with a colon ':' character that carry information about the request target:

- The :method header field includes the HTTP method ([RFC7231], Section 4).
- The :scheme header field includes the scheme portion of the target URI ([RFC3986], Section 3.1). :scheme is not restricted to http and https schemed URIs. A proxy or gateway can translate requests for non-HTTP schemes, enabling the use of HTTP to interact with non-HTTP services.
- The :authority header field includes the authority portion of the target URI ([RFC3986], Section 3.2). The authority MUST NOT include the deprecated userinfo subcomponent for http or https schemed URIs. To ensure that the HTTP/1.1 request line can be reproduced accurately, this header field MUST be omitted when translating from an HTTP/1.1 request that has a request target in origin or asterisk form (see [RFC7230], Section 5.3). Clients that generate HTTP/2 requests directly SHOULD instead omit the Host header field. An intermediary that converts an HTTP/2 request to

HTTP/1.1 MUST create a Host header field if one is not present in a request by copying the value of the :authority header field.

- The :path header field includes the path and query parts of the target URI (the path-absolute production from [RFC3986] and optionally a '?' character followed by the query production, see [RFC3986], Section 3.3 and [RFC3986], Section 3.4). This field MUST NOT be empty; URIs that do not contain a path component MUST include a value of '/', unless the request is an OPTIONS request in asterisk form, in which case the :path header field MUST include '*'.

HTTP/2定义了一个以字符“:”开头的报头域，包含目标请求的信息：

- :method 报头字段包含了HTTP方法
- :scheme字段包含了目标URI方案部分。
  :scheme并不是被限制于http和https类的URIS。代理或者网路关口可以转化非HTTP体系的请求使其可以于非HTTP得请求互动
- :authority
  报头字段包含了目标URI的权限部分。这个权限绝对不能包含http:或者https:
  URIs的废弃的用户信息子成份。
  为了保证HTTP/1.1请求行能被精确复制，当原始请求有请求目标或者星号形式(见[
  http-p1
  ]，5.3节)的HTTP/1.1请求进行转换时这个字段必须被忽略。客户端直接生成HTTP/2请求的相反应该忽略Host报头字段。如果其中一个请求
  没有Host字段，中介端将HTTP/2请求转换为HTTP/1.1请求的时候必须复制:authority字段的值来生成Host字段。
- :path
  字段包含目标URI的路径及查询部分(绝对路径由[RFC3986]以及可选的‘?’字符后面跟着查询词组成见xx)。这个字段绝对不能为空；URI不包
  含path组件的必须包含一个'/'值，除非请求是一个星号形式的可选请求：这种情况下:path报头字段必须包含""。是一个星号形式的可选请求：这种
  情况下:path报头字段必须包含""。

8.1.2.2 Response Header Fields 响应报头字段

A single :status header field is defined that carries the HTTP status code field (see [RFC7231], Section 6). This header field MUST be included in all responses, otherwise the response is malformed (Section 8.1.2.5).

定义了一个单独的:status
报头字段携带了HTTP状态码信息(见RFC7231,章节6)。这个报头字段必须包含在所有的响应中，否则响应就是不规范的(章节8.1.2.5)。

HTTP/2 does not define a way to carry the version or reason phrase that is included in an HTTP/1.1 status line.

HTTP/2没有定义一种方式携带HTTP/1.1状态行中的版本或原因短语信息。

8.1.2.3 Header Field Ordering 报头字段顺序

HTTP Header Compression [COMPRESSION] does not preserve the order of header fields, because the relative order of header fields with different names is not important. However, the same header field can be repeated to form a list (see [RFC7230], Section 3.2.2), where the relative order of header field values is significant. This repetition can occur either as a single header field with a comma-separated list of values, or as several header fields with a single value, or any combination thereof. Therefore, in the latter case, ordering needs to be preserved before compression takes place.

HTTP报头压缩并不保留报头字段的顺序，因为不同名称的字段的相对位置并不重要。然而，当相同字段重复组成一个列表的时候(见RFC7230，章节3.2.2)，报头字段值的相对位置就有意义。这种重复会出现在以逗号分隔的单个报头字段
值列表中，或者作为几个报头字段的单一值，或他们的任何组合。因此，再后者情况下，报头需在再压缩前保留字段顺序。

To preserve the order of multiple occurrences of a header field with the same name, its ordered values are concatenated into a single value using a zero-valued octet (0x0) to delimit them.

为了保留同个名称的报头字段出现多次的顺序，他们的顺序使用单个以零值字节(0x0)分隔的值连接来保留。

After decompression, header fields that have values containing zero octets (0x0) MUST be split into multiple header fields before being processed.

解压缩后，报头字段包含有0字节的必须分割成多个报头字段之后才能进一步处理。

For example, the following HTTP/1.x header block:

例如，下面的HTTP/1.x报头区块：

```
Content-Type: text/html
Cache-Control: max-age=60, private
Cache-Control: must-revalidate
```

contains three Cache-Control directives; two directives in the first Cache-Control header field, and the third directive in the second Cache-Control field. Before compression, they would need to be converted to a form similar to this (with 0x0 represented as '\0'):

包含三个缓存控制指令；两个在第一个缓存控制报头字段，一个在第二个缓存控制字段。在压缩前，它们需要转成类似格式("\0"表示0x0):

```
cache-control: max-age=60, private\0must-revalidate
               content-type: text/html
```

Note here that the ordering between Content-Type and Cache-Control is not preserved, but the relative ordering of the Cache-Control directives - as well as the fact that the first two were comma-separated, while the last was on a different line - is.

注意这里内容类型与缓存控制的顺序是不保留的，但是缓存控制指令的相对顺序——事实上也就是前两个是用逗号分隔，而后一个是在不一样的行——是保留的。

Header fields containing multiple values MUST be concatenated into a single value unless the ordering of that header field is known to be not significant.

报头字段包含多个值必须组成单个值，除非报头字段的顺序是不重要的。

The special case of set-cookie - which does not form a comma-separated list, but can have multiple values - does not depend on ordering. The set-cookie header field MAY be encoded as multiple header field values, or as a single concatenated value.

特殊情况是设置cookie——不需要形成一个逗号分隔的列表，但是可以有多个值——不需要依赖顺序。设置cookie字段可以被编码成多行报头字段值，或者单个的连接值。

8.1.2.4 Compressing the Cookie Header Field 报头Cookie字段压缩

The Cookie header field [COOKIE] can carry a significant amount of redundant data.

Cookie报头字段可以携带大量的冗余数据。

The Cookie header field uses a semi-colon (";") to delimit cookie-pairs (or "crumbs"). This header field doesn't follow the list construction rules in HTTP (see [RFC7230], Section 3.2.2), which prevents cookie-pairs from being separated into different name-value pairs. This can significantly reduce compression efficiency as individual cookie-pairs are updated.

Cookie字段使用“;”来分割cookie-对(或叫面包屑)。这报头字段不遵循HTTP中的构建规则(见RFC7230，章节3.2.2)，以防cookie-对被分隔成不同的键值对。单个cookie-对更新的时候能显著提升压缩效率。

To allow for better compression efficiency, the Cookie header field MAY be split into separate header fields, each with one or more cookie-pairs. If there are multiple Cookie header fields after decompression, these MUST be concatenated into a single octet string using the two octet delimiter of 0x3B, 0x20 (the ASCII string "; ").

为了更好的压缩效率，Cookie字段可以被分隔成多个报头字段，每个包含一个或者多个cookie对。如果解压后有多个Cookie报头字段，他们必须由两个字节的0x3B, 0x20(ASCII";")连接成单个字段。

The Cookie header field MAY be split using a zero octet (0x0), as defined in Section 8.1.2.3. When decoding, zero octets MUST be replaced with the cookie delimiter ("; ").

Coookie报头字段也可以使用0值字节(0x0)来分割，同章节8.1.2.3中定义。当解码的时候，0值字节必须被替换成Cookie的分隔符(";")。

Therefore, the following sets of Cookie header fields are semantically equivalent, though the final form might appear in a different order after compression and decompression.

因此，下列Cookie报头字段集合在语言上是相等，尽管压缩和编码后在最终形式上可能看起来不同。

```
cookie: a=b; c=d; e=f

cookie: a=b\0c=d; e=f

cookie: a=b
cookie: c=d
cookie: e=f
```

8.1.2.5 Malformed Messages 不规范的消息

A malformed request or response is one that uses a valid sequence of HTTP/2 frames, but is otherwise invalid due to the presence of prohibited header fields, the absence of mandatory header fields, or the inclusion of uppercase header field names.

不规范的请求或者响应是一个使用了有效序列的HTTP/2帧，但是使用了禁止的报头字段、必须字段缺失或者字段名称使用了大写。

A request or response that includes an entity body can include a content-length header field. A request or response is also malformed if the value of a content-

length header field does not equal the sum of the DATA frame payload lengths that form the body.

包含实体的请求或者响应可以保护一个实体内容长度的报头字段。如果内容长度报头字段的值不等于组成实体的数据帧载荷长度，同样是不规范的。

Intermediaries that process HTTP requests or responses (i.e., any intermediary not acting as a tunnel) MUST NOT forward a malformed request or response.

中介端处理HTTP请求或者响应(除了用来作为隧道的所有中介端)绝对不能转发一个不规范的请求或者响应。

Implementations that detect malformed requests or responses need to ensure that the stream ends. For malformed requests, a server MAY send an HTTP response prior to closing or resetting the stream. Clients MUST NOT accept a malformed response. Note that these requirements are intended to protect against several types of common attacks against HTTP; they are deliberately strict, because being permissive can expose implementations to these vulnerabilities.

中介端检测到不规范的请求或者响应必须保证流已经终止。对于不规范的请求，服务端可以发送之前提到的响应来关闭或者重置流。客户端绝对不能接收一个不规范的响应。请注意，这些要求是为了防止一些针对HTTP的常见攻击；故意这么严格，是
因为允许这些情况的话可能会暴露这些漏洞的实现。

8.1.3 Examples 示例

This section shows HTTP/1.1 requests and responses, with illustrations of equivalent HTTP/2 requests and responses.

这个部分介绍了HTTP/1.1的请求与响应，并带有HTTP/2请求与响应的插图。

An HTTP GET request includes request header fields and no body and is therefore transmitted as a single HEADERS frame, followed by zero or more CONTINUATION frames containing the serialized block of request header fields. The HEADERS frame in the following has both the END_HEADERS and END_STREAM flags set; no CONTINUATION frames are sent:

一个带有请求报头字段但没有正文的HTTP
GET请求将被转换成一个单独的报头帧，后面跟着0个或者多个包含序列化的报头字段区块的延续帧。序列中最后一个报头帧将有END_HEADERS和END_STREAM标记。

```
  GET /resource HTTP/1.1            HEADERS
  Host: example.org          ==>      + END_STREAM
  Accept: image/jpeg                  + END_HEADERS
```

```
                                        :method = GET
                                        :scheme = https
                                        :path = /resource
                                        host = example.org
                                        accept = image/jpeg
```

Similarly, a response that includes only response header fields is transmitted as a HEADERS frame (again, followed by zero or more CONTINUATION frames) containing the serialized block of response header fields.

相似的，只带有报头字段的响应将被转换成一个报头帧(同样的，后面跟着0个护着多个延续帧)，且包含序列化的响应报头字段区块。序列中最后一个报头帧将有END_HEADERS和END_STREAM标记。

```
  HTTP/1.1 304 Not Modified          HEADERS
  ETag: "xyzzy"                ==>     + END_STREAM
  Expires: Thu, 23 Jan ...             + END_HEADERS
                                         :status = 304
                                         etag: "xyzzy"
                                         expires: Thu, 23 Jan ...
```

An HTTP POST request that includes request header fields and payload data is transmitted as one HEADERS frame, followed by zero or more CONTINUATION frames containing the request header fields, followed by one or more DATA frames, with the last CONTINUATION (or HEADERS) frame having the END_HEADERS flag set and the final DATA frame having the END_STREAM flag set:

带有报头和载荷数据的POST
HTTP请求将被转换成一个报头帧，后面跟着0个或者多个带有请求报头字段的延续帧，同时后面跟着一个或者多个数据帧。延续帧或者报头帧的最后一帧有END_HEADERS标记，最后一个数据帧拥有END_STREAM标记。

```
 POST /resource HTTP/1.1           HEADERS
  Host: example.org          ==>     - END_STREAM
  Content-Type: image/jpeg           + END_HEADERS
  Content-Length: 123                  :method = POST
                                       :scheme = https
  {binary data}                        :path = /resource
                                       :authority = example.org
                                       content-type = image/jpeg
                                       content-length = 123

                                   DATA
                                     + END_STREAM
```

```
{binary data}
```

Note that data contributing to any given header field could be spread between header block fragments. The allocation of header fields to frames in this example is illustrative only.

需要注意的是任何给定的报头字段的相关数据都可能被分割到报头区块碎片中。这个例子中的报头字段配置仅仅作为说明。

A response that includes header fields and payload data is transmitted as a HEADERS frame, followed by zero or more CONTINUATION frames, followed by one or more DATA frames, with the last DATA frame in the sequence having the END_STREAM flag set:

带有报头字段及载荷数据的响应将被转换成一个报头帧，后面跟着0个或多个延续帧，另外后面跟着一个或多个数据帧，序列中的最后一个数据帧拥有END_STREAM标记。

```
HTTP/1.1 200 OK                    HEADERS
Content-Type: image/jpeg    ==>      - END_STREAM
Content-Length: 123                  + END_HEADERS
                                       :status = 200
{binary data}                          content-type = image/jpeg
                                       content-length = 123

                                   DATA
                                     + END_STREAM
                                   {binary data}
```

Trailing header fields are sent as a header block after both the request or response header block and all the DATA frames have been sent. The HEADERS frame starting the trailers header block has the END_STREAM flag set.

所有的请求或者响应报头区块以及所有的数据帧发送之后，尾报头字段作为一个报头区块发送。带有尾部的报头/延续帧序列包含一个带有END_HEADERS及END_STREAM标记的终止帧。

```
HTTP/1.1 200 OK                    HEADERS
Content-Type: image/jpeg    ==>      - END_STREAM
Transfer-Encoding: chunked           + END_HEADERS
Trailer: Foo                           :status         = 200
                                       content-length = 123
123                                    content-type    = image/jpeg
{binary data}                          trailer         = Foo
0
Foo: bar                           DATA
```

```
  - END_STREAM
{binary data}

HEADERS
  + END_STREAM
  + END_HEADERS
    foo: bar
```

8.1.4 Request Reliability Mechanisms in HTTP/2 HTTP/2响应可靠性机制

In HTTP/1.1, an HTTP client is unable to retry a non-idempotent request when an error occurs, because there is no means to determine the nature of the error. It is possible that some server processing occurred prior to the error, which could result in undesirable effects if the request were reattempted.

在HTTP/1.1中，HTTP客户端在发送错误时不能重试一个非幂等的请求，因为没有方式来确定错误的性质。也许在错误出现前服务端已经对信息有所处理，以至于如果重新尝试发送会得到一些不想要的结果。

HTTP/2 provides two mechanisms for providing a guarantee to a client that a request has not been processed:

- The GOAWAY frame indicates the highest stream number that might have been processed. Requests on streams with higher numbers are therefore guaranteed to be safe to retry.
- The REFUSED_STREAM error code can be included in a RST_STREAM frame to indicate that the stream is being closed prior to any processing having occurred. Any request that was sent on the reset stream can be safely retried.

HTTP/2提供了两种机制来确保让客户端知道请求没有被处理:

- 超时帧指示了流可能被处理的最大流流标示。在更大数字的流上的请求可以保证安全的重试。
- RST_STREAM帧中可以包含REFUSED_STREAM错误码来指示流由于之前的处理正在关闭。重置流上的任何请求都可以安全重试。

Requests that have not been processed have not failed; clients MAY automatically retry them, even those with non-idempotent methods.

未经处理的请求且没有失败；客户可以自动重试，甚至包括那些非幂等元的方法。

A server MUST NOT indicate that a stream has not been processed unless it can guarantee that fact. If frames that are on a stream are passed to the

application layer for any stream, then REFUSED_STREAM MUST NOT be used for that stream, and a GOAWAY frame MUST include a stream identifier that is greater than or equal to the given stream identifier.

服务端绝对不能表示一个流未被处理除非它能确保这个事实。如果流上的帧被传递给应用层的任何流，绝对不能在这个流上使用REFUSED_STREAM，而且一个超时帧必须包含一个大于或等于给定流表示的标识符。

In addition to these mechanisms, the PING frame provides a way for a client to easily test a connection. Connections that remain idle can become broken as some middleboxes (for instance, network address translators, or load balancers) silently discard connection bindings. The PING frame allows a client to safely test whether a connection is still active without sending a request.

除了这些机制，PING帧给客户的提供了一种方式来简单测试连接。保持空闲的连接可能被一些中间件(例如网络地址翻译或负载均衡器)静默丢弃连接绑定而打破。PING帧允许客户端在无需发送请求的情况下安全地测试连接是否依旧激活。

8.2 Server Push 服务端推送

HTTP/2 enables a server to pre-emptively send (or "push") one or more associated responses to a client in response to a single request. This feature becomes particularly helpful when the server knows the client will need to have those responses available in order to fully process the response to the original request.

HTTP/2允许服务端针对客户端一个单独的请求，主动的发送(或推送)一个或者多个相关的响应。这种特定在服务端知道客户端需要这些响应来完整的处理最初的请求的时候特别有用。

Pushing additional responses is optional, and is negotiated between individual endpoints. The SETTINGS_ENABLE_PUSH setting can be set to 0 to indicate that server push is disabled.

推送额外的响应是可选的，并且由各个单独的终端之间协商。SETTINGS_ENABLE_PUSH设置设置为0来标识服务端推送是无效的。

Because pushing responses is effectively hop-by-hop, an intermediary could receive pushed responses from the server and choose not to forward those on to the client. In other words, how to make use of the pushed responses is up to that intermediary. Equally, the intermediary might choose to push additional responses to the client, without any action taken by the server.

因为推送的响应是有效地逐跳，中介端接从服务端接收到推送响应的可以选择不转发这些到客户端。也就是说，如何使用推送响应取决于这些中介端。相等的，中介可能选择不推送的额外的响应给客户端，不需要服务端进行任何操作。

A client cannot push. Thus, servers MUST treat the receipt of a PUSH_PROMISE frame as a connection error (Section 5.4.1) of type PROTOCOL_ERROR. Clients MUST reject any attempt to change the SETTINGS_ENABLE_PUSH setting to a value other than 0 by treating the message as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

客户端不能推送。因此，服务端收到客户端的PUSH_PROMISE帧必须作为连接错误(章节5.4.1)处理。客户端必须拒绝任何尝试修改SETTINGS_ENABLE_PUSH设置值为0以外的值，并将这消息作为类型为协议错误的连接错误(章节5.4.1)处理。

A server can only push responses that are cacheable (see [RFC7234], Section 3); promised requests MUST be safe (see [RFC7231], Section 4.2.1) and MUST NOT include a request body.

服务端只能推送可以被缓存的响应(见RFC7234,章节3);被承诺的请求必须是安全的(见RFC7231,章节4.2.1)，而且绝对不能包含一个请求主体。

8.2.1 Push Requests 推送请求

Server push is semantically equivalent to a server responding to a request; however, in this case that request is also sent by the server, as a PUSH_PROMISE frame.

服务端推送语义上等同于服务端响应一个请求；然而，这种情况下请求也是由服务端发送的，作为一个PUSH_PROMISE帧。

The PUSH_PROMISE frame includes a header block that contains a complete set of request header fields that the server attributes to the request. It is not possible to push a response to a request that includes a request body.

PUSH_PROMISE包含了一个报头区块，含有完整的服务端属性请求报头字段。不可能对带有请求实体的请求进行推送。

Pushed responses are always associated with an explicit request from the client. The PUSH_PROMISE frames sent by the server are sent on that explicit request's stream. The PUSH_PROMISE frame also includes a promised stream identifier, chosen from the stream identifiers available to the server (see Section 5.1.1).

推送的响应总是与客户端的一个明确的请求相关。服务端在这个明确的请求流上发送PUSH_PROMISE帧。PUSH_PROMISE帧一般包含被承诺的流标识符，从可用的服务端流标识符中选择(见章节5.1.1)。

The header fields in PUSH_PROMISE and any subsequent CONTINUATION frames MUST be a valid and complete set of request header fields (Section 8.1.2.1). The server MUST include a method in the :method header field that is safe and cacheable. If a client receives a PUSH_PROMISE that does not include a complete and valid set of header fields, or the :method header field identifies a method that is not safe, it MUST respond with a stream error (Section 5.4.2) of type PROTOCOL_ERROR.

在
PUSH_PROMISE或者任何其他延续的帧中的报头字段必须是完整的请求报头字段(章节8.1.2.1)。服务端必须在:method字段中包含一个
安全而且可缓存的方法。如何客户端收到不包含完整而且有效的报头字段的PUSH_PROMISE帧、或者:method表示的方法不是安全的，客户端必须
响应一个类型为协议错误的流错误(章节5.4.2)。

The server SHOULD send PUSH_PROMISE (Section 6.6) frames prior to sending any frames that reference the promised responses. This avoids a race where clients issue requests prior to receiving any PUSH_PROMISE frames.

服务端应当在发送任何被承诺的响应之前发送一个PUSH_PROMISE帧(章节6.6)。这避免了客户端在收到任何PUSH_PROMISE帧前发出请求而出现的竞赛。

For example, if the server receives a request for a document containing embedded links to multiple image files, and the server chooses to push those additional images to the client, sending push promises before the DATA frames that contain the image links ensures that the client is able to see the promises before discovering embedded links. Similarly, if the server pushes responses referenced by the header block (for instance, in Link header fields), sending the push promises before sending the header block ensures that clients do not request them.

例
如：如果服务端收到文档请求包含多个嵌入式的图像链接，而且服务端选择推送那些额外的图像给客户端，再数据帧前发送push
promises能确保客户端能够在发现内嵌链接前看到这些承诺。类似的，如果服务端推送与报头区块(例如，在Link报头域)相关的响应，再发送报头区
块前推送承诺能确保客户端不请求它们。

PUSH_PROMISE frames MUST NOT be sent by the client.

PUSH_PROMISE帧绝对不能由客户端发送。

PUSH_PROMISE frames can be sent by the server in response to any client-initiated stream, but the stream MUST be in either the "open" or "half closed (remote)" state with respect to the server. PUSH_PROMISE frames are interspersed with the frames that comprise a response, though they cannot be interspersed with HEADERS and CONTINUATION frames that comprise a single header block.

PUSH_PROMISE可以由服务端在任意由客户端打开的流上发送。他们必须在对服务端状态为“打开”或者“半封闭(远端)”的流上发送。PUSH_PROMISE帧由响应帧穿插组成，不过他们不能由包含单个报头区块的报头帧和延续帧组成。

Sending a PUSH_PROMISE frame creates a new stream and puts the stream into the “reserved (local)” state for the server and the “reserved (remote)” state for the client.

发送一个推送承诺帧创建了一个新的流并且对服务端将流转到“保留(本地)”状态、对客户端转“保留(远端)”的状态。

8.2.2 Push Responses 推送响应

After sending the PUSH_PROMISE frame, the server can begin delivering the pushed response as a response (Section 8.1.2.2) on a server-initiated stream that uses the promised stream identifier. The server uses this stream to transmit an HTTP response, using the same sequence of frames as defined in Section 8.1. This stream becomes "half closed" to the client (Section 5.1) after the initial HEADERS frame is sent.

发送
PUSH_PROMISE帧后，服务端可以开始接收被推送进来的响应作为一个由服务端发起的使用被承诺流标识的流的响应(章节8.1.2.2)。服务端使
用这些流传送一个HTTP响应，使用HttpSequence中定义的相同的帧序列。在初始化报头帧发送后，流对客户端(章节5.1)变为“半封闭”状 态。

Once a client receives a PUSH_PROMISE frame and chooses to accept the pushed response, the client SHOULD NOT issue any requests for the promised response until after the promised stream has closed.

一旦客户端接收到PUSH_PROMISE帧并且选择接受推送的响应，客户端不应该对被承诺的响应发起人和请求，直到被承诺的流被关闭为止。

If the client determines, for any reason, that it does not wish to receive the pushed response from the server, or if the server takes too long to begin sending the promised response, the client can send an RST_STREAM frame, using

either the CANCEL or REFUSED_STREAM codes, and referencing the pushed stream's identifier.

如果客户端以任何理由决定不希望接受服务端推送的响应，或者服务端花费太长时间才开始发送承诺的响应，客户端可以发送一个RST_STREAM帧，使用CANCEL或者REFUSED_STREAM码来关联被推送的流标识符。

A client can use the SETTINGS_MAX_CONCURRENT_STREAMS setting to limit the number of responses that can be concurrently pushed by a server. Advertising a SETTINGS_MAX_CONCURRENT_STREAMS value of zero disables server push by preventing the server from creating the necessary streams. This does not prohibit a server from sending PUSH_PROMISE frames; clients need to reset any promised streams that are not wanted.

客
户端可以使用SETTINGS_MAX_CONCURRENT_STREAMS设置来限制服务端推送的响应的并发量。广播值为0的
SETTINGS_MAX_CONCURRENT_STREAMS能防止服务端创建必需的流。这不能禁止服务端发送PUSH_PROMISE帧；客户端需 要重置任何不需要的被承诺的流。

Clients receiving a pushed response MUST validate that the server is authorized to provide the response, see Section 10.1. For example, a server that offers a certificate for only the example.com DNS-ID or Common Name is not permitted to push a response for https://www.example.org/doc.

客户端收到推送响应必须验证服务端是授权提供响应的，见章节10.1。例如，服务端只带有针对example.com的DNS的授权证书是不允许给https://www.example.org/doc推送给响应的。

The response for a PUSH_PROMISE stream begins with a HEADERS frame, which immediately puts the stream into the “half closed (remote)” state for the server and “half closed (local)” state for the client, and ends with a frame bearing END_STREAM, which places the stream in the "closed" state.

对推送承诺的帧的响应以一个报头帧开始，并马上将流转变成对服务端的“半封闭(远端)”状态以及对客户端的“半封闭(本地)”状态，并且以一个带有END_STREAM的帧结尾，这个将把流转到“关闭”状态。

Note:The client never sends a frame with the END_STREAM flag for a server push.

注意：客户端从不会在服务端推送的时候发送一个带有END_STREAM标记的帧。

8.3 The CONNECT Method CONNECT方法

In HTTP/1.x, the pseudo-method CONNECT ([RFC7231], Section 4.3.6) is used to convert an HTTP connection into a tunnel to a remote host. CONNECT is primarily used with HTTP proxies to establish a TLS session with an origin server for the purposes of interacting with https resources.

在HTTP/1.x中，伪方法连接(RFC7231，章节4.3.6)用来转换HTTP连接成隧道到远端主机。连接主要用HTTP代理为使用HTTPS资源相互作用的目的源服务器建立TLS会话。

In HTTP/2, the CONNECT method is used to establish a tunnel over a single HTTP/2 stream to a remote host, for similar purposes. The HTTP header field mapping works as mostly as defined in Request Header Fields (Section 8.1.2.1), with a few differences. Specifically:

在HTTP/2中，连接方法用来在一个单一的HTTP/2流上建立一个通向远端主机的隧道，目的类似。HTTP报头字段寻址工作大部分同请求报头字段(章节8.1.2.1)中定义，有一部分不同。具体为：

- The :method header field is set to CONNECT.
- The :scheme and :path header fields MUST be omitted.
- The :authority header field contains the host and port to connect to (equivalent to the authority-form of the request-target of CONNECT requests, see [RFC7230], Section 5.3).
- :method是连接中包含:method报头字段。
- :scheme和:path报头字段必须被忽略。
- :authority报头字段包含主机及连接的端口(相当于authority形式的请求目标连接请求，见RFC7230章节5.3)。

A proxy that supports CONNECT establishes a TCP connection [TCP] to the server identified in the :authority header field. Once this connection is successfully established, the proxy sends a HEADERS frame containing a 2xx series status code to the client, as defined in [RFC7231], Section 4.3.6.

支持CONNECT的代理服务器建立一个TCP连接到服务器，如同:authority报头字段中所定义的。一旦连接建立成功，代理发送一个报头帧包含一个2xx序列状态码到客户端，如RFC7231中定义，章节4.3.6。

After the initial HEADERS frame sent by each peer, all subsequent DATA frames correspond to data sent on the TCP connection. The payload of any DATA frames sent by the client are transmitted by the proxy to the TCP server; data received from the TCP server is assembled into DATA frames by the proxy. Frame types other than DATA or stream management frames (RST_STREAM, WINDOW_UPDATE, and

PRIORITY) MUST NOT be sent on a connected stream, and MUST be treated as a stream error (Section 5.4.2) if received.

初始化报头帧由各个对等端发送后，所有随后的与数据对应的数据帧在TCP连接上发送。客户端发送的数据帧载荷由代
理转换再发送给TCP服务器；从TCP服务器接收到的数据由代理组装成数据帧。数据帧或者流管理帧以外的帧(RST_STREAM、
WINDOW_UPDATE和PRIORITY)绝对不能再建立的流上发送，如果收到这样的帧必须作为流错误(章节5.4.2)处理。

The TCP connection can be closed by either peer. The END_STREAM flag on a DATA frame is treated as being equivalent to the TCP FIN bit. A client is expected to send a DATA frame with the END_STREAM flag set after receiving a frame bearing the END_STREAM flag. A proxy that receives a DATA frame with the END_STREAM flag set sends the attached data with the FIN bit set on the last TCP segment. A proxy that receives a TCP segment with the FIN bit set sends a DATA frame with the END_STREAM flag set. Note that the final TCP segment or DATA frame could be empty.

TCP 连接可以被各个对等端关闭。数据帧上的END_STREAM标记被认为与TCP
FIN比特相同。客户端在收到带有END_STREAM标记的帧后被期望应该发送一个带有
END_STREAM标记的数据帧。代理接收到带有END_STREAM
标记的数据帧将在发送这些数据的时候在最后的TCP段上设置FIN位。带有接收到带有FIN位的TCP端发送一个带有END_STREAM标记的数据帧。 注意最后的TCP端或者数据帧可以为空。

A TCP connection error is signaled with RST_STREAM. A proxy treats any error in the TCP connection, which includes receiving a TCP segment with the RST bit set, as a stream error (Section 5.4.2) of type CONNECT_ERROR. Correspondingly, a proxy MUST send a TCP segment with the RST bit set if it detects an error with the stream or the HTTP/2 connection.

TCP连接错误由RST_STREAM来标记。代理对外TCP连接中的任何错误，包括接收到设置了RST位的TCP段，作为类型为连接错误的流错误(章节5.4.2)处理。相应的，代理如果检测到流或者HTTP/2连接的错误必须发送一个设置了RST位的TCP段。

9. Additional HTTP Requirements/Considerations
额外HTTP要求/考虑

This section outlines attributes of the HTTP protocol that improve interoperability, reduce exposure to known security vulnerabilities, or reduce the potential for implementation variation.

这段概况了HTTP协议的属性，包括提高互操作性、减少暴露已知的安全漏洞，或者减少执行变动的可能。

9.1 Connection Management 连接管理

HTTP/2 connections are persistent. For best performance, it is expected clients will not close connections until it is determined that no further communication with a server is necessary (for example, when a user navigates away from a particular web page), or until the server closes the connection.

HTTP/2连接是永久性的。为了最佳的性能，它期待直到确定与服务端的进一步沟通不再必要的时候，客户端才会关闭连接(例如，当用户导航到其他特定的网页)，或者直到服务端关闭连接。

Clients SHOULD NOT open more than one HTTP/2 connection to a given host and port pair, where host is derived from a URI, a selected alternative service [ALT-SVC], or a configured proxy.

客户端不应该再给定的目的地上打开多个HTTP/2连接，目的地是由给定的URI确定的IP地址及TCP端口【这里我们需要小心Alt-Svc】，或者配置的代理的IP和端口。

A client can create additional connections as replacements, either to replace connections that are near to exhausting the available stream identifier space (Section 5.1.1), to refresh the keying material for a TLS connection, or to replace connections that have encountered errors (Section 5.4.1).

客户端可以创建额外的连接作为替代，或者取代快要用尽可用流标识空间(章节5.1.1)的连接，或者替换遇到错误(章节5.4.1)的连接。

A client MAY open multiple connections to the same IP address and TCP port using different Server Name Indication [TLS-EXT] values or to provide different TLS client certificates, but SHOULD avoid creating multiple connections with the same configuration.

客户端可以使用不相同的服务端名称标识值或者提供不一样的TLS客户端证书对相同IP地址及TCP端口打开多个连接，但应该避免对相同的配置上创建多个连接。关于客户端证书相关的更多信息.

Servers are encouraged to maintain open connections for as long as possible, but are permitted to terminate idle connections if necessary. When either endpoint

chooses to close the transport-level TCP connection, the terminating endpoint SHOULD first send a GOAWAY (Section 6.8) frame so that both endpoints can reliably determine whether previously sent frames have been processed and gracefully complete or terminate any necessary remaining tasks.

服务端被孤立尽可能长的保持打开的连接，但在必要下允许关闭空闲的连接。当任意一个终端决定关闭传输层的TCP连接，决定关闭的终端应首先发送一个GOAWAY帧这样两个终端都能可靠的确定之前发送的帧是否已经被处理及优雅的完成或者终止任何必要的剩余任务。

9.1.1 Connection Reuse 复用连接

Clients MAY use a single server connection to send requests for URIs with multiple different authority components as long as the server is authoritative (Section 10.1). For http resources, this depends on the host having resolved to the same IP address.

客户端可以使用单个服务端连接来发送多个不同认证组件的URIs请求，只要服务端是认证的(章节 10.1)。对于http资源来说，这个取决于对同个IP地址已经解析的主机端。

For https resources, connection reuse additionally depends on having a certificate that is valid for the host in the URI. That is the use of server certificate with multiple subjectAltName attributes, or names with wildcards. For example, a certificate with a subjectAltName of *.example.com might permit the use of the same connection for a.example.com and b.example.com.

对于https资源，复用连接还另外需要一个对于URI中的主机已经验
证过的证书。这是多subjectAltName熟悉或者使用通配符的名称的服务端证书的使用。例如，一个带有*.example.com的
subjectAltName的证书将允许a.example.com和b.example.com.使用同个连接。

In some deployments, reusing a connection for multiple origins can result in requests being directed to the wrong origin server. For example, TLS termination might be performed by a middlebox that uses the TLS Server Name Indication (SNI) [TLS-EXT] extension to select the an origin server. This means that it is possible for clients to send confidential information to servers that might not be the intended target for the request, even though the server has valid authentication credentials.

在
某些部署环境中，多个源的复用连接可能导致请求被指向错误的源服务器。例如，代理可能使用TLS服务端名称指示(SNI)[TLS-EXT]扩展选择一个
源服务器来执行TLS终止。这意味着可能客户端未像预期那样发送机密信息到请求的服务器，尽管服务端有验证的权限证书。

A server that does not wish clients to reuse connections can indicate that it is not authoritative for a request by sending a 421 (Not Authoritative) status code in response to the request (see Section 9.1.2).

服务端不希望客户端复用连接的可以在响应(章节9.1.2)中发送一个421(未验证)的状态码来指示请求是未验证的。

9.1.2 The 421 (Not Authoritative) Status Code 421(未验证)状态码

The 421 (Not Authoritative) status code indicates that the current origin server is not authoritative for the requested resource, in the sense of [RFC7230], Section 9.1 (see also Section 10.1).

421(未验证)状态码标识当前源服务器对请求的资源未验证，详见RFC7230，章节9.1(同见章节10.1)。

Clients receiving a 421 (Not Authoritative) response from a server MAY retry the request - whether the request method is idempotent or not - over a different connection. This is possible if a connection is reused (Section 9.1.1) or if an alternative service is selected ([ALT-SVC]).

客户端接收到服务端发送的421(未验证)响应可以在不同的连接上重试请求-
不管这个请求是不是幂等的。这在连接是复用的(章节9.1.1)或可选的服务被选择(ALT-SVC)下是可能的。

This status code MUST NOT be generated by proxies.

这个状态码绝对不能由代理端生成。

A 421 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

421响应缓存是默认的；举例：除非被定义的方法或显示的缓存控制另有说明(RFC7234,
章节4.2.2)。

9.2 Use of TLS Features 使用TLS功能

Implementations of HTTP/2 MUST support TLS 1.2 [TLS12] for HTTP/2 over TLS. The general TLS usage guidance in [TLSBCP] SHOULD be followed, with some additional restrictions that are specific to HTTP/2.

实现HTTP/2必须支持TLS
1.2。通用的TLS用法指导应该遵循，同时加上对HTTP/2的特定支持。

9.2.1 TLS Features TLS功能

The TLS implementation MUST support the Server Name Indication (SNI) [TLS-EXT] extension to TLS. HTTP/2 clients MUST indicate the target domain name when negotiating TLS.

TLS实现必须支持服务端名称标识(SNI)的TLS扩展。HTTP/2客户端再协商TLS的时候必须标明目标域名名称。

The TLS implementation MUST disable compression. TLS compression can lead to the exposure of information that would not otherwise be revealed [RFC3749]. Generic compression is unnecessary since HTTP/2 provides compression features that are more aware of context and therefore likely to be more appropriate for use for performance, security or other reasons.

TLS实现必须禁止压缩。TLS压缩可能导致信息暴露。通用的压缩是不必要的，因为HTTP/2提供的压缩功能更加上下文，因为可能是更符合使用性能、安全或者其他原因。

The TLS implementation MUST disable renegotiation. An endpoint MUST treat a TLS renegotiation as a connection error (Section 5.4.1) of type PROTOCOL_ERROR. Note that disabling renegotiation can result in long-lived connections becoming unusable due to limits on the number of messages the underlying cipher suite can encipher.

TLS实现必须禁用协商。终端必须将TLS协商作为类型为协议错误的连接错误(章节5.4.1)处理。需要注意的是由于密码套件可以加密的消息的次数限制，禁用协商可导致长期连接变成不可用。

A client MAY use renegotiation to provide confidentiality protection for client credentials offered in the handshake, but any renegotiation MUST occur prior to sending the connection preface. A server SHOULD request a client certificate if it sees a renegotiation request immediately after establishing a connection.

客户端可以使用协商来对客户端握手平局提供机密保护，但是任何协商必须在发送连接序言之前。服务端如果看到连接建立后马上协商应该请求客户端证书。

This effectively prevents the use of renegotiation in response to a request for a specific protected resource. A future specification might provide a way to support this use case.

这有效的防止了通过协商来获取一个特定的受保护的资源的请求。未来的规范可能会提供一种方式来支持这种需求。

9.2.2 TLS Cipher Suites TLS加密套件

The set of TLS cipher suites that are permitted in HTTP/2 is restricted. HTTP/2 MUST only be used with cipher suites that have ephemeral key exchange, such as the ephemeral Diffie-Hellman (DHE) [TLS12] or the elliptic curve variant (ECDHE) [RFC4492]. Ephemeral key exchange MUST have a minimum size of 2048 bits for DHE or security level of 128 bits for ECDHE. Clients MUST accept DHE sizes of up to 4096 bits. HTTP MUST NOT be used with cipher suites that use stream or block ciphers. Authenticated Encryption with Additional Data (AEAD) modes, such as the Galois Counter Model (GCM) mode for AES [RFC5288] are acceptable.

HTTP/2
中授权使用的TLS加密套件是不对外公开的。HTTP/2必须仅在支持密钥交换的加密套件下使用，如短暂的Diffie-Hellman（DHE）或椭圆
曲线的变体（ecdhe）。交换的密钥必须具有最小尺寸的2048位（DHE）或128位的安全级别（ecdhe）。客户端必须接受多达4096位DHE
尺寸。HTTP绝对不能使用用流或区块密码的加密套件。使用额外数据(AEAD)模式的认证加密，例如针对AES[RFC5288] 的Galois Counter Model (GCM)模式是允许的。

Clients MAY advertise support of other cipher suites in order to allow for connection to servers that do not support HTTP/2 to complete without the additional latency imposed by using a separate connection for fallback.

客户端可以广播对其他加密套件的支持来允许不支持HTTP/2的服务端的连接能在不需要额外的延迟(因为需要使用单独的连接来降级)下完成。

An implementation SHOULD NOT negotiate a TLS connection for HTTP/2 without also negotiating a cipher suite that meets these requirements. Due to implementation limitations, it might not be possible to fail TLS negotiation. An endpoint MUST immediately terminate an HTTP/2 connection that does not meet these minimum requirements with a connection error (Section 5.4.1) of type INADEQUATE_SECURITY.

实现中对于HTTP/2不应该在没有使用符合
需求的加密套件协商的情况下进行TLS协商。由于实施的限制，不可能基于所有需求来使TLS协商失败。终端必须终止不符合TLS最小需求的TLS会话上建
立的HTTP/2连接，并作为类型为INADEQUATE_SECURITY的连接错误(章节5.4.1)处理。

9.3 GZip Content-Encoding 内部编码Gzip压缩

客户端必须支持HTTP响应体的Gzip压缩。不管接收白头字段的编码的值，服务端可以发送Gzip编码响应。一个压缩的响应还必须承担适当的内容编码报头字段。

10. Security Considerations 安全性考虑

10.1 Server Authority 服务端认证

A client is only able to accept HTTP/2 responses from servers that are authoritative for those resources. This is particularly important for server push (Section 8.2), where the client validates the PUSH_PROMISE before accepting the response.

客户端只有经过权限验证才能获取HTTP/2响应的资源。这在服务器推送中(章节8.2)尤为重要，客户端在接收响应前验证PUSH_PROMISE帧。

HTTP/2 relies on the HTTP/1.1 definition of authority for determining whether a server is authoritative in providing a given response, see [RFC7230], Section 9.1. This relies on local name resolution for the "http" URI scheme, and the authenticated server identity for the "https" scheme (see [RFC2818], Section 3).

HTTP/2依据HTTP/1.1权限定义来检测服务端是否有权限提供给定的响应，见RFC2818章节3.这依赖于本地“HTTP”URI方案的域名解析，以及服务端提供的“https”方案验证。

A client MUST discard responses provided by a server that is not authoritative for those resources.

客户端绝对不能以任何形式使用服务端提供的客户端没有权限的资源。

10.2 Cross-Protocol Attacks 跨协议攻击

In a cross-protocol attack, an attacker causes a client to initiate a transaction in one protocol toward a server that understands a different protocol. An attacker might be able to cause the transaction to appear as valid transaction in the second protocol. In combination with the capabilities of the web context, this can be used to interact with poorly protected servers in private networks.

在跨协议攻击中，攻击者使客户端在一种协议中向解析另一种协议的服务器启动一个交易。攻击者可能能够使交易看起来在第二种协议中是合法的。结合对web上下文的利用，这个可以针对保护不力的服务器在秘密网络下进行互动。

Completing a TLS handshake with an ALPN identifier for HTTP/2 can be considered sufficient protection against cross protocol attacks. ALPN provides a positive indication that a server is willing to proceed with HTTP/2, which prevents attacks on other TLS-based protocols.

在ALPN和TLS合作的情况下对于阻止HTTP/2被协议攻击是足够的。ALPN提供了一个积极的指示说明服务器愿意处理HTTP/2,这有助于阻止基于TLS协议的攻击。

The encryption in TLS makes it difficult for attackers to control the data which could be used in a cross-protocol attack on a cleartext protocol.

TLS中的加密使得攻击者很难控制明文协议中能被用来进行跨协议攻击的数据。

The cleartext version of HTTP/2 has minimal protection against cross-protocol attacks. The connection preface (Section 3.5) contains a string that is designed to confuse HTTP/1.1 servers, but no special protection is offered for other protocols. A server that is willing to ignore parts of an HTTP/1.1 request containing an Upgrade header field in addition to the client connection preface could be exposed to a cross-protocol attack.

HTTP/2明文版本对于跨协议攻击具有很小的保护措施。连接序言(章节3.5)包含一个字符串，是用来迷惑HTTP/1.1服务器的，但是对于其他版本协议没有提供特殊保护。服务端愿意忽略包含升级字段的HTTP/1.1请求部分的可以认为是一次跨协议攻击。

10.3 中介端封装攻击

HTTP/2 header field names and values are encoded as sequences of octets with a length prefix. This enables HTTP/2 to carry any string of octets as the name or value of a header field. An intermediary that translates HTTP/2 requests or responses into HTTP/1.1 directly could permit the creation of corrupted HTTP/1.1 messages. An attacker might exploit this behavior to cause the intermediary to create HTTP/1.1 messages with illegal header fields, extra header fields, or even new messages that are entirely falsified.

HTTP/2报头名称和值编码成带有长度前缀的字节序列。这使得HTTP/2能够携带任何字符串的字节作为报头字段的名称或值。中介端直接转换HTTP/2请求或响应到HTTP/1.1时可以允许HTTP/1.1消息创建的损坏。攻击者可以利用这个行
为让中介端创建HTTP/1.1消息时带有非法报头字段、额外报头字段，甚至是完全伪造的新消息。

Header field names or values that contain characters not permitted by HTTP/1.1, including carriage return (ASCII 0xd) or line feed (ASCII 0xa) MUST NOT be translated verbatim by an intermediary, as stipulated in [RFC7230], Section 3.2.4.

报头字段名称或值带有不被HTTP/1.1允许的字符，包括回车(U+000D)或者换行(U+000A),绝对不能被中介端逐行解析，见[HTTP-p1]定义，章节3.2.4.

Translation from HTTP/1.x to HTTP/2 does not produce the same opportunity to an attacker. Intermediaries that perform translation to HTTP/2 MUST remove any instances of the obs-fold production from header field values.

从HTTP/1.x转换到HTTP/2不会被攻击者利用出现类似的情况。执行转换的中介端必须移除折叠obs的任何实例。

10.4 推送响应的缓存

Pushed responses do not have an explicit request from the client; the request is provided by the server in the PUSH_PROMISE frame.

推送响应并没有一个来自客户端的明确请求；请求是服务端从PUSH_PROMISE帧中提供的。

Caching responses that are pushed is possible based on the guidance provided by the origin server in the Cache-Control header field. However, this can cause issues if a single server hosts more than one tenant. For example, a server might offer multiple users each a small portion of its URI space.

缓存响应推送可能是基于原始服务器的缓存控制报头字段的指导。然而，如果服务端主机包含多个用户可能会导致问题。例如，服务端可能为多个用户每个提供小部分的URI空间。

Where multiple tenants share space on the same server, that server MUST ensure that tenants are not able to push representations of resources that they do not have authority over. Failure to enforce this would allow a tenant to provide a representation that would be served out of cache, overriding the actual representation that the authoritative tenant provides.

当多个用户共享同一台服务器时，该服务器必须确保用户不能推送没有权限使用的资源。如果不能确保这个将导致用户可能提供超出缓存以外的内容，覆盖用户实际有权限提供的内容。

Pushed responses for which an origin server is not authoritative (see Section 10.1) are never cached or used.

没有权限的源服务器(见10.1章节)推送响应将不会被缓存或者使用。

10.5 拒绝服务的注意事项

An HTTP/2 connection can demand a greater commitment of resources to operate than a HTTP/1.1 connection. The use of header compression and flow control depend on a commitment of resources for storing a greater amount of state.

Settings for these features ensure that memory commitments for these features are strictly bounded.

HTTP/2连接可以要求使用比HTTP/1.1连接更大的资源。报头压缩和流量控制的使用取决于承诺的资源存储更大量的状态。这些功能的设置确保这些承诺的内存是严格限制的。处理能力不能在相同的高速缓存中被守护。

The number of PUSH_PROMISE frames is not constrained in the same fashion. A client that accepts server push SHOULD limit the number of streams it allows to be in the "reserved (remote)" state. Excessive number of server push streams can be treated as a stream error (Section 5.4.2) of type ENHANCE_YOUR_CALM.

推送承诺帧的数量并不是以相同方式约束的。客户端收到服务端推送应该限制允许转成“保留(远端)”状态的流的数量。服务端推送流数量超出可以作为类型为ENHANCE_YOUR_CALM的流错误(章节5.4.2)处理。

Processing capacity cannot be guarded as effectively as state capacity.

处理能力不能像状态容量一样被有效的保护。

The SETTINGS frame can be abused to cause a peer to expend additional processing time. This might be done by pointlessly changing SETTINGS parameters, setting multiple undefined parameters, or changing the same setting multiple times in the same frame. WINDOW_UPDATE or PRIORITY frames can be abused to cause an unnecessary waste of resources.

设
置帧可能被滥用导致对等端花费额外的处理时间。这可能是毫无意义改变设置参数、设置多个未定义的参数，或者在同个帧中多次修改同个值。 WINDOW_UPDATE 或
PRIORITY帧也可能被滥用导致资源的不必要的浪费。服务端可能在没有权限为客户端产生过量工作时错误地假定源服务器的ALTSVC帧。

Large numbers of small or empty frames can be abused to cause a peer to expend time processing frame headers. Note however that some uses are entirely legitimate, such as the sending of an empty DATA frame to end a stream.

大量的小或空的贞可能被滥用导致对等端花费额外的时间处理报头帧。但需要注意的是有些使用是完全合法的，例如发送空数据帧到结束流。

Header compression also offers some opportunities to waste processing resources; see Section 8 of [COMPRESSION] for more details on potential abuses.

报头压缩也可能导致处理资源的浪费；见[COMPRESSION]章节8查更多潜在滥用的细节。

Limits in SETTINGS parameters cannot be reduced instantaneously, which leaves an endpoint exposed to behavior from a peer that could exceed the new limits. In particular, immediately after establishing a connection, limits set by a server are not known to clients and could be exceeded without being an obvious protocol violation.

设置参数的限制不能瞬间降低，这使终端对对等端暴露的行为可能超出新的限制。特别是，连接连接后瞬间，服务端设置的限制并不被客户端知道，而且客户端可能在不明显违反协议的情况下超出限制。

All these features - i.e., SETTINGS changes, small frames, header compression - have legitimate uses. These features become a burden only when they are used unnecessarily or to excess.

所有的这些功能，即设置的修改、小帧、报头压缩使用都是合法的。他们只有在不必要或者多余的使用时才会成为负担。

An endpoint that doesn't monitor this behavior exposes itself to a risk of denial of service attack. Implementations SHOULD track the use of these features and set limits on their use. An endpoint MAY treat activity that is suspicious as a connection error (Section 5.4.1) of type ENHANCE_YOUR_CALM.

终端不监测这种行为可能暴露其遭受拒绝服务攻击的风险。具体实现应当跟踪这些功能的使用并限制它们的使用。终端可以对待这些可疑的活动作为类型为ENHANCE_YOUR_CALM的连接错误(章节5.4.1)处理。

10.5.1 Limits on Header Block Size

A large header block (Section 4.3) can cause an implementation to commit a large amount of state. In servers and intermediaries, header fields that are critical to routing, such as :authority, :path, and :scheme are not guaranteed to be present early in the header block. In particular, values that are in the reference set cannot be emitted until the header block ends.

一个较大的报头区块(章节4.3)可以导致具体实现中提交大量的状态。在服务端和中介端，报头字段对路由至关重要，例如:authority, :path,
和:scheme并没有保证在报头区块中靠前呈现。特别是，引用集合中的值只有在报头区块结束才能被提交。

This can prevent streaming of the header fields to their ultimate destination, and forces the endpoint to buffer the entire header block. Since there is no hard limit to the size of a header block, an endpoint could be forced to exhaust available memory.

这可能阻止流的报头字段到达其最终目的地，并强制终端来缓存整个报头区块。由于对报头区块没有硬性规定限制，终端可能被强制耗尽可用内存。

A server that receives a larger header block than it is willing to handle can send an HTTP 431 (Request Header Fields Too Large) status code [RFC6585]. A client can discard responses that it cannot process. The header block MUST be processed to ensure a consistent connection state, unless the connection is closed.

服务端接收到大于它愿意处理的报头区块可以发送一个HTTP 431(请求报头字段过大)状态码RFC6585。终端可以丢弃其不能处理的响应。报头区块必须被处理以确保连续的连接状态，除非连接是关闭的。

10.6 Use of Compression

HTTP/2 enables greater use of compression for both header fields (Section 4.3) and entity bodies. Compression can allow an attacker to recover secret data when it is compressed in the same context as data under attacker control.

HTTP/2允许更大的使用报头字段(见4.3章节)和响应主体(见9.3章节)的压缩。压缩可以使攻击者在相同上下文数据压缩的攻击控制下恢复秘密数据。

There are demonstrable attacks on compression that exploit the characteristics of the web (e.g., [BREACH]). The attacker induces multiple requests containing varying plaintext, observing the length of the resulting ciphertext in each, which reveals a shorter length when a guess about the secret is correct.

明文网络(例如[BREACH])下有针对压缩的明显攻击。攻击者诱导包含不同明文的多个请求，观察各个得到的密文的长度，当密码猜测是正确时就揭示了更短的长度。

Implementations communicating on a secure channel MUST NOT compress content that includes both confidential and attacker-controlled data unless separate compression dictionaries are used for each source of data. Compression MUST NOT be used if the source of data cannot be reliably determined.

具体实现在一个安全通道通信时绝对不能压缩保密的和受攻击者控制的内容，除非对于各个数据源的压缩字典是不同的。绝对不能使用数据源可靠性不确定的压缩数据。

Further considerations regarding the compression of header fields are described in [COMPRESSION].

关于报头uziduan压缩的更多表述参照[COMPRESSION]

10.7 填充的使用

Padding within HTTP/2 is not intended as a replacement for general purpose padding, such as might be provided by TLS [TLS12]. Redundant padding could even be counterproductive. Correct application can depend on having specific knowledge of the data that is being padded.

HTTP/2的填充不打算作为通用填充的替换，如可能在TLS
[TLS12]提供的。多余的填充可能会适得其反。正确的应用依靠的是对于填充的数据有具体的认知。

To mitigate attacks that rely on compression, disabling or limiting compression might be preferable to padding as a countermeasure.

为了减轻依靠压缩的攻击危险，禁用或者限制压缩可能是作为填充的最好对策。

Padding can be used to obscure the exact size of frame content, and is provided to mitigate specific attacks within HTTP. For example, attacks where compressed content includes both attacker-controlled plaintext and secret data (see for example, [BREACH]).

填充可以用来混淆帧内容的实际大小，而且减少HTTP中的特殊攻击。例如，压缩的内容包含攻击者控制的明文和秘密数据的攻击(见BREACH)。

Use of padding can result in less protection than might seem immediately obvious. At best, padding only makes it more difficult for an attacker to infer length information by increasing the number of frames an attacker has to observe. Incorrectly implemented padding schemes can be easily defeated. In particular, randomized padding with a predictable distribution provides very little protection; similarly, padding payloads to a fixed size exposes information as payload sizes cross the fixed size boundary, which could be possible if an attacker can control plaintext.

使用填充可能导致比明显看起来的更少的保护。最好情况下，通过增加攻击者需要观察的帧的数量，填充能使得攻击者更难推
断长度信息。但不正确的实现填充方案可能被轻松破解。特别是，带有可预测分布的随机填充提供保护非常小；或者填充的载荷是一个固定大小的公开信息做未来分
布在固定大小的边界，这可能导致攻击者能控制明文的情况下破解。

Intermediaries SHOULD retain padding for DATA frames, but MAY drop padding for HEADERS and PUSH_PROMISE frames. A valid reason for an intermediary to change the amount of padding of frames is to improve the protections that padding provides.

中介端不应该移除填充，但中介如果是想提高填充的保护可以移除填充添加不同数量的填充量。

10.8 隐私注意事项

Several characteristics of HTTP/2 provide an observer an opportunity to correlate actions of a single client or server over time. This includes the value of settings, the manner in which flow control windows are managed, the way priorities are allocated to streams, timing of reactions to stimulus, and handling of any optional features.

HTTP/2的一些特点导致观察者有机会观察单个客户端或服务器的行为。这包含设置的值、流量控制窗口的管理方式、优先分配流的方法、反应时间以及任何可选功能的处理。

As far as this creates observable differences in behavior, they could be used as a basis for fingerprinting a specific client, as defined in Section 1.8 of [HTML5].

至于这个可观察到的行为的差异，他们可以作为特定的客户端的识别的基础，见HTML5章节1.8中定义。

11. IANA Considerations

A string for identifying HTTP/2 is entered into the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [TLSALPN].

标识HTTP/2的字符串已经收录到TLSALPN中建立的"Application Layer Protocol Negotiation (ALPN) Protocol IDs"注册表中。

This document establishes a registry for frame types, settings, and error codes. These new registries are entered into a new "Hypertext Transfer Protocol (HTTP) 2 Parameters" section.

本文档建立了帧类型、设置和错误码的记录。这些新的记录被收录到新的"Hypertext Transfer Protocol (HTTP) 2 Parameters" 章节中。

This document registers the HTTP2-Settings header field for use in HTTP; and the 421 (Not Authoritative) status code.

本文档记录了HTTP使用的HTTP2-Settings报头字段；以及421(未验证)状态码。

This document registers the PRI method for use in HTTP, to avoid collisions with the connection preface (Section 3.5).

为了避免与连接序言(章节3.5)冲突， 本文档注册了在HTTP中使用的PRI方法。

11.1 HTTP/2识别字符串的注册

This document creates two registrations for the identification of HTTP/2 in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [TLSALPN].

本文档在"应用层协议协商(ALPN)协议IDs"中的[TLSALPN]注册列表创建了两种HTTP/2标识符注册。

The "h2" string identifies HTTP/2 when used over TLS:

Protocol:HTTP/2 over TLS Identification Sequence:0x68 0x32 ("h2") Specification:This document The "h2c" string identifies HTTP/2 when used over cleartext TCP:Protocol:HTTP/2 over TCP Identification Sequence:0x68 0x32 0x63 ("h2c") Specification:This document

使用TLS时“h2”字符串标识HTTP/2:

- 协议: TLS上的HTTP/2
- 标识序列 ： 0x68 0x32 ("h2")
- 定义： 本文档
- "h2c"字符串标识使用明文TCP时的HTTP/2: TCP上的HTTP/2
- 协议： TCP上的HTTP/2
- 标识序列： 0x68 0x32 0x63 ("h2c")
- 定义： 本文档

11.2 Frame Type Registry

This document establishes a registry for HTTP/2 frame types codes. The "HTTP/2 Frame Type" registry manages an 8-bit space. The "HTTP/2 Frame Type" registry operates under either of the "IETF Review" or "IESG Approval" policies [RFC5226] for values between 0x00 and 0xef, with values between 0xf0 and 0xff being reserved for experimental use.

本 文档建立了HTTP/2帧类型码的注册表。"HTTP/2 Frame Type"注册表管理一个8位的空间。" “HTTP/2 Frame Type” 注册表在 “IETF Review” 或者 “IESG Approval”政策下操作0x00 到 0xef的值， 0xf0 到 0xff的值是预留给实验用的。

New entries in this registry require the following information:

Frame Type:A name or label for the frame type. Code:The 8-bit code assigned to the frame type. Specification:A reference to a specification that includes a

description of the frame layout, it's semantics and flags that the frame type uses, including any parts of the frame that are conditionally present based on the value of flags.

这个注册表中新的元素需要提供以下信息：

- 帧类型：帧类型的名称或者标记
- 编码：帧类型的8位编码标识
- 定义：包含帧使用的布局、语义及标记的描述的定义参考规范，包含任何帧根据标记值有条件呈现的部分。

The entries in the following table are registered by this document.

下面表格中的实体已经被本文档所注册。

Frame TypeCodeSection
DATA0x0Section 6.1
HEADERS0x1Section 6.2
PRIORITY0x2Section 6.3
RST_STREAM0x3Section 6.4
SETTINGS0x4Section 6.5
PUSH_PROMISE0x5Section 6.6
PING0x6Section 6.7
GOAWAY0x7Section 6.8
WINDOW_UPDATE0x8Section 6.9
CONTINUATION0x9Section 6.10

11.3 Settings Registry

This document establishes a registry for HTTP/2 settings. The "HTTP/2 Settings" registry manages a 16-bit space. The "HTTP/2 Settings" registry operates under the "Expert Review" policy [RFC5226] for values in the range from 0x0000 to 0xefff, with values between and 0xf000 and 0xffff being reserved for experimental use.

本文档为HTTP/2设置建立了注册 表。"HTTP/2 Settings"注册表管理了一个16位的空间。"HTTP/2 Settings"注册表在"Expert Review"政策(RFC5226)下操作0x0000 到 0xefff的值，0xf000 到 0xffff的值是为实验保留的。

New registrations are advised to provide the following information:

- Name:A symbolic name for the setting. Specifying a setting name is optional.
- Code:The 16-bit code assigned to the setting.

- Initial Value:An initial value for the setting.
- Specification:A stable reference to a specification that describes the use of the setting.

新的注册建议提供以下信息：

- 名称 ： 设置的语义名称。定义一个设置的名称是可选的
- 代码：16位的指定设置的编码
- 初始值：设置的初始值
- 说明：描述设置使用的稳定的参考定义

An initial set of setting registrations can be found in Section 6.5.2.

设置注册表的初始集合可以在6.5.2章节中找到。

NameCodeInitial ValueSpecification
HEADER_TABLE_SIZE0x14096Section 6.5.2
ENABLE_PUSH0x21Section 6.5.2
MAX_CONCURRENT_STREAMS0x3(infinite)Section 6.5.2
INITIAL_WINDOW_SIZE0x465535Section 6.5.2

11.4 错误码注册

This document establishes a registry for HTTP/2 error codes. The "HTTP/2 Error Code" registry manages a 32-bit space. The "HTTP/2 Error Code" registry operates under the "Expert Review" policy [RFC5226].

本文档建立了HTTP/2错误码注册表。“HTTP/2错误码”注册表管理一个32位空间。“HTTP/2错误码”注册表在“专家评审”政策下操作[RFC5226]。

Registrations for error codes are required to include a description of the error code. An expert reviewer is advised to examine new registrations for possible duplication with existing error codes. Use of existing registrations is to be encouraged, but not mandated.

错误码的注册表需要包括错误码的描述信息。专家评审应该检查新的错误代码以防止可能重复现有错误代码。建议使用已有的注册表，但是不是必须的。

New registrations are advised to provide the following information:

- Name:A name for the error code. Specifying an error code name is optional.
- Code:The 32-bit error code value.
- Description:A brief description of the error code semantics, longer if no detailed specification is provided.

- Specification:An optional reference for a specification that defines the error code.

The entries in the following table are registered by this document.

新的注册应提供以下信息：

- 错误码：32位错误码值
- 名称： 错误码名称。指定一个错误码的名称是可选的。
- 描述： 描述错误码的适用条件
- 规范： 定义错误码规范的可选参考。

错误码等级初始表可以在第7章节中找到。

NameCodeDescriptionSpecification
NO_ERROR0x0Graceful shutdownSection 7
PROTOCOL_ERROR0x1Protocol error detectedSection 7
INTERNAL_ERROR0x2Implementation faultSection 7
FLOW_CONTROL_ERROR0x3Flow control limits exceededSection 7
SETTINGS_TIMEOUT0x4Settings not acknowledgedSection 7
STREAM_CLOSED0x5Frame received for closed streamSection 7
FRAME_SIZE_ERROR0x6Frame size incorrectSection 7
REFUSED_STREAM0x7Stream not processedSection 7
CANCEL0x8Stream cancelledSection 7
COMPRESSION_ERROR0x9Compression state not updatedSection 7
CONNECT_ERROR0xaTCP connection error for CONNECT methodSection 7
ENHANCE_YOUR_CALM0xbProcessing capacity exceededSection 7
INADEQUATE_SECURITY0xcNegotiated TLS parameters not acceptableSection 7

11.5 HTTP2设置报头字段注册

This section registers the HTTP2-Settings header field in the Permanent Message Header Field Registry [BCP90].

- Header field name:HTTP2-Settings
- Applicable protocol:http
- Status:standard
- Author/Change controller:IETF
- Specification document(s):Section 3.2.1 of this document
- Related information:This header field is only used by an HTTP/2 client for Upgrade-based negotiation.

本章节注册永久消息报头字段注册[BCP90]中的HTTP2设置报头字段。

- 报头字段名称： HTTP2-Settings

- 应用层协议： http
- 状态： 标准
- 作者/修改操作者：LETF
- 文档定义: 本文档3.2.1章节
- 相关信息： 本字段只在HTTP/2客户端基于升级的协商中使用。

11.6 PRI方法注册

This section registers the PRI method in the HTTP Method Registry ([RFC7231], Section 8.1).

Method Name:PRI Safe : No Idempotent : No Specification document(s) : Section 3.5 of this document Related information: This method is never used by an actual client. This method will appear to be used when an HTTP/1.1 server or intermediary attempts to parse an HTTP/2 connection preface.

本章节注册HTTP方法注册[HTTP-p2]中的PRI方法。

- 方法名称 ： PRI
- - 安全： 否
- 幂等元 ： 否
- 文档定义: 本文档3.5章节
- 相关信息:
  这个方法从不会被确切的客户端使用。该方法只在HTTP/1.1服务端或者中介端试图解析HTTP/2连接序言中使用。

11.7 The 421 Not Authoritative HTTP Status Code

This document registers the 421 (Not Authoritative) HTTP Status code in the Hypertext Transfer Protocol (HTTP) Status Code Registry ([RFC7231], Section 8.2).

- Status Code:421
- Short Description: Not Authoritative
- Specification: Section 9.1.2 of this document

本文档在Hypertext Transfer Protocol
(HTTP)状态码注册表(RFC7231,见章节8.2)中注册了HTTP 421(未验证)状态码。

- 状态码:421
- 简短描述: 未验证权限
- 定义: 本文档章节9.1.2

12. Acknowledgements

This document includes substantial input from the following individuals:

本文档包含以下个人的大量贡献：

- Adam Langley, Wan-Teh Chang, Jim Morrison, Mark Nottingham, Alyssa Wilk, Costin Manolache, William Chan, Vitaliy Lvin, Joe Chan, Adam Barth, Ryan Hamilton, Gavin Peters, Kent Alstad, Kevin Lindsay, Paul Amer, Fan Yang, Jonathan Leighton (SPDY contributors).
- Gabriel Montenegro and Willy Tarreau (Upgrade mechanism).
- William Chan, Salvatore Loreto, Osama Mazahir, Gabriel Montenegro, Jitu Padhye, Roberto Peon, Rob Trace (Flow control).
- Mike Bishop (Extensibility).
- Mark Nottingham, Julian Reschke, James Snell, Jeff Pinner, Mike Bishop, Herve Ruellan (Substantial editorial contributions).
- Alexey Melnikov was an editor of this document during 2013.
- A substantial proportion of Martin's contribution was supported by Microsoft during his employment there.